Developing a NOMAD Plugin

In this tutorial you will learn how to create and develop a NOMAD plugin. As an example we

will create a plugin to log data for a simple sintering process.

Prerequisites

- A GitHub account. This can be created for free on github.com (https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home).

- Basic understanding of Python.

- Basic understanding of NOMAD metainfo, see for example tutorial 8 (https://www.fairmat-nfdi.eu/events/fairmat-tutorial-8/tutorial-8-materials).

!!! note

Several software development concepts are being used during this tutorial.

Here is a list with some further information on each of them:

* what is Git (https://learn.microsoft.com/en-us/devops/develop/git/what-is-git)

* what is VSCode, i. e., an Integrated Development Environment (IDE) (https://aws.amazon.com/what-is/ide/)

* what is Pip (https://realpython.com/lessons/what-is-pip-overview/)

* what is a Python virtual environment (https://realpython.com/python-virtual-environments-a-primer/#why-do-you-need-virtual-environments)

* creating a Python package (https://packaging.python.org/en/latest/tutorials/packaging-projects/)

* uploading a package to PyPI (https://www.freecodecamp.org/news/how-to-create-and-upload-your-first-python-package-to-pypi/)

* what is cruft (https://cruft.github.io/cruft/)

Create a Git(Hub) repository

Firstly, we recommend to use git to version control your NOMAD plugin.

There is a GitHub template repository that can be used for this at github.com/FAIRmat-NFDI/nomad-plugin-template (https://github.com/FAIRmat-NFDI/nomad-plugin-template).

To use the template you should choose the "Create an new repository" option after pressing

the green "Use this template" button in the upper right corner.

Please note that you have to be logged into to GitHub to see this option.

!Use template (./images/use_template_dark.png#gh-dark-mode-only)

!Use template (./images/use_template_light.png#gh-light-mode-only)

Enter a name (I will use "nomad-sintering" for mine) for your repository and click "Create Repository".

Generate the plugin structure

Next, we will use a cookiecutter template to create the basic structure of our NOMAD plugin.

There are now two options for how to proceed.

1. You can use the GitHub codespaces environment to develop your plugin, or

2. If you have access to a Linux computer you can also run the same steps locally.

1. Using GitHub codespaces

To use a GitHub codespace for the plugin development you should choose the "Create codespace on main" option after pressing the green " Code" button in the upper right corner.

!Use codepace (./images/codespace_dark.png#gh-dark-mode-only)

!Use codespace (./images/codespace_light.png#gh-light-mode-only)

2. Developing locally

If you have a Linux machine and prefer to develop locally you should **instead** click the

"Local" tab after pressing the green " Code" button, copy the path, and clone your repository by running:

```sh
git clone PATH/COPIED/FROM/REPOSITORY
```

and move inside the top directory

```
cd REPOSITORY_NAME
```

You will also need to install cruft (https://pypi.org/project/cruft/), preferably using `pipx`:

```sh
pipx is strongly recommended.
pipx install cruft
If pipx is not an option,
you can install cruft in your Python user directory.
python -m pip install --user cruft
```

Run cruft

The next step is to run cruft to use our cookiecutter template:

```sh
cruft create https://github.com/FAIRmat-NFDI/cookiecutter-nomad-plugin
```

Cookiecutter prompts you for information regarding your plugin and I will enter the following for my example:

```no-highlight
  [1/12] full_name (John Doe): Hampus Näsström
  [2/12] email (john.doe@physik.hu-berlin.de): hampus.naesstroem@physik.hu-berlin.de
  [3/12] github_username (foo): hampusnasstrom
  [4/12] plugin_name (foobar): sintering
  [5/12] module_name (sintering):
   [6/12] short_description (Nomad example template): A schema package plugin for sintering.
  [7/12] version (0.1.0):
  [8/12] Select license
    1 - MIT
    2 - BSD-3
    3 - GNU GPL v3.0+
    4 - Apache Software License 2.0
    Choose from [1/2/3/4] (1):
  [9/12] include_schema_package [y/n] (y): y
  [10/12] include_normalizer [y/n] (y): n
  [11/12] include_parser [y/n] (y): n
  [12/12] include_app [y/n] (y): n
```

There you go - you just created a minimal NOMAD plugin:

!!! note

    In the above prompt, we pressed `y` for schema_package, this creates a python package

with a plugin entry point for a schema package.

```no-highlight
nomad-sintering/
├── LICENSE
├── MANIFEST.in
├── README.md
├── docs
│   └── ...
├── mkdocs.yml
├── move_template_files.sh
├── pyproject.toml
├── src
│   └── nomad_sintering
│       ├── __init__.py
│       └── schema_packages
│           ├── __init__.py
│           └── mypackage.py
└── tests
    ├── conftest.py
    ├── data
    │   └── test.archive.yaml
    └── schema_packages
        └── test_schema.py
```

!!! note

    The project `nomad-sintering` is created in a new directory, we have included a

helper script to move all the files to the parent level of the repository.

```sh
sh CHANGE_TO_PLUGIN_NAME/move_template_files.sh
```

!!! warning "Attention"

The `CHANGE_TO_PLUGIN_NAME` should be substituted by the name of the plugin you've created. In the above case it'll be `sh nomad-sintering/move_template_files.sh`.

Finally, we should add the files we created to git and commit the changes we have made:

```sh
git add -A
git commit -m "Generated plugin from cookiecutter template"
git push
```

Enable Cruft updates

In order to receive updates from our cookiecutter template we have included a GitHub action that automatically checks for updates once a week (or by triggering it manually). In order for this action to run we need to give the action permission to write and create pull requests. To do this we should go back to the plugin repo and head to the settings tab and navigate to the Actions/General options on the left:

!Use template (./images/github_settings_dark.png#gh-dark-mode-only)

!Use template (./images/github_settings_light.png#gh-light-mode-only)

At the very bottom of this place you should mark the "Read and write permissions" and the "Allow GitHub Actions to create and approve pull requests" options and click save.

!Use template (./images/workflow_permissions_dark.png#gh-dark-mode-only)

!Use template (./images/workflow_permissions_light.png#gh-light-mode-only)

Setting up the python environment

Creating a virtual environment

Before we can start developing we recommend to create a virtual environment using Python 3.9

```sh
python3.9 -m venv .pyenv
source .pyenv/bin/activate
```

Installing the plugin

Next we should install our plugin package in editable mode and using the nomad package index

```sh
pip install --upgrade pip
pip install -e '.[dev]' --index-url https://gitlab.mpcdf.mpg.de/api/v4/projects/2187/packages/pypi/simple
```

!!! note

Until we have an official PyPI NOMAD release with the latest NOMAD version, make sure to include NOMAD's internal package registry (e.g. via --index-url). The latest PyPI package available today is version 1.2.2 and it misses some updates functional to this tutorial.

In the future, when a newer release of `nomad-lab` will be available (1.2.2) you can omit the `--index-url`.

Importing a yaml schema

The schema

We will now convert the yaml schema package from part 2 where we described a sintering

step:

```yaml
definitions:
  name: 'Tutorial 13 sintering schema'
  sections:
    TemperatureRamp:
      m_annotations:
        eln:
          properties:
            order:
              - "name"
              - "start_time"
              - "initial_temperature"
              - "final_temperature"
              - "duration"
              - "comment"
      base_sections:
        - nomad.datamodel.metainfo.basesections.ProcessStep
      quantities:
        initial_temperature:
          type: np.float64
          unit: celsius
          description: "initial temperature set for ramp"
```

```
        m_annotations:
          eln:
            component: NumberEditQuantity
            defaultDisplayUnit: celsius
      final_temperature:
        type: np.float64
        unit: celsius
        description: "final temperature set for ramp"
        m_annotations:
          eln:
            component: NumberEditQuantity
            defaultDisplayUnit: celsius
  Sintering:
    base_sections:
      - nomad.datamodel.metainfo.basesections.Process
      - nomad.datamodel.data.EntryData
    sub_sections:
      steps:
        repeats: True
        section: '#/TemperatureRamp'
```

We can grab this file from the tutorial repository using curl

```sh
curl -L -o sintering.archive.yaml "https://raw.githubusercontent.com/FAIRmat-NFDI/AreaA-Examples/main/tutorial13/part3/files/sintering.archive.yaml"
```

```

`metainfo-yaml2py`

We will now use an external package `metainfo-yaml2py` to convert the yaml schema package

into python class definitions.

First we install the package with `pip`:

```sh

pip install metainfoyaml2py

```

Then we can run the `metainfo-yaml2py` command on the `sintering.archive.yaml` file with

the `-n` flag for adding `normalize()` functions (will be explained later)

and specify the output directory, with the `-o` flag, to be our `schema_packages` directory:

```sh

metainfo-yaml2py sintering.archive.yaml -o src/nomad_sintering/schema_packages -n

```

Updating `__init__.py` and `pyproject.toml`

The metadata of our package is defined in the `__init__.py` file and here we now need to

add the sintering package that we just created.

If we take a look in that file we can see an example created by the cookiecutter template.

We can go ahead and copy the `MySchemaPackageEntryPoint` class and the `mypackage`

instance and paste them below.
```

We then need to change:

1. the name of the class,

2. the import in the load function to import our sintering schema package,

3. the name of the instance and the class it uses,

4. ideally we should also update the description and the name.

