

eXPars-PLD User Manual

eLabFTW to NeXus Parser for PLD Fabrications

Emanuele D'Amico

Version v0.2.1, 2026-05-14: alpha untested

Table of Contents

Introduction.....	1
NFFA-DI and FAIR Principles.....	1
eLabFTW	2
The output: HDF5 and NeXus files	2
Using the software	4
Downloading the source code.....	4
Preparing the environment.....	4
Configuration through .env file	5
Running the program	7

Introduction

eXPars - short for *eLabFTW to NeXus Parser* - is (hopefully) a family of specialized parsing software applications, mainly developed in Python, whose primary job is to automatically transform experimental metadata and data - originally stored as JSON objects inside an electronic lab notebook - into standardized, self-descriptive **NeXus files**.

The software is designed to fetch "scattered" data (often distributed across multiple linked entries) from our eLN^[1] of choice - **eLabFTW** - where the data is originally stored as JSON objects. It then parses the included metadata to resolve the full dataset which is then used to create a dictionary following a pre-established schema (dependent on the analysis or fabrication method, e.g., PLD, XRD, or RHEED), and finally uses said dictionary to produce an **HDF5/NeXus file** which complies with the **FAIR Principles** and the guidelines given within the context of the Italian PNRR^[2] **NFFA-DI**.

Specifically, **eXPars-PLD** is designed for **Pulsed Laser Deposition / PLD** fabrications.

NFFA-DI and FAIR Principles

PNRR (*Piano Nazionale di Ripresa e Resilienza*) is Italy's national recovery plan from the aftermaths of COVID-19.

NFFA-DI (*Nano Foundries and Fine Analysis - Digital Infrastructure*) is a project within this plan aimed at creating a distributed digital infrastructure for nanoscience and nanotechnology. In practice, NFFA-DI provides a unified cyber-platform for researchers to access advanced instrumentation, simulation tools, and data management services across multiple Italian research centers.

Like most modern scientific projects NFFA-DI is *FAIR by design*, meaning it strives for total compliance to **FAIR Principles**. FAIR is the acronym of the four main characteristics all compliant projects should share:

- Findable: «Metadata and data should be easy to find for both humans and computers.»
- Accessible: «Once the user finds the required data, she/he/they need to know how they can be accessed, possibly including authentication and authorisation.»
- Interoperable: «The data usually need to be integrated with other data. In addition, the data need to interoperate with applications or workflows for analysis, storage, and processing.»
- Reusable: «Metadata and data should be well-described so that they can be replicated and/or combined in different settings.»

Source: [GO FAIR](#)

eXPars-PLD contributes to NFFA-DI goals by enabling automated data harmonization: converting local PLD experiment records into a common, shareable format (NeXus) with a mutually agreed upon schema, thereby making the data interoperable across the entire NFFA-DI ecosystem.



More info on NFFA-DI at nffa-di.it.

eLabFTW

eLabFTW is an open-source, web-based electronic laboratory notebook and resource manager. It acts as a central digital hub for one or more laboratories, organizing information (as database entries) into two main constructs:

- **Experiments:** They are the core feature of eLabFTW, can contain structured data (via custom JSON fields), unstructured text, timestamps, tags, links to files (attachments), and relations to database items.
- **Resources or Items:** This is a separate, structured inventory for items like raw materials (targets, substrates), instruments (UHV machines) or samples. Each entry is built from customizable templates with defined metadata (e.g. for a substrate batch we have name, manufacturer, geometry, available pieces left...).

Although separated into different database constructs, experiments and items all have their own unique, incremental internal ID, which we'll simply call **elabid** to distinguish it from other identifiers, with no academic utility but extremely important when dealing with eLabFTW from a developer's perspective.

In a software like eLabFTW where data can (and will) be spread out through multiple entries, a particularly useful feature is **linking**: the software allows you to link experiments or items with each other, using elabid's as identifiers. For a PLD deposition, you can link the experiment describing a single layer to the target used, the substrate, the PLD instrument and the sample produced itself (all of which are eLabFTW items). This creates a complete provenance graph which can be (not-so) easily resolved starting from the sample's metadata and a chain of HTTP requests.

