



SOFTWARE TOOL ARTICLE

Adamant: a JSON schema-based metadata editor for research data management workflows [version 1; peer review: 2 approved]

Ihda Chaerony Siffa , Jan Schäfer , Markus M. Becker 

Leibniz Institute for Plasma Science and Technology (INP), Greifswald, Felix-Hausdorff-Straße 2, 17489, Germany

V1 First published: 29 Apr 2022, 11:475
<https://doi.org/10.12688/f1000research.110875.1>
 Latest published: 19 Jul 2022, 11:475
<https://doi.org/10.12688/f1000research.110875.2>

Abstract

The web tool Adamant has been developed to systematically collect research metadata as early as the conception of the experiment. Adamant enables a continuous, consistent, and transparent research data management (RDM) process, which is a key element of good scientific practice ensuring the path to Findable, Accessible, Interoperable, Reusable (FAIR) research data. It simplifies the creation of on-demand metadata schemas and the collection of metadata according to established or new standards. The approach is based on JavaScript Object Notation (JSON) schema, where any valid schema can be presented as an interactive web-form. Furthermore, Adamant eases the integration of numerous available RDM methods and software tools into the everyday research activities of especially small independent laboratories. A programming interface allows programmatic integration with other software tools such as electronic lab books or repositories. The user interface (UI) of Adamant is designed to be as user friendly as possible. Each UI element is self-explanatory and intuitive to use, which makes it accessible for users that have little to no experience with JSON format and programming in general. Several examples of research data management workflows that can be implemented using Adamant are introduced. Adamant (client-only version) is available from: <https://plasma-mds.github.io/adamant>.

Keywords




Research Data Management, JSON Schema, FAIR Principles






This article is included in the [Research on Research, Policy & Culture](#) gateway.

Open Peer Review

Approval Status 

	1	2	3
version 2 (revision) 19 Jul 2022			 view
version 1 29 Apr 2022	 view	 view	

- Felix Shaw** , Earlham Institute, Norwich, UK
- Frank Krüger** , University of Rostock, Rostock, Germany
- Martin Thomas Horsch** , Norwegian University of Life Sciences, Ås, Norway

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: Ihda Chaerony Siffa (ihda.chaeronymsiffa@inp-greifswald.de)

Author roles: **Chaerony Siffa I:** Conceptualization, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Schäfer J:** Conceptualization, Writing – Review & Editing; **Becker MM:** Conceptualization, Funding Acquisition, Methodology, Project Administration, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: The work was funded by the Federal Ministry of Education and Research (BMBF) under the grant mark 16QK03A granted to the Leibniz Institute for Plasma Science and Technology (INP). The responsibility for the content of this publication lies with the authors.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2022 Chaerony Siffa I *et al.* This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Chaerony Siffa I, Schäfer J and Becker MM. **Adamant: a JSON schema-based metadata editor for research data management workflows [version 1; peer review: 2 approved]** F1000Research 2022, 11:475 <https://doi.org/10.12688/f1000research.110875.1>

First published: 29 Apr 2022, 11:475 <https://doi.org/10.12688/f1000research.110875.1>

Introduction

The demand of funding organizations and publishers to make research data discoverable, accessible, interoperable, and reusable in the sense of the FAIR principles¹ and the associated need for digitization and automation of research data management (RDM) processes pose new challenges for some research fields. In particular, for smaller laboratories in physics departments that operate away from large-scale experiments in astrophysics, or high-energy physics, new processes for research data management must be introduced and established. While the aspects of findability and accessibility can be realized without much effort by means of a data publication on a generic platform, the interoperability and reusability of the (domain-specific) data and metadata often pose a major challenge and requires certain conventions and standards in the communities. There are hardly any common research data management processes in areas such as optics and low-temperature plasma physics, and handwritten laboratory notebooks are still the standard in many places.² On the other hand, electronic laboratory notebook (ELN) systems, repositories, collaborative tools, and (meta-)data standards already exist that can support the implementation of the FAIR principles and Open Science practices. A major challenge is to integrate these tools and standards into the everyday research activities. This is especially challenging when extremely diverse needs have to be addressed at institutions, no specific standard procedures and (meta-)data standards exist, and only a few scientists at a time are dealing with similar instruments and data.

Adamant has been developed to support especially these often small laboratories and departments in the implementation of digital research data management processes and the step-by-step adoption of the FAIR principles. To this end, a tool is provided for the easy use or compilation and creation of domain-specific metadata and metadata schemas based on the widely used JavaScript Object Notation (JSON) schema standard, which recently has been gaining traction in several scientific communities.³⁻⁷ The use of JSON schema in Adamant enables straightforward validation of the metadata ensuring its quality.^{8,9} Furthermore, storing the metadata in JSON format maintains the adaptability of the created metadata with different RDM tools and processes, and increases the findability of the research data to which the metadata is attached, in view of the fact that JSON documents are human- and machine-readable.^{8,10,11} An Application Programming Interface (API) based integration with other systems enables the seamless embedding of Adamant into institutional research data management processes. This positions Adamant directly alongside electronic laboratory notebook systems and makes it suitable for ensuring standards-compliant documentation of scientific studies as early as the planning and execution of experiments.

In its current form, Adamant (v1.0.0) offers various features that can support diverse workflows within RDM activities. The features are as follow:

- Rendering of interactive web-form based on a valid JSON schema
- User-friendly editing process of the rendered web-form and the corresponding schema
- Creating a valid JSON schema and web-form from scratch
- Live validation for various field types
- Quick re-use of existing schemas from a list
- Downloadable JSON schema and its form data
- API-based integration as various form submission functionalities

Methods

This section describes the thought process of selecting the technology stack for the development of Adamant, and the implementation details of the tool's functionalities. Subsequently, detailed descriptions on how to set up and operate Adamant are described. The code is available from [GitHub](#) and is archived with Zenodo.²⁹

Implementation

Technology stack and architecture

Adamant has been developed as a web-based tool. This decision was motivated by the end user experience, where end users can use Adamant directly on a web browser available on their machines and handheld devices with no prior installation and configurations required. We considered several things when choosing the right technology stack for

