# PyNWB

***Release 0.2.0.post.dev123***

**Jan 18, 2018**

# Getting Started

PyNWB is a Python package for working with NWB files. It provides a high-level API for efficiently working with Neurodata stored in the NWB format.

Neurodata Without Borders: Neurophysiology (NWB:N) is a project to develop a unified data format for cellular-based neurophysiology data, focused on the dynamics of groups of neurons measured under a large range of experimental conditions.

The NWB:N team consists of neuroscientists and software developers who recognize that adoption of a unified data format is an important step toward breaking down the barriers to data sharing in neuroscience.

CHAPTER 1

# Dependencies

PyNWB has the following minimum requirements, which must be installed before you can get started using PyNWB.

1. Python 2.7, Python 3.5 or Python 3.6
2. pip

CHAPTER 2

# Installation

## 2.1 Install release from PyPI

The Python Package Index (PyPI) is a repository of software for the Python programming language.

To install or update PyNWB distribution from PyPI simply run:

```
$ pip install -U pynwb
```

This will automatically install the following required dependencies:

1. h5py
2. numpy
3. python-dateutil
4. requests
5. ruamel.yaml
6. six

## 2.2 Install release from Conda-forge

Conda-forge is a community led collection of recipes, build infrastructure and distributions for the conda package manager.

To install or update PyNWB distribution from conda-forge using conda simply run:

```
$ conda install -c conda-forge pynwb
```

## 2.3 Install latest pre-release

This is useful to tryout the latest features and also setup continuous integration of your own project against the latest version of PyNWB.

```
$ pip install -U pynwb --find-links https://github.com/NeurodataWithoutBorders/pynwb/
↪releases/tag/latest  --no-index
```

CHAPTER 3

# For developers

## 3.1 Install from Git repository

For development an editable install is recommended.

```
$ pip install -U virtualenv
$ virtualenv ~/pynwb
$ ~/pynwb/bin/activate
$ git clone git@github.com:NeurodataWithoutBorders/pynwb.git
$ cd pynwb
$ pip install -r requirements.txt -r requirements-dev.txt
$ pip install -e .
```

## 3.2 Run tests

For running the tests, it is required to install the development requirements.

```
$ pip install -U virtualenv
$ virtualenv ~/pynwb
$ ~/pynwb/bin/activate
$ git clone git@github.com:NeurodataWithoutBorders/pynwb.git
$ cd pynwb
$ pip install -r requirements.txt -r requirements-dev.txt
$ pip install -e .
$ tox
```

## 3.3 Following PyNWB Style Guide

Before you create a Pull Request, make sure you are following PyNWB style guide (PEP8). To do that simply run the following command in the project's root directory.

```
$ flake8
```

CHAPTER 4

# How to contribute to NWB:N software and documents

## 4.1 Code of Conduct

This project and everyone participating in it is governed by our code of conduct guidelines. By participating, you are expected to uphold this code. Please report unacceptable behavior.

## 4.2 Types of Contributions

### 4.2.1 Did you find a bug? or Do you intend to add a new feature or change an existing one?

- **Identify the appropriate repository** for the change you are suggesting:
  - Use nwb_schema for any changes to the NWB:N format schema, schema language, storage and other NWB:N related documents
  - Use PyNWB for any changes regarding the PyNWB API and the corresponding documentation
- **Ensure the feature or change was not already reported** by searching on GitHub under PyNWB Issues and nwh-schema issues , respectively .
- If you are unable to find an open issue addressing the problem then open a new issue on the respective repository. Be sure to include:
  - **brief and descriptive title**
  - **clear description of the problem you are trying to solve***. Describing the use case is often more important than proposing a specific solution. By describing the use case and problem you are trying to solve gives the development team and ultimately the NWB:N community a better understanding for the reasons of changes and enables others to suggest solutions.
  - **context** providing as much relevant information as possible and if available a **code sample** or an **executable test case** demonstrating the expected behavior and/or problem.