In this optic, eXPars-PLD interacts with eLabFTW via its REST API (Application Programming Interface). It reads a starting sample's ID (the entry point), fetches the relevant JSON metadata, chains requests using the elabid's of the sample's linked resources and experiments, rebuilds the entire dataset and if available downloads attached instrument files (e.g., RHEED intensities, images) to package all of it into the final NeXus file.

The output: HDF5 and NeXus files

The output of eXPars is an **HDF5 (Hierarchical Data Format ver. 5) file**, which is a powerful file format designed to store and organize large volumes of numerical data. It acts like a virtual file system inside a single file, using a hierarchical group/dataset structures in the same way a file system uses folders and files - with both elements having their own metadata; this way the file is self-describing, containing all relevant information like a small database. HDF5 also supports efficient slicing, compression and parallel I/O. The file extension of such format is **.h5**.

On the other hand, **NeXus** is a common data standard built on top of HDF5. It defines fixed

conventions for naming groups, datasets and attributes, specifically for neutron, X-ray, and now materials science experiments. NeXus provides "application definitions" (like *NXpld_fabrication* for PLD) that specify exactly which fields must/may appear. NeXus is also heavily promoted by *FAIRmat*, a German-based consortium, part of the NFDI, whose main mission is providing scientists «with a FAIR data infrastructure and the skills and tool they need to make the most of it»^[3]. The file extension of such format is **.nxs**, but generally file viewers treat the two formats similarly.

Last but not least, NeXus is also the format of choice for data sharing in the NFFA-DI guidelines. Which brings us to the reason why eXPars exists.

Reading HDF5/NeXus files

While writing an HDF5/NeXus file usually requires dedicated software and/or a good knowledge of programming and familiarity with specific libraries (like h5py), there are multiple ways to read these files even without such knowledge.

One of such ways would be using the online NeXus file viewer of the NCNR (*NIST Center for Neutron Research*), available on their [website](#). The "Browse..." button at the bottom allows for uploading both h5 and nxs files, although drag and drop also works.

Another similar but in my opinion more elegant online file viewer is the one hosted by the HDF5 Group: [MyHDF5](#). Other than the more modern appearance this viewer doesn't upload files to any remote server, with every operation happening locally in your browser; the drag and drop works better meaning you won't accidentally reload the page if you miss the dropping area, and the viewer also allows for opening multiple concurrent files, and downloading h5 files from URL.

[1] Acronym for "electronic Lab Notebook".

[2] PNRR stands for *National Recovery and Resilience Plan*.

[3] As stated on their [website](#).

Using the software



This software requires Python 3.12 or later.
The module **venv** and the package manager **pip** are also required.

Downloading the source code



Currently (2026-05-14) the source code is hosted on a private Gitea instance, owned by Emanuele D'Amico.
If the site is down for maintenance or temporarily unavailable please contact the webmaster via [e-mail](#).

The source code can be acquired directly via **git**, or downloaded from the official repository on [Gitea D'Amico](#).

```
git clone https://gitea.damico.ing/emanuele/eXParser-PLD.git eXPars-PLD
cd eXPars-PLD # enter directory
ls
LICENSE      docs/        output/      src/
README.md    glossary    requirements.txt tests/
```

Optionally, you can access the code in the development branch by executing:

```
git checkout dev
```

Preparing the environment

Before starting eXPars-PLD v0.2.1 requires a total of 6 modules to be installed, which are listed [here](#). Since installing a Python module system-wide is almost never a good idea, start by creating and activating a virtual environment.

In the software folder, run:

```
# Calls venv module to create new Python virtual environment in .venv:
python3 -m venv .venv
# If command is successful, running ls should show a new .venv folder:
ls -ld .*
.venv
# Activate venv:
source .venv/bin/activate
```

```
emanuele@Lagann:~/git/parser-PLD$ source .venv/bin/activate
(.venv) emanuele@Lagann:~/git/parser-PLD$
```

Figure 1. Most shells like Bash show very clearly when you're working inside a virtual environment.