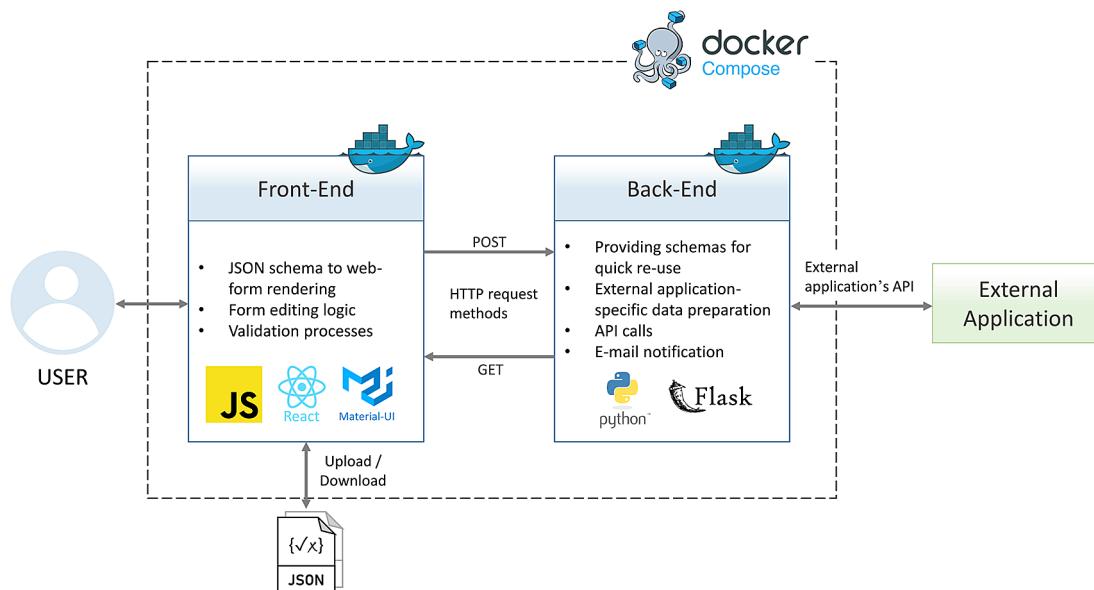


Figure 1. Overview of Adamant's software architecture. API, Application Programming Interface; JSON, JavaScript Object Notation; HTTP, Hypertext Transfer Protocol.

developing Adamant, such as the availability and ease of use of the technology, whether its community is active and big, and the possibility of the technology still being relevant in the coming years, often indicated by its adoption rate and popularity. For these reasons, we adopted a technology stack consisting solely of open-source software, such as ReactJS¹² and Flask¹³ (as the main development frameworks). This stack utilizes two of the most popular programming languages, namely JavaScript and Python¹⁴ (Python Programming Language, RRID:SCR_008394). On top of that, we utilized Docker (Docker Desktop, RRID:SCR_016445) and Docker Compose for straightforward packaging and deployment of the developed tool.¹⁵ Like a typical web tool or application, Adamant consists of front-end and back-end layers. The front-end part serves as the presentation layer in the form of a Graphical User Interface (GUI) for end users to interact with. The back-end part is a server-side part of the tool with the main task of providing processes that are not suitable to be run on the client-side^[1]. Lastly, the front-end and back-end layers are bridged using an API built upon the Hypertext Transfer Protocol (HTTP) request methods, such as `POST` and `GET`.

The front-end has been developed in the Node.js¹⁶ (v14.15.5) JavaScript runtime environment making use of ReactJS (v17.0.2), which is a popular JavaScript library for developing a highly interactive browser-based GUI. ReactJS is combined with the Material-UI library¹⁷ (v4.12.3), which contains many pre-existing user interface (UI) components. The pre-existing components can be modified to one's need straightforwardly thus accelerating the UI development. The main bulk of the features are implemented on the front-end. This includes the automatic rendering process of a JSON schema into a web-form, form editing logic (editing of the rendered web-form or creating one from scratch), and input validation processes. The back-end is written in Python (v3.8.7) using the Flask library (v2.0.2). Flask is a lightweight web framework and straightforward to use making it suitable for the scope of Adamant development. The back-end's tasks include providing the front-end with available JSON schemas from the server, data preparation, necessary e-mail notification for users, and API calls for communication with external applications (API based integration), e.g., ELN and online data repository systems.

Figure 1 provides an overview of Adamant's software architecture and the main functionalities of each application layer along with the corresponding technologies.

Front-end: JSON schema to editable web-form

A JSON schema, like every other JSON document, contains key (word)-value pair instances, where the value parts may contain another set of keyword-value pairs, i.e., nested objects.^{8,10,11} In a JSON schema, there are a number of schema-specific keywords that are mainly used for validation and annotation of the elements in a schema.^{8,9} For example, a typical

¹ In this paper, the terms "front-end" and "client-side" are used interchangeably, as well as "back-end" and "server-side".

set of schema-specific keywords found in a JSON schema and its sub-schema contains `$id/id`, `$schema`, `title`, `type`, and `properties` keywords.⁹ On many occasions, more specific keywords are used to further annotate or describe the elements in a schema. The specific functions of these keywords are dictated by the schema specification version a JSON schema is specified in, which is declared at the very beginning of the schema under the `$schema` keyword. A comprehensive documentation of the JSON schema standard can be found at <https://json-schema.org/>. An example of a JSON schema with the specification version draft 4 is shown in Listing 1, which contains the typical keywords stated earlier, and several field elements. One of the main features of Adamant is to create a web-form representation of a given JSON schema, where users can fill it in and create a JSON document containing the form data for anything they want to describe, e.g., a lab experiment or a dataset. It is worth noting that there are several freely-available libraries and tools for JSON form rendering that conform to the JSON schema specifications.^{18–21} Unfortunately, the available libraries and tools, though some offer direct editing of the schema, do not provide a user-friendly interface for schema (or form) creation and editing right out of the box as what we have envisioned with Adamant. Therefore, we decided to build a custom JSON schema form renderer from the ground up in order to achieve this, which allows us to maintain full control as well as flexibility during the development of Adamant's current and future features. At the time of writing this paper, Adamant (v1.0.0) supports the rendering and editing of JSON schemas with a specification version draft 4 or 7.

Adamant employs a recursive form field rendering process that makes accessing of the keyword-value pairs located within nested objects possible. The rendering process starts by determining the value of the `type` keyword in the uppermost object (or parent object) of the schema. The `type` keyword in this location commonly has a string value of `object`. This value indicates that a `properties` keyword is present within the same location. The `properties` keyword's value is a set of keyword-value pairs (an object), where each pair contains the information that the process needs to render the field. As the process iterates over these keyword-value pairs, it renders a certain form field type according to its `type` keyword's value. The acceptable types in a JSON schema are `string`, `number`, `integer`, `boolean`, `array`, and `object`. If the process stumbles upon a field keyword that has the type of `object` again, then a