- Be sure to select the appropriate labels (see *Issue Labels, Projects, and Milestones*) for your tickets so that they can be processed accordingly.
- NWB:N is currently being developed primarily by staff at scientific research institutions and industry, most of which work on many different research projects. Please be patient, if our development team is not able to respond immediately to your issues. In particular issues that belong to later project milestones may not be reviewed or processed until work on that milestone begins.

### 4.2.2 Did you write a patch that fixes a bug or implements a new feature?

See the `Contributing Patches and Changes` section below for details.

### 4.2.3 Did you fix whitespace, format code, or make a purely cosmetic patch in source code?

Source code changes that are purely cosmetic in nature and do not add anything substantial to the stability, functionality, or testability will generally not be accepted unless they have been approved beforehand. One of the main reasons is that there are a lot of hidden costs in addition to writing the code itself, and with the limited resources of the project, we need to optimize developer time. E.g,. someone needs to test and review PRs, backporting of bug fixes gets harder, it creates noise and pollutes the git repo and many other cost factors.

### 4.2.4 Do you have questions about NWB:N?

Join the NWB:N mailing list for updates or ask questions on our google group.

### 4.2.5 Informal discussions between developers, users, and team?

The https://nwb-users.slack.com slack is currently used mainly for internal discussions between developers, users, and teams.

## 4.3 Contributing Patches and Changes

To contribute to the PyNWB and nwb_schema, you must submit your changes to the `dev` branch via a Pull Request.

From your local copy directory, use the following commands.

1. First create a new branch to work on

```
$ git checkout -b <new_branch>
```

2. Make your changes.
3. Push your feature branch to origin (i.e. github)

```
$ git push origin <new_branch>
```

4. Once you have tested and finalized your changes, create a pull request targeting `dev` as the base branch:
   - Ensure the PR description clearly describes the problem and solution.
   - Include the relevant issue number if applicable.

- Before submitting, please ensure that the code follows the standard coding style of the repsective repository.
- **NOTE:** Contributed branches will be removed by the development team after the merge is complete and should, hence, not be used after the pull request is complete.

## 4.4 Issue Labels, Projects, and Milestones

### 4.4.1 Labels

Labels are used to describe the general scope of an issue, e.g., whether it describes a bug or feature request etc. Please review and select the appropriate labels for the respective Git repository:

- PyNWB issue labels
- nwb-schema issue labels

### 4.4.2 Milestones

Milestones are used to define the scope and general timeline for issues. Please review and select the appropriate milestones for the respective Git repository:

- PyNWB milestones
- nwb-schema milestones

### 4.4.3 Projects

Projects are currently used mainly on the NeurodataWithoutBorders organization level and are only accessible to members of the organization. Projects are used to plan and organize developments across repositories. We currently do not use projects on the individual repository level, although that might change in the future.

## 4.5 Styleguides

### 4.5.1 Git Commit Message Styleguide

- Use the present tense ("Add feature" not "Added feature")
- The first should be short and descriptive.
- Additional details may be included in further paragraphs.
- If a commit fixes an issues, then include "Fix #X" where X is the number of the issue.
- Reference relevant issues and pull requests liberally after the first line.

### 4.5.2 Documentation Styleguide

All documentations is written in reStrcturedText (RST) using Sphinx.

### 4.5.3 Format Specification Styleguide

**Coming soon**

### 4.5.4 Python Code Styleguide

**Coming soon**

CHAPTER 5

# Licence and Copyright

See the Readme and correspoding licence files for details about the copyright and licence.

# CHAPTER 6

# Introduction

PyNWB provides a high-level Python API for reading and writing NWB formatted HDF5 files. This section will provide a broad overview of the functionality provided for reading and writing neurophysiology data into NWB files.