At this point you're free to install the requirements through **pip**:

```
# Install from list in requirements.txt:
pip install -r requirements.txt
```

Most of the warnings displayed by pip are safe and generally it's not dangerous to ignore them. Unless pip exits abruptly returning an error, your environment is ready to work.

Configuration through .env file

Much like the previous step, configuring the software with your settings (API key, eLabFTW URL...) is something you do *una tantum* and then usually forget about it.

Inside the eXPars-PLD folder there's a file called **.env.example**. Rename it removing ".example", then open it with your editor of choice. This is your **.env** (or **dotenv**) file.

```
mv .env.example .env
vim .env
# The file presents itself like this:
1 | api_key=""
2 | elabid=""
3 | ELABFTW_API_URL="https://elabftw.fisica.unina.it/api/v2"
4 | operative_unit="cnr-spin.na"
```

- **api_key** is your own personal eLabFTW API key. Generating one is an easy task explained in full detail below.
- **elabid** is the elabid of the resource you'd like to select (your starting sample); this field can (and probably should) be left blank - in which case the application prompts you for an elabid on runtime, and your answer will not be stored meaning you can easily rerun the program with a different target.
- **ELABFTW_API_URL** is the URL of your eLabFTW instance; if you're running this from the laboratories in Monte S. Angelo, Naples, you're probably leaving this field as it is.
- **operative_unit** is the operative unit you ran your experiments from. It's only needed to compose the filename of the NeXus, which can be easily modified anytime later, and it's not necessary for creating the file itself.

None of these fields are required, meaning you can technically skip this entire section. If any of the first three keys are blank or missing you will be prompted to provide the necessary info at runtime, and your answers will not be memorized - meaning e.g. you will have to provide your API key

every time you run the program.



Do NOT confuse .env with .venv: the first is a file containing all the environmental variables you need to run eXPars-PLD properly, the latter is a directory containing your virtual environment with all the required modules.

Generating an eLabFTW API key

eLabFTW has its own [API documentation](#) on which you can rely. A new API key can be generated in the Settings → API Keys page by giving it a name and an access level:

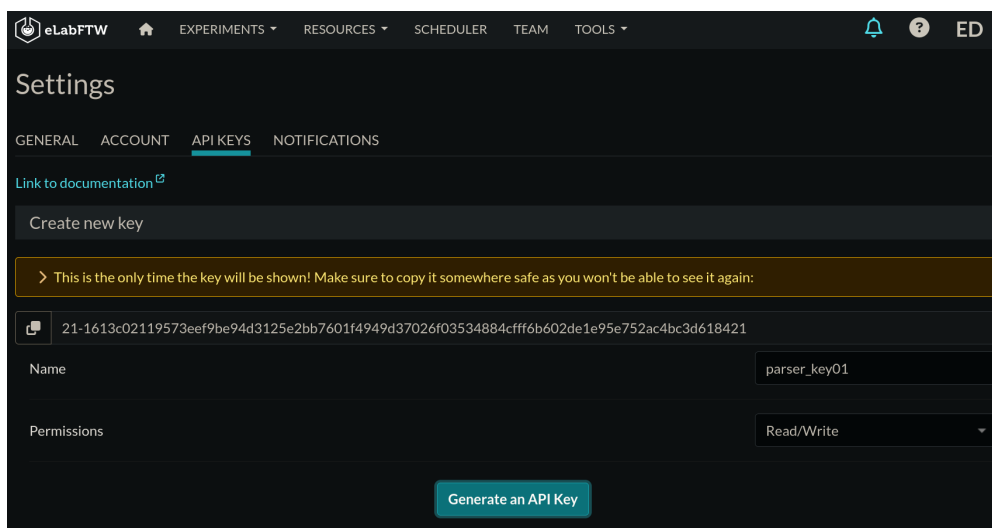


Figure 2. Screenshot from our eLabFTW. The key must have a name and permissions. Naturally, the key you see here in clear has been invalidated.