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "http://scanning-electron-microscopy",
  "title": "Scanning Electron Microscopy (SEM)",
  "description": "A schema to describe a Scanning Electron Microscopy used in an
    experiment (demo schema)",
  "type": "object",
  "required": ["DeviceModel", "SEMParameters"],
  "properties": {
    "DeviceModel": {
      "title": "Model of SEM Device",
      "description": "SEM device model used in the experiment",
      "type": "string"
    },
    "SEMParameters": {
      "title": "SEM Parameters",
      "description": "SEM parameters used in the experiment",
      "type": "object",
      "properties": {
        "AccelerationVoltage": {
          "title": "Acceleration Voltage [kV]",
          "description": "Voltage applied to accelerate the electrons",
          "type": "number"
        },
        "WorkingDistance": {
          "title": "Working Distance [mm]",
          "description": "Distance from the lens to the sample/specimen",
          "type": "number"
        },
        "ProbeCurrent": {
          "title": "Probe Current [nA]",
          "description": "Electrical current or electron beam focused on the
            sample/specimen",
          "type": "number"
        }
      }
    }
  }
}
```

Listing 1. Example of a draft-4 JSON schema containing typical schema-specific keywords presented in blue with their values presented in black, and field element keywords presented in red. JSON, JavaScript Object Notation.

container will be rendered, and the recursion process is initiated. This process iterates over the current `properties` keyword's value following the same procedure as before, and renders each field within the newly created container. In this way, the process will not terminate until all form fields are rendered. This is a condition where the process does not find any field keyword with the type of `object` anymore. Apart from the `type` keyword's values, the renderer also makes use of other keywords' values to adorn the form field with useful information, such as the values of `title` and `description` keywords (annotation keywords). The main steps of this form field rendering process is also represented in a flowchart diagram as shown in [Figure 2](#).

Another main feature of Adamant is the capability of editing the rendered fields. This feature allows users to change the values of relevant keywords in the schema, and re-order the rendered fields within the same container. To achieve this feature, apart from the rendering of the form fields, each rendered field is also supplied with several editing interfaces along with the editing and input handling logic. Creating a schema (or form) from scratch is another feature made possible by using the same principle of this form editing feature, which basically presents the user with a blank schema and lets them add the field elements as required.

For a certain use case, a file upload functionality can be included in a rendered form. This functionality is intended as an alternative way of enriching the metadata when texts are no longer sufficient. For example, an experiment may have a graphical description of its set-up. In this case, the user can include an image file within the metadata using this file upload functionality. To add this functionality, a string-type field element, with the addition of `"contentEncoding": "base64"` keyword-value pair, must be specified in the schema. This particular keyword-value pair will inform the rendering function to create a file upload field element, where the selected file is read as a string of base64-encoded binary data. Fortunately, adding the file upload functionality can be done in only a couple of mouse clicks in Adamant. For a smooth experience, the uploaded file is recommended to have the size of less than 500 kilobytes.

Lastly, [Table 1](#) presents a complete list of the implemented field types and their relevant keywords in the current version of Adamant (v1.0.0).

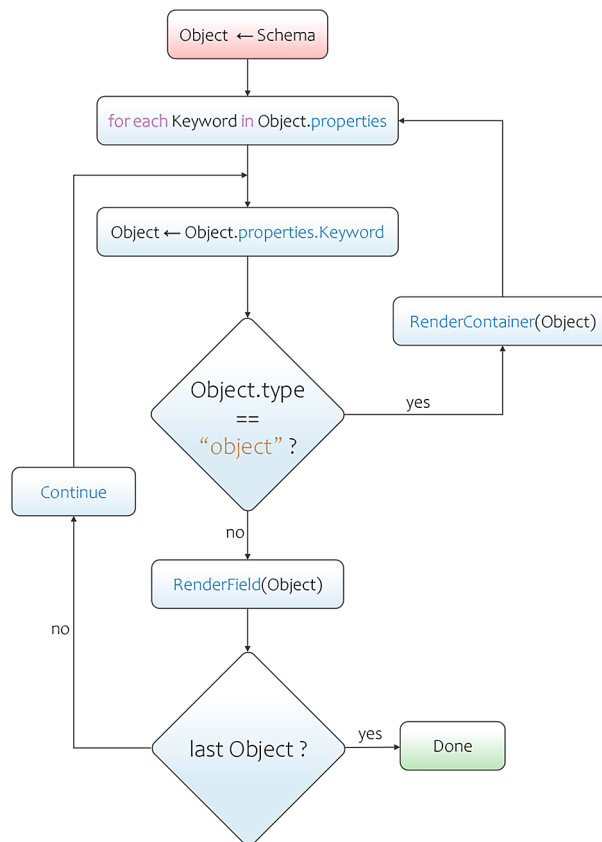


Figure 2. JSON schema to web-form rendering flowchart. JSON, JavaScript Object Notation.

Table 1. Implemented JSON schema field types and their relevant keywords in Adamant v1.0.0. Note that the `id` keyword only works with the JSON schema specification version draft 4, whereas `$id` is used for the newer specification drafts. Lastly, the `contentEncoding` keyword is intended to be used with the specification version draft 7 or newer. JSON, JavaScript Object Notation.

Field type	Implemented keywords	Note
String	<code>title, id, \$id, description, type, enum, contentEncoding, default, minLength, maxLength</code>	<code>contentEncoding</code> can only receive a string value of "base64"
Number	<code>title, id, \$id, description, type, enum, default, minimum, maximum</code>	
Integer	<code>title, id, \$id, description, type, enum, default, minimum, maximum</code>	
Boolean	<code>title, id, \$id, description, type, default</code>	
Array	<code>title, id, \$id, description, type, default, items, minItems, maxItems, uniqueItems</code>	
Object	<code>title, id, \$id, description, type, properties, required</code>	

Front-end: JSON form data validation using Ajv

As the user fills in the form, a JSON document containing the entered data is created and updated accordingly and simultaneously, this JSON document is called JSON form data. Upon finishing the form filling process, the created JSON form data can be used for other operations. To ensure the correctness or quality of the entered data, the JSON form data is validated against its schema prior further operation. For this validation purpose, Adamant takes the advantage of the Ajv library (v8.8.2).²² By using this library, any discrepancies between the JSON form data and its schema can be identified. For example, if a field is required to be filled and the user does not provide any input, then the process will throw a validation error with relevant error messages. Many other keyword specific validations are included in this library, which helps with ensuring the quality of the form data with respect to its schema.

Back-end: API-based integration