The changes could look something like this:

```py
class SinteringEntryPoint(SchemaPackageEntryPoint):
    def load(self):
        from nomad_sintering.schema_packages.sintering import m_package
        return m_package
sintering = SinteringEntryPoint(
    name='Sintering',
    description='Schema package for describing a sintering process.',
)
```

Finally, we also need to add our new entry point to the `pyproject.toml`.

At the bottom of the toml you will see how this was done for the example and we just need

to replicate that with whatever we called our instance:

```toml
sintering = "nomad_sintering.schema_packages:sintering"
```

Before we continue, we should commit our changes to git:

```sh
git add -A
```

```
git commit -m "Added sintering classes from yaml schema"
git push
```

Ruff autoformatting

If we check the actions tab of the GitHub repository we might see that the last commit caused an error in the Ruff format checking. We can either disable this workflow (not recommended) or we can check and format our code with Ruff.

To check what Ruff thinks about our code we run:

```sh
ruff check .
```

To fix any issues we can run:

```sh
ruff check . --fix
```

And commit the changes:

```sh
git add -A
git commit -m "Ruff linting"
git push
```

Adding a normalize function

Next we will add some functionality to our use case through a so called "normalize" function. This allows us to add functionality to our schemas via Python code.

The use case

For this tutorial we will assume that we have a recipe file for our hot plate that we will

parse:

```csv
step name,duration [min],initial temperature [C],final temperature [C]
heating, 30, 25, 300
hold, 60, 300, 300
cooling, 30, 300, 25
```

We can grab this file from the tutorial repository and place it in the tests/data directory using curl

```sh
curl -L -o tests/data/sintering_example.csv "https://raw.githubusercontent.com/FAIRmat-NFDI/AreaA-Examples/main/tutorial13/part3/files/sintering_example.csv"
```

Adding the code

The first thing we need to add is a new `Quantity` in our `Sintering` class to hold the recipe file:

```py
data_file = Quantity(
    type=str,
    description='The recipe file for the sintering process.',
    a_eln={
        "component": "FileEditQuantity",
    },
)
```

Here we have used the `a_eln` component annotation to add a `FileEditQuantity`. You will
see in part 4 how this looks in the GUI.

Secondly we need to update the normalize method to read the data file and update the corresponding data.

First we will check if the `self.data_file` is present and, if so, use the `archive.m_context.raw_file()` method to open the file and read it with the pandas function `read_csv()`:

```py
if self.data_file:
  with archive.m_context.raw_file(self.data_file) as file:
    df = pd.read_csv(file)
```

We will then create a list to hold the steps, iterate over our data frame, create an instance of a `TemperatureRamp`, and fill them.

```py
    steps = []
    for i, row in df.iterrows():
      step = TemperatureRamp()
      step.name = row['step name']
      step.duration = ureg.Quantity(float(row['duration [min]']), 'min')
      step.initial_temperature = ureg.Quantity(row['initial temperature [C]'], 'celsius')
      step.final_temperature = ureg.Quantity(row['final temperature [C]'], 'celsius')
      steps.append(step)
```

Here we have used the NOMAD unit registry to handle all the units.

Finally, we will assign the `self.steps` with our new list of steps.

```py
  self.steps = steps
```

We also need to add the import of pandas and the NOMAD unit registry to the top of our `sintering.py` file:

```py
from nomad.units import ureg
import pandas as pd
```