CHAPTER 7

# Software Architecture

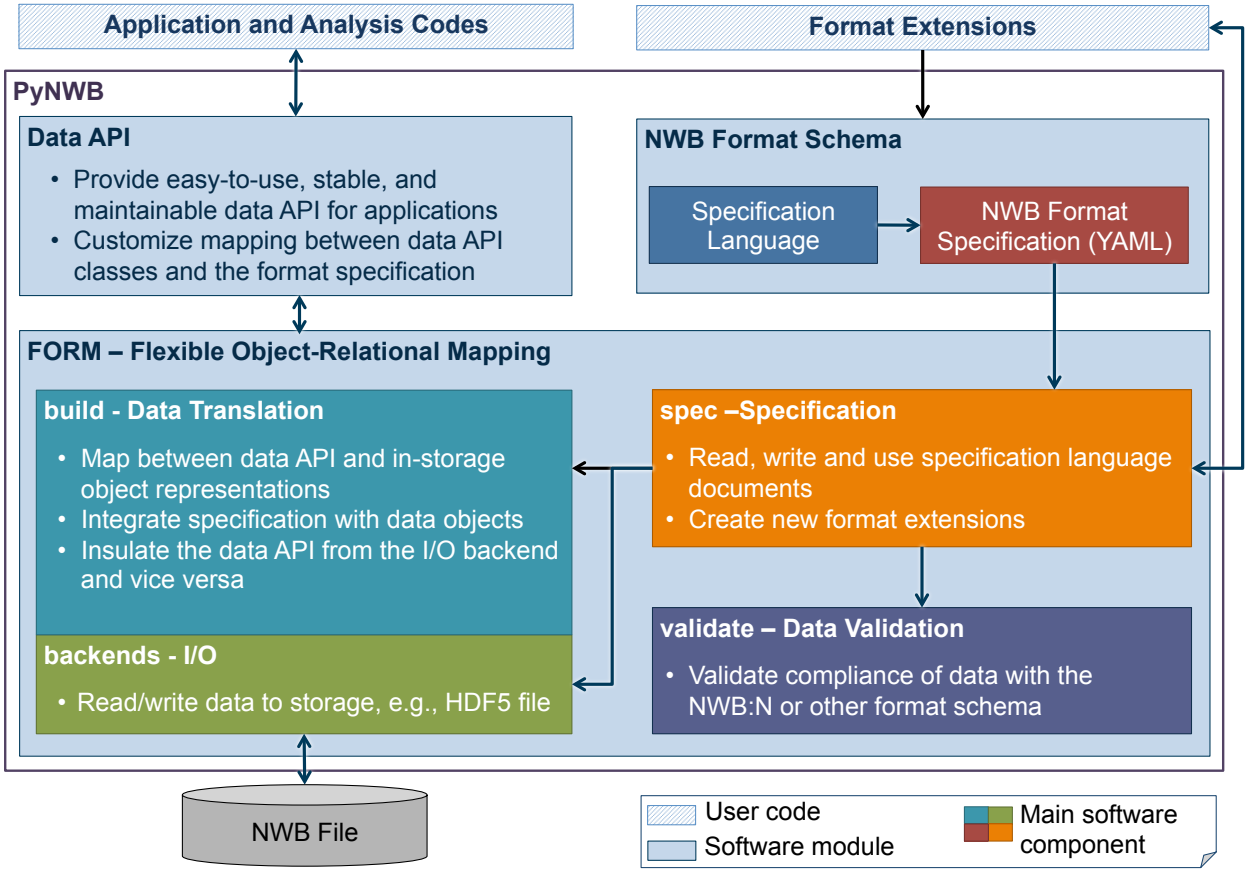


Fig. 7.1: Overview of the high-level software architecture of PyNWB.

CHAPTER 8

# NWB Format

The NWB Format is built around two concepts: *TimeSeries* and *Modules*.

*TimeSeries* are objects for storing time series data, and *Modules* are objects for storing and grouping analyses. The following sections describe these classes in further detail.

CHAPTER 9

# NWBFile

NWB files are represented in PyNWB with *NWBFile* objects. *NWBFile* objects provide functionality for creating *TimeSeries* datasets and *Modules*, as well as functionality for storing experimental metadata and other metadata related to data provenance.

## 9.1 TimeSeries

TimeSeries objects store time series data. These Python objects correspond to TimeSeries specifications provided by the NWB format specification. Like the NWB specification, TimeSeries Python objects follow an object-oriented inheritance pattern. For example, the class *TimeSeries* serves as the base class for all other TimeSeries types.