Many existing web applications provide APIs for seamless integration with other (external) web tools or applications. For instance, web applications that are commonly used in the RDM community include online data repository systems, ELNs, collaborative and version control applications, etc. These applications, more often than not, provide API endpoints that are straightforward to use. Several web tools have also been developed each addressing certain challenges in RDM, such as COPO,³ and Dendro.²³ Adamant is equipped with a back-end layer whose main purpose is to ensure the possibility of seamless integration with such web applications. The back-end is written in Python (v3.8.7) using the Flask library (v2.0.2), which is advantageous given numerous web applications provide Python libraries for their API calls. The connection between Adamant's front-end and back-end is achieved primarily using the `POST` and `GET` HTTP request methods. The `POST` method is used to send data from the front-end to the back-end, which is then pre-processed (if needed) and relayed to the external application (using the provided API). The `GET` method is used when the client-side requires information or data only available in the server-side or from an external application, e.g., when retrieving schemas from the server, or retrieving tags and database items from an external application. With this straightforward approach, various functionalities can be implemented in the server-side that can help with realizing application-specific RDM workflows.

Operation

Many features of Adamant can be explored directly in the client-only version available online at <https://plasma-mds.github.io/adamant>. This version only lacks the submission-related features, which require the server-side to operate. To access the full features, Adamant can be set up and deployed on a local machine or a server that allows for running both the client and server sides. In most instances, Adamant is not resource intensive, and can be used on a typical smartphone and laptop. However, for deployment and development, we recommend to set up Adamant on a system with at least a 64-bit CPU (with 4 cores) and 8GB of RAM. At the time of writing, Adamant has been tested on Firefox (91.5.1esr), Chrome (v97.0.4692.99), and Chrome Mobile (v97.0.4692.98) web browsers. The following subsections introduce the deployment process of Adamant and how users can interact with it.

Setting up Adamant on a local machine

For development purpose, Adamant can be set up on a local machine. For a Linux system (tested on Ubuntu 20.04.4 LTS), the following steps can be taken for this:

1. `$ git clone https://github.com/plasma-mds/adamant.git` — clone the repository
2. `$ cd adamant` — go to adamant project directory
3. `adamant$ npm install` — install the dependencies for the client-side
4. `adamant$ cd backend` — go to backend directory
5. `adamant/backend$ python3 -m venv venv` — create a python virtual environment
6. `adamant/backend$ source ./venv/bin/activate` — activate the virtual environment
7. `adamant/backend$ pip install -r requirements.txt` — install the dependencies for the back-end
8. `adamant/backend$./venv/bin/flask run --no-debugger` — start the back-end
9. Start a new terminal
10. `adamant$ npm start` — on the new terminal, in the adamant project directory, start the client-side

Typically, a web-browser presenting the client-side will open automatically following the last command. In any case, Adamant can be accessed at <http://localhost:3000>.

Deploying Adamant using Docker

Adamant is intended to be deployed on an institutional server in the internal network. In this way, anyone who is connected to the institute’s network can use Adamant directly. We recommend using Docker to deploy the production build of Adamant, which can be done with the following steps:

1. `$ git clone https://github.com/plasma-mds/adamant.git` — clone the repository
2. `$ cd adamant` — go to adamant project directory
3. `adamant$ docker-compose build` — build the docker images for both back-end and front-end
4. `adamant$ docker-compose up -d` — start both client and server containers, i.e., the whole system

By default, the deployed system can be accessed at <http://localhost:3000>.

General overview of the Adamant UI

The general overview of the Adamant UI is presented in [Figure 3](#). Adamant renders a given JSON schema into a web-form by uploading the schema from a local drive using the “BROWSE SCHEMA” button, or if already existing, the schema can be selected from a list next to the browse button. Creating a JSON schema from scratch is possible by clicking the “CREATE FROM SCRATCH” button. After uploading or selecting the schema, the schema is checked by Adamant to see whether the schema file type and structure are valid. If the schema is valid, the schema can be rendered by clicking the “RENDER” button. The rendering can be undone by clicking the “CLEAR” button, which also discards the current schema.

As a demonstration, [Figure 3](#) shows the rendered web-form based on the schema represented in [Listing 1](#). Each rendered field is shown along with its editing interfaces, and the field container’s header is rendered in blue for a quick

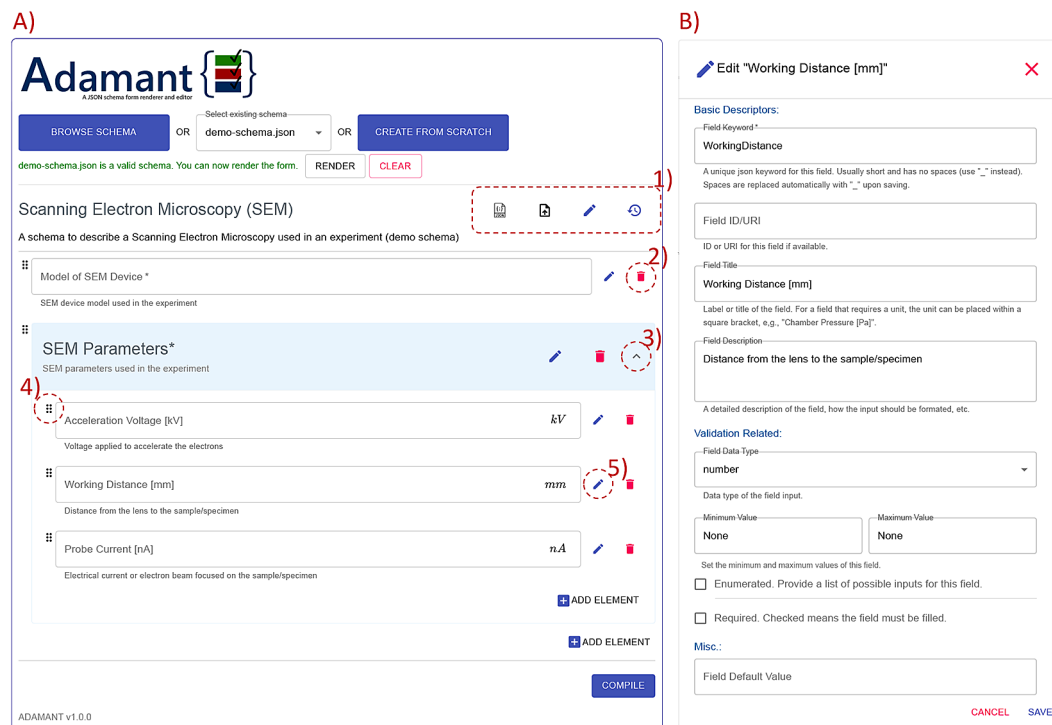


Figure 3. Overview of the Adamant UI with a rendered web-form based on the schema in Listing 1 as an example. (A) Main corpus of the UI; (1) from left to right: JSON schema viewer, auto-populate form, edit schema description, revert all changes; (2) remove form field; (3) collapse or expand the field container; (4) field drag handle; (5) edit field description and (B) field editing panel (as a pop-up on top of the main UI) triggered by clicking (5) the edit button. UI, user interface; JSON, JavaScript Object Notation.

identification. Furthermore, the field container can be expanded and collapsed for a good user experience. The editing interfaces found in each field consists of edit and remove button icons, and a drag handle icon. These interfaces allow the user to change the values of relevant keywords in the schema (directly affecting the rendered field), and re-order the rendered fields by dragging and dropping the field to the desired order within the same container. The changes are reflected to the rendered form immediately, which promotes a What You See Is What You Get (WYSIWYG) application experience for the users. On top of that, each field container is equipped with the “ADD ELEMENT” button for adding a new field within the corresponding container. By default, Adamant renders the form in the edit mode right after a schema is rendered. The editing mode can be concluded by pressing the “COMPILE” button, which removes all editing interfaces (thus also removes the editing functionalities) and readies the form for use. In case of needing to edit the form again, the user is able to initiate the editing mode again, and without losing the already entered data.