The **name** of the key is a descriptor for you to remember why you created it in the first place - something like "parser_key01". The **permissions** can either be "Read/Write" or "Read-Only": in the first scenario the key may also be used to edit or create entries you own on eLabFTW, while read-only key only allow GET requests. eXPars-PLD doesn't require writing permissions, so both options will do.



A few warnings.

- The key eLabFTW generates is only shown once, then stored encrypted in the database. This means that after closing or refreshing the page the key is lost forever if not saved on an external support. Which brings us to the second warning.
- Store and protect your API key like you would your password, as it gives full/limited access to your account exactly like your password, but without the protection given by 2FA/MFA. For this purpose there are many offline (like [KeePass](#)) or online (like [BitWarden](#)) **password managers**.
- Your .env file is NOT a safe place to *store* your API key. Once pasted there be very careful who you share your files with, and be careful not to expose your key when sending your NeXus files to other computers. If you don't trust your awareness leave the api_key field blank and just paste your API key in the terminal every time you run eXPars-PLD.

Running the program

Open a terminal into the project folder. Before attempting to run the program:

- Make sure your virtual environment is active, or if it isn't run:
`source .venv/bin/activate`
- Make sure the required modules are installed, or if they aren't run: `pip install -r requirements.txt`
- Make sure your .env file is properly set, or if it isn't make sure you know how to paste into the terminal the API key, the elabid of the required source and the URL of your eLabFTW instance (ending in `/api/v2`).

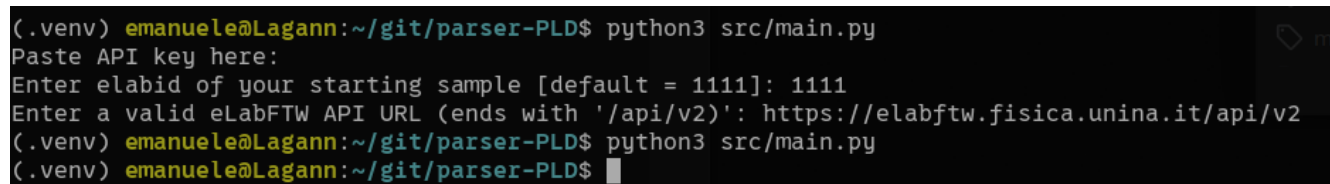
When you're ready, run:

```
python3 src/main.py
```

If your .env file is completely filled out with valid values the only output you may read on the terminal are warnings or worst-case-scenario errors. Next chapter will cover all such cases. If your .env file lacks one or more values you will be asked to input the missing info at runtime.

Entering missing values if prompted

If you decide to run without a valid .env file (again, worst-case-scenario) you will be prompted to enter the required information directly into the terminal.



```
(.venv) emanuele@Lagann:~/git/parser-PLD$ python3 src/main.py
Paste API key here:
Enter elabid of your starting sample [default = 1111]: 1111
Enter a valid eLabFTW API URL (ends with '/api/v2)': https://elabftw.fisica.unina.it/api/v2
(.venv) emanuele@Lagann:~/git/parser-PLD$ python3 src/main.py
(.venv) emanuele@Lagann:~/git/parser-PLD$
```

Figure 3. The difference between running eXPars-PLD with no .env, and with a properly filled out .env. Same parameters, same output.

First and foremost you will be prompted for a valid API key. To paste your key in the terminal either right-click (PowerShell and other terminal emulators), right-click > Paste, Ctrl + Shift + V (on most terminal emulators) or middle-click (Linux).

Then you will be prompted for an elabid - which is a positive integer number. You can find your sample's elabid on eLabFTW, above the sample's name and before the sample's label and status. See [Figure 4](#).

Last but not least you will be prompted for a valid eLabFTW API endpoint URL. Such URL is composed by the base URL of your eLabFTW instance, closing with `/api/v2`. For instance: `https://elabftw.fisica.unina.it/api/v2`.

eXPars-PLD v0.2.1 will not validate such URL or return some very specific error.