The following TimeSeries objects are provided by the API and NWB specification:

- *TimeSeries* - a general
  - *ElectricalSeries*
    - *SpikeEventSeries*
  - *AnnotationSeries*
  - *AbstractFeatureSeries*
  - *ImageSeries*
    - *ImageMaskSeries*
    - *OpticalSeries*
    - *TwoPhotonSeries*
  - *IndexSeries*
  - *IntervalSeries*
  - OptogeneticSeries
  - *PatchClampSeries*

        * *CurrentClampSeries*
            · *IZeroClampSeries*
        * *CurrentClampStimulusSeries*
        * *VoltageClampSeries*
        * *VoltageClampStimulusSeries*
    - *RoiResponseSeries*
    - *SpatialSeries*

## 9.2 Modules

Modules are objects that group together common analyses done during processing of data. Module objects are unique collections of analysis results. To standardize the storage of common analyses, NWB provides the concept of an *Interface*, where the output of common analyses are represented as objects that extend the `Interface` class. In most cases, you will not need to interact with the `Interface` class directly. More commonly, you will be creating instances of classes that extend this class. For example, a common analysis step for spike data (represented in NWB as a *SpikeEventSeries* object) is spike clustering. In NWB, the result of kind of analysis will be reprsented with a *Clustering* object.

The following Interface objects are provided by the API and NWB specification:

- `Interface`
    - *BehavioralEpochs*
    - *BehavioralEvents*
    - *BehavioralTimeSeries*
    - *ClusterWaveforms*
    - *Clustering*
    - *CompassDirection*
    - *DfOverF*
    - *EventDetection*
    - *EventWaveform*
    - *EyeTracking*
    - *FeatureExtraction*
    - *FilteredEphys*
    - *Fluorescence*
    - *ImageSegmentation*
    - *ImagingRetinotopy*
    - *LFP*
    - *MotionCorrection*
    - *Position*

# CHAPTER 10

# Examples

The following examples will reference variables that may not be defined within the block they are used in. For clarity, we define them here:

```
import numpy as np

rate = 10.0
np.random.seed(1234)
data_len = 1000
ephys_data = np.random.rand(data_len)
ephys_timestamps = np.arange(data_len) / rate
spatial_timestamps = ephys_timestamps[::10]
spatial_data = np.cumsum(np.random.normal(size=(2, len(spatial_timestamps))), axis=-
↪1).T
```

## 10.1 Creating and Writing NWB files

When creating a NWB file, the first step is to create the *NWBFile*. The first argument is the name of the NWB file, and the second argument is a brief description of the dataset.

```
from datetime import datetime
from pynwb import NWBFile

f = NWBFile('the PyNWB tutorial', 'my first synthetic recording', 'EXAMPLE_ID',␣
↪datetime.now(),
            experimenter='Dr. Bilbo Baggins',
            lab='Bag End Laboratory',
            institution='University of Middle Earth at the Shire',
            experiment_description='I went on an adventure with thirteen dwarves to␣
↪reclaim vast treasures.',
            session_id='LONELYMTN')
```

Once you have created your NWB and added all of your data and other necessary metadata, you can write it to disk using the *HDF5IO* class.

```
from pynwb import get_manager
from pynwb.form.backends.hdf5 import HDF5IO

filename = "example.h5"
io = HDF5IO(filename, manager=get_manager(), mode='w')
io.write(f)
io.close()
```

### 10.1.1 Creating Epochs

Experimental epochs are represented with *Epoch* objects. To create epochs for an NWB file, you can use the *NWBFile* instance method *create_epoch*.

```
epoch_tags = ('example_epoch',)

ep1 = f.create_epoch(source='an hypothetical source', name='epoch1', start=0.0,␣
↪stop=1.0,
                     tags=epoch_tags,
                     description="the first test epoch")

ep2 = f.create_epoch(source='an hypothetical source', name='epoch2', start=0.0,␣
↪stop=1.0,
                     tags=epoch_tags,
                     description="the second test epoch")
```

### 10.1.2 Creating Electrode Groups

Electrode groups (i.e. experimentally relevant groupings of channels) are represented by *ElectrodeGroup* objects. To create an electrode group, you can use the *NWBFile* instance method *create_electrode_group*.