Adding and removing schemas

A number of schemas can be fixed in the back-end. The back-end serve these stored schemas to the front-end, from which the end users can select. This is important for a quick re-use of the schemas, especially the ones that are regularly used. For this purpose, one can simply add the desired schemas to `adamant/backend/schemas/` directory. Likewise, a schema can be removed from this directory. The list of the schemas in the front-end is updated every time the front-end is refreshed.

Use cases

At its core, Adamant is a JSON schema form renderer-editor and JSON form data creator presented in an interactive and user-friendly UI. It can be operated by anyone with no prior knowledge of JSON format and coding. This section elaborates the general and *ad hoc* use cases of Adamant that it can be suited to.

Collection of structured and standardized metadata

The creation or use of a valid JSON schema and the collection of corresponding form data being consistent with the schema can be considered as the general use case of Adamant, which simplifies the collection of structured and

standardized metadata. This is particularly relevant for laboratories that are not embedded in large research clusters providing access to established standards and digital research data management workflows. On the one hand, the ability to define required metadata fields and formats supports metadata quality assurance, and on the other hand, storing metadata in JSON format enables further (automated) processing of metadata. This both directly contributes to the “FAIRness” of metadata.

Creation of an eLabFTW experiment with schema-compliant metadata

The use case introduced here aims to highlight how the previously described ability to interface with external systems through API-based integration can be used to generate schema-compliant experiment descriptions in eLabFTW. The open source ELN system eLabFTW^{24,25} (eLabFTW, RRID:SCR_013971) is becoming increasingly popular, especially at universities and research institutions. Given Adamant’s features (JSON schema-based metadata creation, input validations, etc.), one can see why it could be beneficial to collect structured metadata using Adamant rather than doing it directly in the ELN system, especially when the ELN system is designed to be as general as possible in order to accommodate various kinds of experiments from different scientific fields. Note that eLabFTW already offers the possibility to use predefined templates and to edit JSON files. However, a user-friendly possibility to work directly with JSON schemas and to validate the entered metadata on the basis of these predefined schemas is missing so far.

Figure 4 shows the workflow for creating an experiment using Adamant in the eLabFTW system from the end-user’s perspective. The user generates the experiment form by uploading their experiment schema, or selecting it from the list if it already exists. Alternatively, the user can create the form from scratch. If necessary, the user can edit the form (affecting the given schema). For example, changing the title of a certain field, removing a field, etc. When satisfied, the user can compile the rendered form and proceed to fill it in. Upon completing the form, Adamant validates the entered data against the given schema and notifies the user if discrepancies between the entered data and the schema are found, which the user can rectify thereafter. When the entered data are valid, the user can proceed to review the form. After a thorough review, the user can proceed and submit the form. During the submission process, the user will be prompted to input the URL of their eLabFTW system and their eLabFTW API token. Optionally, the user can provide the title and the tags for their experiment. Upon submission, Adamant’s back-end prepares the content and creates a new experiment in eLabFTW

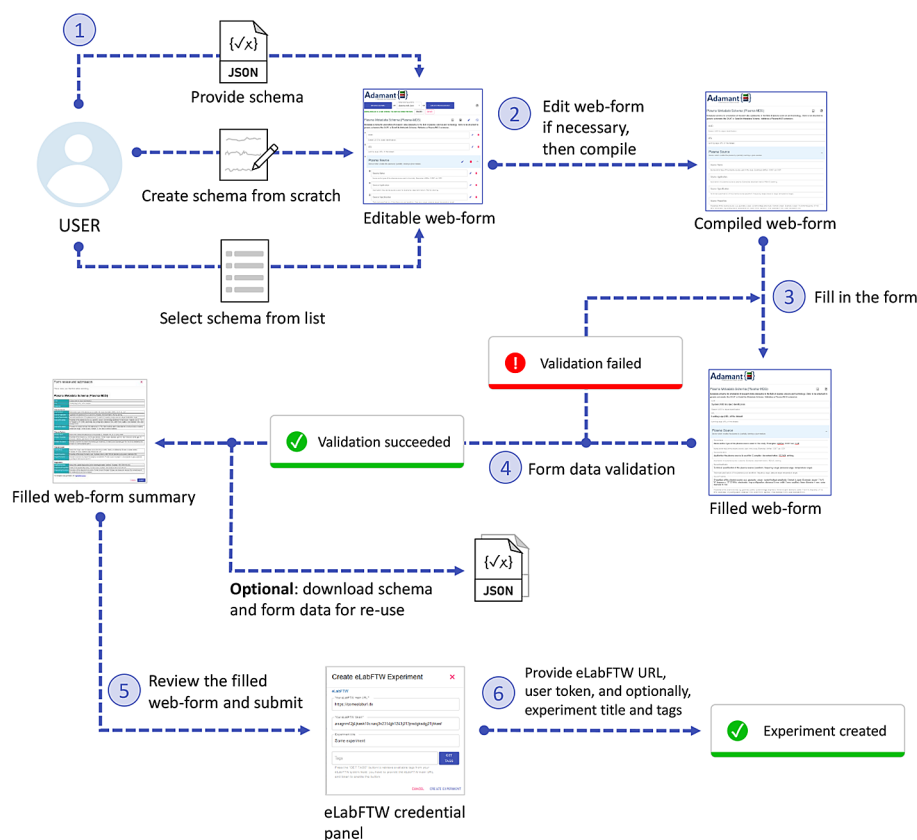


Figure 4. Workflow for creating a new experiment in the eLabFTW system with Adamant.

accordingly, making use of the available eLabFTW API for python, namely elabapy (v0.8.2).^{26,27} The user will be notified upon a successful submission, and the created experiment can be viewed in the eLabFTW system.

The eLabFTW system (v4.2.2) makes use of the TinyMCE text editor²⁸ for free text descriptions of experiments. Here, HTML content can be loaded and rendered. Based on this functionality, the present implementation of the described workflow generates an HTML description list content based on the submitted JSON form data and its schema. The title of the field (obtained from the schema) is used and paired with the value of this field, as shown in [Figure 5\(a\)](#), where the titles are encapsulated within the `<dt>` tags, the values within the `<dd>` tags, and the complete pairs in the `<dl>` tags. The description list is rendered in a way to attain or simulate the look of a form, as presented in [Figure 5\(b\)](#), which increases the readability of the content. This result is achieved by using a custom Cascading Style Sheets (CSS) styling, which can be provided to the eLabFTW system. The generation of this description list content is carried out on Adamant's front-end side. Note that it is equally possible to upload the experiment information prepared in Adamant directly to the eLabFTW experiment in JSON format. However, this would present it in the form of additional information to the experiment description. Since we consider the structured metadata to be the main component of the experiment documentation, the described workaround based on the HTML description lists in the main body has been chosen. Still, the schema, JSON form data, and uploaded files (if available) are sent as attachments to the eLabFTW experiment. In this way, the schema and form data that describe the experiment are preserved and available for re-use, while retaining the high readability of the experiment body for the user.

Submission of metadata for research lab workflow