Here are all the changes combined:

```py
from nomad.units import ureg
import pandas as pd
class Sintering(Process, EntryData, ArchiveSection):
    '''
    Class autogenerated from yaml schema.
    '''
    m_def = Section()
    steps = SubSection(
        section_def=TemperatureRamp,
        repeats=True,
    )
    data_file = Quantity(
        type=str,
        description='The recipe file for the sintering process.',
```

```
    a_eln={
        "component": "FileEditQuantity",
    },
  )
  def normalize(self, archive, logger: BoundLogger) -    None:
    '''
    The normalizer for the `Sintering` class.
    Args:
        archive (EntryArchive): The archive containing the section that is being
        normalized.
        logger (BoundLogger): A structlog logger.
    '''
    super(Sintering, self).normalize(archive, logger)
    if self.data_file:
      with archive.m_context.raw_file(self.data_file) as file:
        df = pd.read_csv(file)
      steps = []
      for i, row in df.iterrows():
        step = TemperatureRamp()
        step.name = row['step name']
        step.duration = ureg.Quantity(float(row['duration [min]']), 'min')
        step.initial_temperature = ureg.Quantity(row['initial temperature [C]'], 'celsius')
        step.final_temperature = ureg.Quantity(row['final temperature [C]'], 'celsius')
        steps.append(step)
    self.steps = steps
```

Running the normalize function

We will now run the NOMAD processing on a test file to see the normalize function in action.

Create an archive.json file

The first step is to create the test file.

We should add a file with the ending `.archive.yaml` or `archive.json` and which contains

a `data` section and an `m_def` key with the value being our sintering section.

Finally, we should also add the `data_file` key with the value being our `.csv` file from before.

```yaml
data:
  m_def: nomad_sintering.schema_packages.sintering.Sintering
  data_file: sintering_example.csv
```

We can once again grab this file from the tutorial repository and place it in the tests/data directory using curl

```sh
curl -L -o tests/data/test_sintering.archive.yaml "https://raw.githubusercontent.com/FAIRmat-NFDI/AreaA-Examples/main/tutorial13/part3/files/test_sintering.archive.yaml"
```

!!! warning "Attention"

    You might need to modify the package name for the `m_def` if you called your python

    module something other than `nomad_sintering`

Run the NOMAD CLI

To run the processing we us the NOMAD CLI method `parse` with the flag `--show-archive`

and save the output in a json file

```sh
nomad parse tests/data/test_sintering.archive.yaml --show-archive > normalized.archive.json
```

However, when we run this we will get an error from NOMAD!

```bash
could not normalize section (normalizer=MetainfoNormalizer, section=Sintering, exc_info=Cannot convert from 'milliinch' ([length]) to 'second' ([time]))
```

What is happening here is that it has treated our `'min'` unit for duration as `'milliinch'` and not the intended minutes. To fix this we can directly edit the normalize function of the `Sintering` class in the `sintering.py` file by replacing `'min'` with `'minutes'`.

```py
def normalize(self, archive: 'EntryArchive', logger: 'BoundLogger') -> None:
    """
    The normalizer for the `Sintering` class.
    Args:
        archive (EntryArchive): The archive containing the section that is being
        normalized.
        logger (BoundLogger): A structlog logger.
    """
    super().normalize(archive, logger)
```

```
    if self.data_file:
        with archive.m_context.raw_file(self.data_file) as file:
            df = pd.read_csv(file)
        steps = []
        for i, row in df.iterrows():
            step = TemperatureRamp()
            step.name = row['step name']
            # Changed 'min' to 'minutes' here:
            step.duration = ureg.Quantity(float(row['duration [min]']), 'minutes')
```

Since we installed our package in editable mode the changes will take effect as soon as we
save and rerunning the nomad parse command above should now work.

To view the output you can open and inspect the `normalized.archive.json` file. The beginning of that file should look something like:

```json
{
  "data": {
    "m_def": "nomad_sintering.schema_packages.sintering.Sintering",
    "name": "test sintering",
    "datetime": "2024-06-04T16:52:23.998519+00:00",
    "data_file": "sintering_example.csv",
    "steps": [
      {
        "name": "heating",
        "duration": 1800.0,
```

```
      "initial_temperature": 25.0,
      "final_temperature": 300.0
    },
    {
      "name": "hold",
      "duration": 3600.0,
      "initial_temperature": 300.0,
      "final_temperature": 300.0
    },
    {
      "name": "cooling",
      "duration": 1800.0,
      "initial_temperature": 300.0,
      "final_temperature": 25.0
    }
  ]
 },
...
```

Next steps

The next step is to include your new schema in a custom NOMAD Oasis. For more information

on how to setup a NOMAD Oasis you can have a look at

How-to guides/NOMAD Oasis/Install and Oasis (../howto/oasis/install.md).

Before we move one we should make sure that we have committed our changes to git:

```sh
```

```
git add -A

git commit -m "Added a normalize function to the Sintering schema"

git push
```