Before creating an *ElectrodeGroup*, you need to provide some information about the device that was used to record from the electrode. This is done by creating a *Device* object using the instance method *create_device*.

```
device = f.create_device(name='trodes_rig123', source="a source")
```

Once you have created the *Device*, you can create the *ElectrodeGroup*.

```
electrode_name = 'tetrode1'
source = "an hypothetical source"
description = "an example tetrode"
location = "somewhere in the hippocampus"

electrode_group = f.create_electrode_group(electrode_name,
                                           source=source,
                                           description=description,
                                           location=location,
                                           device=device)
```

Finally, you can then create the associated *ElectrodeTable* and *ElectrodeTableRegion*.

```
for idx in [1, 2, 3, 4]:
    f.add_electrode(idx,
                    x=1.0, y=2.0, z=3.0,
                    imp=float(-idx),
                    location='CA1', filtering='none',
                    description='channel %s' % idx, group=electrode_group)

electrode_table_region = f.create_electrode_table_region([0, 2], 'the first and third␣
↪electrodes')
```

### 10.1.3 Creating TimeSeries

TimeSeries objects can be created by instantiating *TimeSeries* objects directly and then adding them to the *NWBFile* using the instance method *add_acquisition*.

This first example will demonstrate instantiating two different types of *TimeSeries* objects directly, and adding them with *add_acquisition*.

```
from pynwb.ecephys import ElectricalSeries
from pynwb.behavior import SpatialSeries

ephys_ts = ElectricalSeries('test_ephys_data',
                            'an hypothetical source',
                            ephys_data,
                            electrode_table_region,
                            timestamps=ephys_timestamps,
                            # Alternatively, could specify starting_time and rate as␣
↪follows
                            # starting_time=ephys_timestamps[0],
                            # rate=rate,
                            resolution=0.001,
                            comments="This data was randomly generated with numpy,␣
↪using 1234 as the seed",
                            description="Random numbers generated with numpy.random.
↪rand")
f.add_acquisition(ephys_ts, [ep1, ep2])

spatial_ts = SpatialSeries('test_spatial_timeseries',
                           'a stumbling rat',
                           spatial_data,
                           'origin on x,y-plane',
                           timestamps=spatial_timestamps,
                           resolution=0.1,
                           comments="This data was generated with numpy, using 1234␣
↪as the seed",
                           description="This 2D Brownian process generated with "
                                       "np.cumsum(np.random.normal(size=(2,␣
↪len(spatial_timestamps))), axis=-1).T")
f.add_acquisition(spatial_ts, [ep1, ep2])
```

### 10.1.4 Using Extensions

The NWB file format supports extending existing data types (See *Extending NWB* for more details on creating extensions). Extensions must be registered with PyNWB to be used for reading and writing of custom neurodata types.

The following code demonstrates how to load custom namespaces.

```
from pynwb import load_namespaces
namespace_path = 'my_namespace.yaml'
load_namespaces(namespace_path)
```

**Note:** This will register all namespaces defined in the file `'my_namespace.yaml'`.

To read and write custom data, corresponding *NWBContainer* classes must be associated with their respective specifications. *NWBContainer* classes are associated with their respective specification using the decorator *register_class*.

The following code demonstrates how to associate a specification with the *NWBContainer* class that represents it.

```
from pynwb import register_class
@register_class('my_namespace', 'MyExtension')
class MyExtensionContainer(NWBContainer):
    ...
```

*register_class* can also be used as a function.

```
from pynwb import register_class
class MyExtensionContainer(NWBContainer):
    ...
register_class('my_namespace', 'MyExtension', MyExtensionContainer)
```

If your *NWBContainer* extension requires custom mapping of the *NWBContainer* class for reading and writing, you will need to implement and register a custom *ObjectMapper*.