Research institutions are often equipped with advanced laboratory instruments, such as Scanning Electron Microscopy (SEM) and Mass Spectrometry (MS) instruments just to name a few. These instruments are pivotal for various research investigations in many scientific fields. However, not every researcher at the institute is able to operate and has access to such instruments. To this end, we propose a job request workflow using Adamant that also incorporates the use of eLabFTW for experiment documentation.

The job request workflow represented in [Figure 6](#) aims to streamline the process of requesting a job or an investigation of an object of interest using a laboratory instrument that needs to be carried out by the designated instrument operator. From Adamant's perspective, this workflow involves two users, namely the researcher who wants to get something done with an instrument they do not know how to use (or do not have access to), denoted as the "requester", and the instrument operator, denoted as the "operator". From the documentary point of view, the requester provides the information required to execute the job, while the operator can add metadata related to the instrument and the analysis method. Hence, the schemas used for this workflow can be divided into two, the request schema and the experiment schema. The request schema is used by the requester, and is a subset of the experiment schema including only the form fields for the request details. The experiment schema is used by the operator, which contains every form field necessary to describe the experiment using the selected instrument. Examples of a request schema and its corresponding experiment schema (complete schema) are available [here](#) and [here](#), respectively. Both users are notified automatically per e-mail for relevant events within the workflow, such as when the requester submits the job request and the operator starts working on the request. The e-mail notifications are made sure to work accordingly by including the necessary information of the requester and operator within the used schemas, and by setting up the e-mail notification configuration for the relevant schemas. An example of the e-mail notification configuration file can be found in `adamant/backend/conf/`, in which each configuration parameter is explained. For using more than one job request workflow, one can simply add another schema-specific configuration into the `confList` keyword. The right configuration is then automatically selected based on the relevant schema titles.

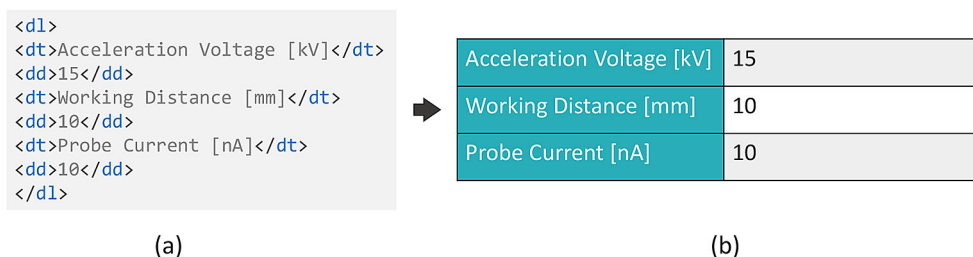


Figure 5. HTML description list rendering in eLabFTW. (a) Title-value pairs in the HTML description list format, and (b) the rendered description list with a custom CSS styling. CSS, Cascading Style Sheets.

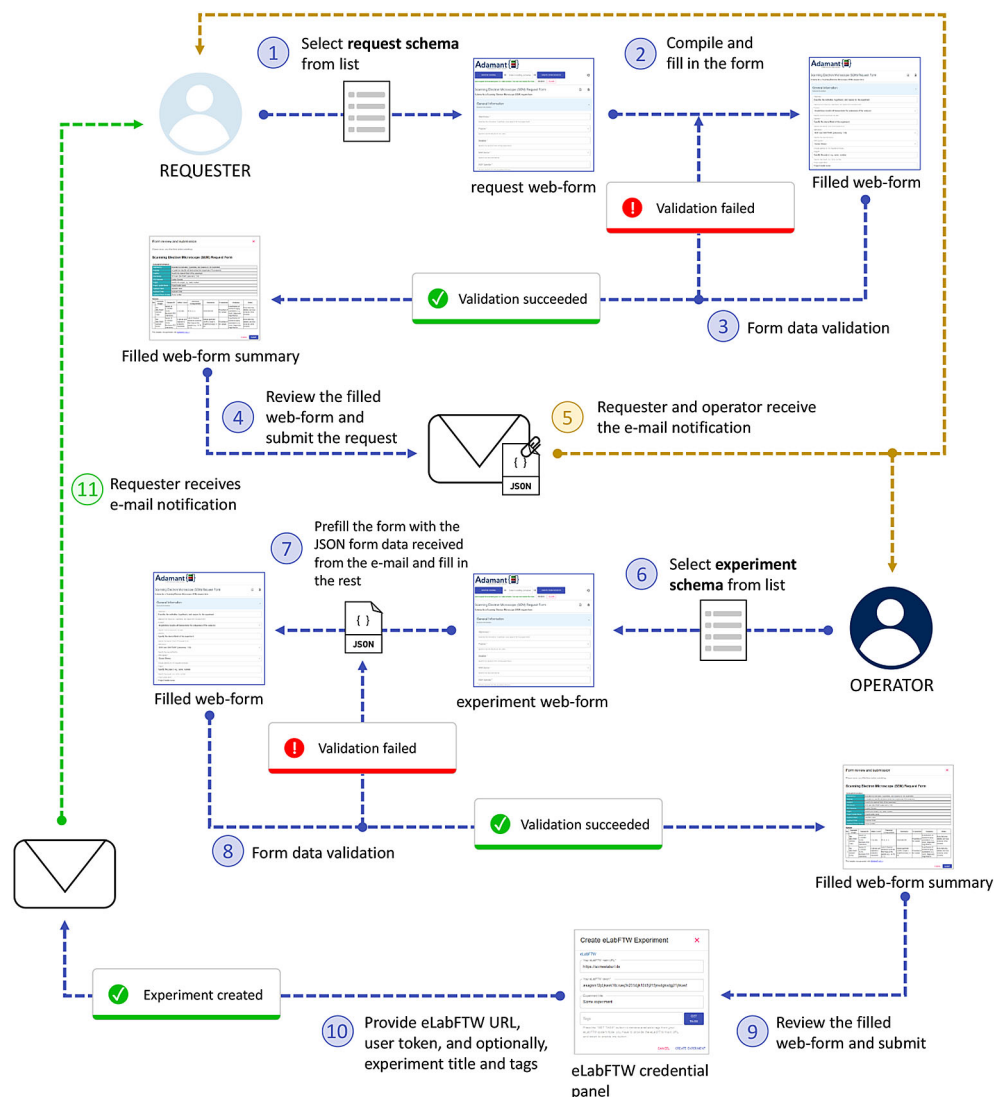


Figure 6. Job request workflow using Adamant involving two users: a requester and an operator. JSON, JavaScript Object Notation.

Conclusions

We have developed a web-based tool named Adamant with the aim of easing the implementation of digital research data management processes. With this, we intend to support the adoption of the FAIR data principles in independent labs, which do not (yet) have access to established research data management infrastructures and domain-specific standards. Adamant provides straightforward compilation and creation of metadata and metadata schemas based on the JSON schema standard, and API-based integration with other available research data management software tools. The intuitive and self-explanatory UI of Adamant increases the accessibility for users with little to no experience with JSON format and programming in general. A flagship implementation of Adamant is the use of the tool, generally, in a core facility or laboratory of a research institute that often hosts advanced scientific equipment, such as the scanning electron microscopy instrument. This is often an advanced expertise that different teams want to use, while a small competence group (experts in such an instrument) ensures that the investigation tasks are optimally processed. Such a use case is demonstrated in the Adamant's submission of metadata for research lab workflow. With that as an example, Adamant can be suited to many specific use cases by extending it with relevant submission functionalities.

Data availability

Underlying data

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

- Adamant (client-only version) available from: <https://plasma-mds.github.io/adamant>
- Source code available from: <https://github.com/plasma-mds/adamant>
- Archived source code at time of publication: <https://doi.org/10.5281/zenodo.6396182>²⁹
- License: [MIT](#)

Acknowledgements

The authors would like to thank Nick Plathe for their helpful suggestions and Laura Vilardell Scholten for developing the description list CSS code.

References

1. Wilkinson MD, Dumontier M, Aalbersberg JJ, *et al.*: **The FAIR guiding principles for scientific data management and stewardship**. *Sci. Data*. 2016; **3**: 160018. [PubMed Abstract](#) | [Publisher Full Text](#)
2. Israel H, Tobschall E, Tristram F: **Dataset for the publication "Umfrage zum Forschungsdatenmanagement in der Physik"**. 2021. [Publisher Full Text](#)
3. Shaw F, Etuk A, Minotto A, *et al.*: **COPO: a metadata platform for brokering FAIR data in the life sciences [version 1; peer review: 1 approved, 1 approved with reservations]**. *F1000Res*. 2020; **9**: 495. [Publisher Full Text](#)
4. Franke S, Paulet L, Schäfer J, *et al.*: **Plasma-MDS, a metadata schema for plasma science with examples from plasma technology**. *Sci. Data*. 2020; **7**: 439. [Publisher Full Text](#)
5. P.-K. consortium: **PDBe-KB: a community-driven resource for structural and functional annotations**. *Nucleic Acids Res*. 10 2019; **48**: D344–D353. [Publisher Full Text](#)
6. The Human Cell Atlas Metadata Standards: JSON Schemas: 2022. Accessed: 2022-04-12. [Reference Source](#)
7. Metadata-Schemas-for-Materials-Science: 2022. Accessed: 2022-04-12. [Reference Source](#)
8. Pezoa F, Reutter JL, Suarez F, *et al.*: **Foundations of JSON Schema**. *Proceedings of the 25th International Conference on World Wide Web*. 2016; pp. 263–273. International World Wide Web Conferences Steering Committee. [Publisher Full Text](#)
9. Droettboom M, *et al.*: **Understanding JSON Schema**. 2022. Accessed: 2022-02-04. [Reference Source](#)
10. Introducing JSON: 2022. Accessed: 2022-03-01. [Reference Source](#)
