How to write an app

Apps provide customized views of data in the GUI, making it easier for the users to navigate and understand the data related to a specific domain. This typically means that certain domain-specific properties are highlighted, different units may be used for physical properties, and specialized dashboards may be presented. This becomes crucial for NOMAD installations to be able to scale with data that contains a mixture of experiments and simulations, different techniques, and physical properties spanning different time and length scales.

Apps only affect the way data is *displayed* for the user: if you wish to affect the underlying data structure, you will need to write a Python schema package (./schema_packages.md) or a YAML schema package (../customization/basics.md).

This documentation shows you how to write an plugin entry point for an app. You should read the documentation on getting started with plugins (./plugins.md) to have a basic understanding of how plugins and plugin entry points work in the NOMAD ecosystem.

Getting started

You can use our template repository (https://github.com/FAIRmat-NFDI/nomad-plugin-template) to create an initial structure for a plugin containing an app. The relevant part of the repository layout will look something like this:

```txt
nomad-example
   ├── src
   │   ├── nomad_example
   │   │   ├── apps
   │   │   │   ├── __init__.py
   ├── LICENSE.txt
   ├── README.md
   └── pyproject.toml
```

See the documentation on plugin development guidelines (./plugins.md#plugin-development-guidelines) for more details on the best development practices for plugins, including linting, testing and documenting.

App entry point

The entry point defines basic information about your app and is used to automatically load the app into a NOMAD distribution. It is an instance of an `AppEntryPoint` and unlike many other plugin entry points, it does not have a separate resource that needs to be lazy-loaded as the entire app is defined in the configuration as an instance of `nomad.config.models.ui.App`. You will learn more about the `App` class in the next sections. The entry point should be defined in `*/apps/__init__.py` like this:

```python
from nomad.config.models.plugins import AppEntryPoint
myapp = MyAppEntryPoint(
    name = 'MyApp',
    description = 'My custom app.',
    app = App(...)
)
```

Here we have instantiated an object `myapp` in which you specify the default parameterization and other details about the app. In the reference you can see all of the available configuration options for an `AppEntryPoint` (../../reference/plugins.md#appentrypoint).

The entry point instance should then be added to the `[project.entry-points.'nomad.plugin']` table in `pyproject.toml` in order for the app to be automatically detected:

```toml
[project.entry-points.'nomad.plugin']
myapp = "nomad_example.apps:myapp"
```

`App` class

The definition fo the actual app is given as an instance of the `App` class specified as part of the entry point. A full breakdown of the model is given below in the app reference (#app-reference), but here is a small example:

```python
from nomad.config.models.plugins import AppEntryPoint
from nomad.config.models.ui import App, Column, Columns, FilterMenu, FilterMenus, Filters
myapp = AppEntryPoint(
    name='MyApp',
    description='App defined using the new plugin mechanism.',
    app = App(
        # Label of the App
        label='My App',
        # Path used in the URL, must be unique
        path='myapp',
        # Used to categorize apps in the explore menu
        category='Theory',
        # Brief description used in the app menu
        description='An app customized for me.',
        # Longer description that can also use markdown
        readme='Here is a much longer description of this app.',
        # Controls the available search filters. If you want to filter by
        # quantities in a schema package, you need to load the schema package
        # explicitly here. Note that you can use a glob syntax to load the
        # entire package, or just a single schema from a package.
        filters=Filters(
            include=['*#nomad_example.schema_packages.mypackage.MySchema'],
        ),
        # Controls which columns are shown in the results table
        columns=Columns(
            selected=[
                'entry_id'

'data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchem
a'
            ],
            options={
                'entry_id': Column(),
                'upload_create_time': Column(),

'data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchem
a': Column(),
            }
        ),
        # Dictionary of search filters that are always enabled for queries made
        # within this app. This is especially important to narrow down the
        # results to the wanted subset. Any available search filter can be
        # targeted here. This example makes sure that only entries that use
        # MySchema are included.
        filters_locked={
            "section_defs.definition_qualified_name:all": [
                "nomad_example.schema_packages.mypackage.MySchema"
            ]
        },
        # Controls the filter menus shown on the left
        filter_menus=FilterMenus(
            options={
                'material': FilterMenu(label="Material"),
            }
        ),
        # Controls the default dashboard shown in the search interface
        dashboard={
            'widgets': [
                {
                    'type': 'histogram',
                    'showinput': False,
                    'autorange': True,
                    'nbins': 30,
                    'scale': 'linear',
                                                                            'quantity':
```

```
'data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchem
a',
                    'layout': {
                        'lg': {
                            'minH': 3,
                            'minW': 3,
                            'h': 4,
                            'w': 12,
                            'y': 0,
                            'x': 0
                        }
                    }
                }
            ]
        }
    )
)
```