Make sure the URL you paste doesn't end with a trailing slash.

<https://elabftw.fisica.unina.it/api/v2/> □
<https://elabftw.fisica.unina.it/api/v2/> □

You won't be prompted for the operative unit, so that will require either setting up a `.env` or manually editing your NeXus files' names. The list of officially approved acronyms for the operative units can be consulted on NFFA-DI's [official website](#).

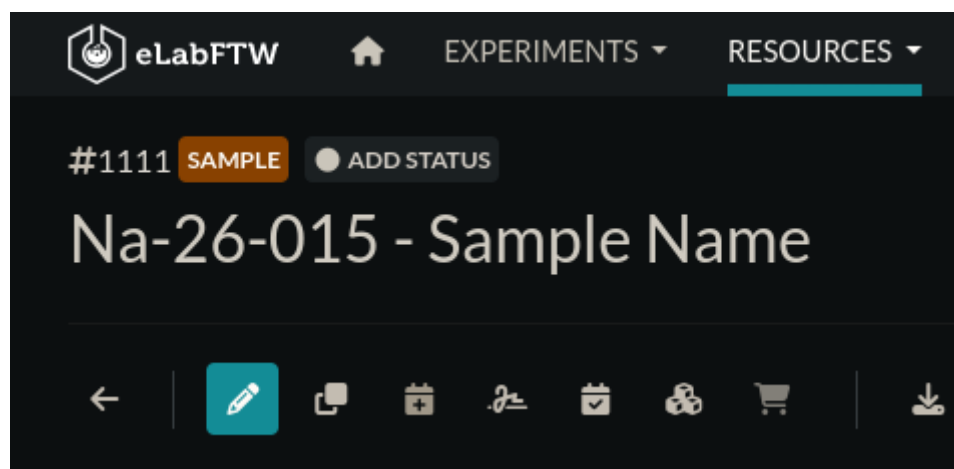


Figure 4. Where to find the elabid of a sample.

Retrieving and verifying your file

By default the NeXus file will be saved in the `output/` folder. Currently (2026-05-14) the software will also save a JSON dictionary with the full chain of all metadata collected on the sample. There is also an `attachments/` folder containing all the attachments downloaded during execution, which will be removed later on.

The file will be recognizable by its name, which should already be in compliance with the following NFFA-DI naming guidelines:

«Each file generated in the context of a Proposal stored on OFED must use the following naming convention: `nffa-di_[proposal_id]_[UO]_[UO_internal_id]`» - where *proposal_id* is the approved ID of the research proposal, *UO* is the [official code](#) of the operative unit, and «*UO_internal_id* is a combination of the technique/instrument acronym and an Experiment ID freely decided».

«Each file generated in the context of an In-house Research Project stored on OFED must use the following naming convention: `nffa-di_[UO]_[project_id]_key`, where the first part of the name adheres to the name of the bucket, while key is arbitrary.»

Source: [NFFA-DI Research Data Policy](#)

This means that the accepted filename for a NeXus file of a PLD, where `proposal_id` is `EXMPL01`, the operative unit is CNR-SPIN Naples and the sample's internal ID is `Na-26-012` the filename will be:

nffa-di **EXMPL01** **cnr-spin.na** **PLD** **Na-26-012**.nxs

<i>proposal_id</i>	<i>UO</i>	<i>technique</i>	<i>free_internal_id</i>
		<i>UO_internal_id</i>	

A NeXus file can be verified through one of the readers listed in [Reading HDF5/NeXus files](#). Pay attention to the following aspects:

- Do I visualize the file correctly?
- Does the file respect the fabrication method's schema?
- Is every required field present? Do I read the same values on eLabFTW and in the NeXus file?
Are the units of measurement present?
- Can I visualize heatmaps and N-axis graphs correctly?

If the answer to all previous questions is "Yes", then the output file is NFFA-DI compliant.