11. Bray T: **The JavaScript Object Notation (JSON) Data Interchange Format**. RFC 7158, 2014.
12. ReactJS API: 2022. Accessed: 2022-02-04. [Reference Source](#)
13. Flask: 2022. Accessed: 2022-02-04. [Reference Source](#)
14. Van Rossum G, Drake FL: *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace; 2009.
15. Merkel D: **Docker: lightweight linux containers for consistent development and deployment**. *Linux Journal*. 2014; **2014**(239): 2.
16. Node.js: 2022. Accessed: 2022-04-12. [Reference Source](#)
17. Material-UI: 2022. Accessed: 2022-02-04. [Reference Source](#)
18. react-jsonschema-form: 2022. Accessed: 2022-02-04. [Reference Source](#)
19. jsonform: 2022. Accessed: 2022-02-04. [Reference Source](#)
20. jsonforms.io: 2022. Accessed: 2022-02-04. [Reference Source](#)
21. json-editor: 2022. Accessed: 2022-02-04. [Reference Source](#)
22. Ajv JSON schema validator: 2022. Accessed: 2022-02-04. [Reference Source](#)
23. Rocha da Silva J, Aguiar Castro J, Ribeiro C, *et al.*: **Dendro: Collaborative research data management built on linked open data**. *The Semantic Web: ESWC 2014 Satellite Events*. Presutti V, Blomqvist E, Troncy R, *et al.*, editors. Cham: Springer International Publishing; 2014; pp. 483–487. [Publisher Full Text](#)
24. eLabFTW: a free and open source electronic lab notebook: 2022. Accessed: 2022-02-04. [Reference Source](#)
25. Carpi N, Minges A, Piel M: **eLabFTW: An open source laboratory notebook for research labs**. *J. Open Source Softw*. 2017; **2**(12): 146. [Publisher Full Text](#)
26. eLabFTW API: 2022. Accessed: 2022-02-04. [Reference Source](#)
27. elabapy at PyPI: 2022. Accessed: 2022-02-04. [Reference Source](#)
28. TinyMCE: 2022. Accessed: 2022-02-04. [Reference Source](#)
29. Chaerony Siffa I, Schäfer J, Becker MM: **plasma-mds/adamant: Adamant Initial Release v1.0.0 (v1.0.0)**. *Zenodo*. 2022. [Publisher Full Text](#)

Open Peer Review

Current Peer Review Status:  

Version 1

Reviewer Report 07 June 2022

<https://doi.org/10.5256/f1000research.122529.r136576>

© 2022 Krüger F. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Frank Krüger 

Institute of Communications Engineering, University of Rostock, Rostock, Germany

The FAIR Guiding Principles require, among other things, the provision of rich metadata for research data and the generating processes. For this purpose, structured information that reuses existing vocabulary in a standardized way is desired. While such thorough documentation improves the reusability of the research data, it demands knowledge about appropriate vocabularies (e.g. DCAT, DDI...) and some minimum technical understanding of structured documentation (e.g. JSON, JSON-LD, RDF...) of researchers from all scientific domains. Furthermore, mechanisms to ensure the provision of some minimal required information are necessary. These requirements increase the complexity of providing FAIR data and create the need for appropriate tool support.

Adamant is a web-based tool for the provision of structured metadata which guides the researcher through the documentation process and provides a means for validation of the content and completeness by providing a schema-based form. The underlying schema can either be loaded from a library or particularly designed for the research process at hand. By providing a rich API, Adamant satisfies further requirements for the integration into complex workflows based on virtual research environments.

From the technical perspective, Adamant employs JSON Schema, a technology to specify the structure of JSON documents including required and optional information. A web-form, based on Python/Flask, is generated from the schema that is presented to the end-user to provide the necessary information. Adamant is publicly hosted for testing purposes and can easily be deployed by using the provided docker image. All code is available from github and archived at Zenodo. The released version worked without any problems for me. I tested the web-form and the integration with elabFTW.

While I like Adamant and the article very much, I have a few suggestions/questions:

1. Adamant is based on JSON Schema and JSON, but recently I recognized an increase in RDF-based (meta)data management. I was wondering if RDF/S and SHACL would also be a viable base for Adamant and why the authors didn't employ those.

2. I further agree with Felix Shaw (Reviewer 1) about the integration of ontologies for the structured and semantic documentation of research data.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Data Science, Provenance, Metadata

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 12 Jul 2022

Ihda Chaerony Siffa, Leibniz Institute for Plasma Science and Technology (INP), Greifswald, Germany

We thank Frank Krüger for their valuable comments. The following are our responses to each individual comment:

- Response for comment 1:** From our understanding, it is feasible to use RDF(S) formats and SHACL for generating web-forms and validation. Our arguments on using JSON and JSON Schema instead of RDF(S) and SHACL are (in no particular order):
 - RDF(S)/SHACL are ontology-related standards that are surely the means of choice in cases where well-defined ontologies already exist. Note that Adamant specifically targets use cases for which ontologies or metadata standards do not yet exist.
 - JSON Schema supports a set of data types that are analogous to those found in many programming languages (e.g., Python, JavaScript). This allows

straightforward implementation of specific input types and their validations.

- To the best of our knowledge, the JSON format is the de-facto standard of API data interchange. Since Adamant can be used with different external applications by means of API integration, describing the metadata and its schema in the JSON format seems sensible.
- Many existing metadata schemas are available in the JSON format.
- On top of that, we plan on using ontologies, serialized in an RDF format, to describe the schema elements that are used in JSON schemas. A Uniform Resource Identifier (URI) can be assigned to each schema element in the JSON format, which allows the interoperability of these schema elements across both formats.

2. **Response for comment 2:** Please see the response under Reviewer 1's comments.

Competing Interests: No competing interests were disclosed.

Reviewer Report 03 May 2022