!!! tip

If you want to load an app definition from a YAML file, this can be easily done with the pydantic `parse_obj` function:

```python
    import yaml
    from nomad.config.models.plugins import AppEntryPoint
    from nomad.config.models.ui import App
    yaml_data = """
```

```
        label: My App
        path: myapp
        category: Theory
    """
    myapp = AppEntryPoint(
        name='MyApp',
        description='App defined using the new plugin mechanism.',
        app=App.parse_obj(
            yaml.safe_load(yaml_data)
        ),
    )
```

Loading custom quantity definitions into an app

By default, none of the quantities from custom schemas are available in an app, and they need to be explicitly added. Each app may define additional **filters** that should be enabled in it. Filters have a special meaning in the app context: filters are pieces of (meta)info that can be queried in the search interface of the app, but also targeted in the rest of the app configuration as explained below in.

!!! note

    Note that not all of the quantities from a custom schema can be exposed as
    filters. At the moment we only support targeting **scalar** quantities from
    custom schemas.

Each schema has a unique name within the NOMAD ecosystem, which is needed to target them in the configuration. The name depends on the resource in which the schema is defined in:

- Python schemas are identified by the python path for the class that inherits from `Schema`. For example, if you have a python package called `nomad_example`, which has a subpackage called `schema_packages`, containing a module called `mypackage.py`, which contains the class `MySchema`, then

the schema name will be `nomad_example.schema_packages.mypackage.MySchema`.

- YAML schemas are identified by the entry id of the schema file together with the name of the section defined in the YAML schema. For example if you have uploaded a schema YAML file containing a section definition called `MySchema`, and it has been assigned an `entry_id`, the schema name will be `entry_id:.MySchema`.

The quantities from schemas may be included or excluded as filter by using the `filters` (#filters) field in the app config. This option supports a wildcard/glob syntax for including/excluding certain filters. For example, to include all filters from the Python schema defined in the class `nomad_example.schema_packages.mypackage.MySchema`, you could use:

```python
filters=Filters(
    include=['*#nomad_example.schema_packages.mypackage.MySchema']
)
```

The same thing for a YAML schema could be achieved with:

```python
filters=Filters(
    include=['*#entry_id:.MySchema']
)
```

Once quantities from a schema are included in an app as filters, they can be targeted in the rest of the app. The app configuration often refers to specific filters to configure parts of the user interface. For example, one could configure the results table to show a new column using one of the schema quantities with:

```python
columns=Columns(
    selected=[
        'entry_id'
        'data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchema'
    ],
    options={
        'entry_id': Column(),
        'upload_create_time': Column(),
        'data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchema': Column(),
    }
)
```

The syntax for targeting quantities depends on the resource:

- For python schemas, you need to provide the path and the python schema name separated by a hashtag (#), for example `data.mysection.myquantity#nomad_example.schema_packages.mypackage.MySchema`.

- For YAML schemas, you need to provide the path and the YAML schema name separated

by a hashtag (#), for example `data.mysection.myquantity#entry_id:.MySchema`.

- Quantities that are common for all NOMAD entries can be targeted by using only

the path without the need for specifying a schema, e.g. `results.material.symmetry.space_group`.

App reference

{{ pydantic_model('nomad.config.models.ui.App')}}