<https://doi.org/10.5256/f1000research.122529.r136392>

© 2022 Shaw F. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Felix Shaw 

Data Infrastructure and Algorithms, Earlham Institute, Norwich, UK

Standards for experimental metadata are in most cases sub-optimal or non-existent. Being able to describe protocols, samples, instruments, etc. in a consistent way is vital to the reusability of data and the reproducibility of work. This situation will only continue with the prevalence of machine learning techniques in science which rely heavily on labeled data. Adamant is a web tool for authoring generic experimental assays and then filling out such assays. Users are presented with an interface in which they can add fields of different data types, and various input restrictions (e.g. enumerations, character lengths, etc.). The system then creates a JSON document that records these fields and can be used later for validation and display. Users can then fill out these generated forms with real assay data which is validated against the JSON document, and in turn, produces another JSON document with the assay data itself.

This is a good tool, is well presented, and is easy to use. In functionality, it seems comparable with the ISATools suite of software, although less mature and (arguably) more intuitive. It uses industry-standard software (Python, Flask, React, etc.) and is under an MIT license on Github,

meaning anyone can download and modify it. This means if uptake is good, there is a fair possibility of community maintenance. There are precious few tools for authoring metadata schemas, and this is a welcome step in the right direction. I downloaded and tested the docker version and it worked without any problem. I didn't download and run the development version.

There are some things I would like to see:

- Creating metadata standards is in my experience very much a community effort. Whilst this tool makes the "process" of making metadata standards easy, there needs to be an emphasis on community agreed standards. Otherwise, we risk the proliferation of a multitude of standards that fit only very niche applications.
- Talking of community agreed standards, it would be nice to have a large selection of existing standard schemas available for users to select, and then possibly edit. This would be a big time-saver for a lot of researchers.
- There doesn't seem to be a way of using ontology terms for field names, values, or units. Ontologies are the best way of disambiguating metadata, and services such as EMBL/EBI's Ontology Lookup Service make real-time searching for ontology terms easy. I would like to see this implemented in the authoring stage.
- I have a feeling this will not scale particularly well. It's fine to fill out a web form for a few samples or assays, but in practice, researchers will have hundreds or thousands of records. Filling out web forms on this scale is impracticable. I would like to know the author's views on how to overcome this issue.
- Whilst the authors give the use case of submitting an assay to eLabFTW, it would be nice to see some other integrations, such as with Dataverse, Dspace, or CKAN repositories, which are often used by smaller research facilities (the apparent target audience for this work).

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Data Science, Open Science, Metadata Standards, Web Development

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 12 Jul 2022

Ihda Chaerony Siffa, Leibniz Institute for Plasma Science and Technology (INP), Greifswald, Germany

We thank Felix Shaw for their insightful and constructive comments and helpful suggestions. Some of the comments are in line with what we already have in mind for further development of the tool which we will include in the next version of the manuscript as an outlook. Nevertheless, we will still attempt to address each individual comment briefly as follows:

- 1. Response for comment 1:** We definitely agree and have experienced this matter first hand. At the very early stage, Adamant is intended to be used to create and design the initial experiment schema, which later can be shared/made public in a publicly available collaborative version control platform (like GitHub). Once the schema is made public, the community can participate in the further development and maintenance of the said schema.
- 2. Response for comment 2:** Thank you very much for this suggestion. Besides the public maintenance of already available schemas for the domain of plasma technology (see our community effort at <https://www.plasma-mds.org>) it is indeed an interesting idea to provide easy access to established standards like Dublin Core, DataCite, etc., via Adamant. We will think further in that direction.
- 3. Response for comment 3:** We agree with the comments regarding ontologies and we have already thought of something in this direction. We will definitely implement features related to the use of ontologies in the future release of Adamant. One thing that we have planned is to have a domain-specific ontology that can be attached to the tool where it can be used to populate the domain-specific knowledge graph.
- 4. Response for comment 4:** This is true. In the field of low-temperature plasma science and technology, there is rarely any case that requires the researcher to fill out hundreds or even thousands of entries in an experiment. However, to make sure that Adamant can be adopted in different research fields, this requirement is, of course, crucial. One thing that we have in mind is to have a feature of uploading a tabular file, which is then translated to its JSON representation. Such a feature would also enable the collection and further processing of standard-conform metadata in "offline" field experiments.
- 5. Response for comment 5:** Currently, we are looking into implementing a feature of dataset publication in the DKAN-based data platform INPTDAT (<https://www.inptdat.de/>). The idea is to re-use the collected experiment metadata

(created with Adamant and stored in the eLabFTW ELN) and bundle it with the dataset for publication. Note that the back-end can be easily adapted to use the existing APIs of the mentioned repository platforms to publish metadata, once it is available in a standardized format.

Competing Interests: No competing interests were disclosed.

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com

F1000Research