*ObjectMapper* extensions are registered with the decorator *register_map*.

```
from pynwb import register_map
from form import ObjectMapper
@register_map(MyExtensionContainer)
class MyExtensionMapper(ObjectMapper)
    ...
```

*register_map* can also be used as a function.

```
from pynwb import register_map
from form import ObjectMapper
class MyExtensionMapper(ObjectMapper)
    ...
register_map(MyExtensionContainer, MyExtensionMapper)
```

If you do not have an *NWBContainer* subclass to associate with your extension specification, a dynamically created class is created by default.

To use the dynamic class, you will need to retrieve the class object using the function *get_class*. Once you have retrieved the class object, you can use it just like you would a statically defined class.

```
from pynwb import get_class
MyExtensionContainer = get_class('my_namespace', 'MyExtension')
my_ext_inst = MyExtensionContainer(...)
```

If using iPython, you can access documentation for the class's constructor using the help command.

### 10.1.5 Write an NWBFile

Writing NWB files to disk is handled by the *pynwb.form* package. Currently, the only storage format supported by *pynwb.form* is HDF5.

Reading and writing to and from HDF5 is handled by the class *HDF5IO*. The only required argument to this is the path of the HDF5 file. A second, optional argument is the *BuildManager* to use for IO.

Briefly, the *BuildManager* is a class that manages objects to be read and written from disk. A PyNWB-specific BuildManager can be retrieved using the module-level function *get_manager*.

Alternatively, the *BuildManager* that a *FORMIO* used can be retrieved from the *manager* attribute.

```
from datetime import datetime

from pynwb import NWBFile, TimeSeries, get_manager
from pynwb.form.backends.hdf5 import HDF5IO

start_time = datetime(1970, 1, 1, 12, 0, 0)
create_date = datetime(2017, 4, 15, 12, 0, 0)

nwbfile = NWBFile('the PyNWB tutorial', 'a test NWB File', 'TEST123', start_time,
                  file_create_date=create_date)

ts = TimeSeries('test_timeseries', 'example_source', list(range(100, 200, 10)),
↪'SIunit',
                timestamps=list(range(10)),
                resolution=0.1)

nwbfile.add_acquisition(ts)

io = HDF5IO("example.h5", manager=get_manager(), mode='w')
io.write(nwbfile)
io.close()
```

**Note:** All *FORMIO* objects are context managers.

The third argument to the *HDF5IO* constructor is the mode for opening the HDF5 file. Valid modes are:

| | |
|---|---|
| r | Readonly, file must exist |
| r+ | Read/write, file must exist |
| w | Create file, truncate if exists |
| w- or x | Create file, fail if exists |
| a | Read/write if exists, create otherwise (default) |

## 10.2 Extending NWB

### 10.2.1 Creating new Extensions

The NWB specification is designed to be extended. Extension for the NWB format can be done so using classes provided in the *pynwb.spec* module. The classes *NWBGroupSpec*, *NWBDatasetSpec*, *NWBAttributeSpec*, and *NWBLinkSpec* can be used to define custom types.

### Attribute Specifications

Specifying attributes is done with *NWBAttributeSpec*.

```
from pynwb.spec import NWBAttributeSpec

spec = NWBAttributeSpec('bar', 'float', 'a value for bar')
```

### Dataset Specifications

Specifying datasets is done with *NWBDatasetSpec*.

```
from pynwb.spec import NWBDatasetSpec

spec = NWBDatasetSpec('A custom NWB type',
                      attribute=[
                          NWBAttributeSpec('baz', 'str', 'a value for baz'),
                      ],
                      shape=(None, None))
```

### Using datasets to specify tables

Tables can be specified using *NWBDtypeSpec*. To specify a table, provide a list of *NWBDtypeSpec* objects to the *dtype* argument.

```
from pynwb.spec import NWBDatasetSpec, NWBDtypeSpec

spec = NWBDatasetSpec('A custom NWB type',
                      attribute=[
                          NWBAttributeSpec('baz', 'str', 'a value for baz'),
                      ],
                      dtype=[
                          NWBDtypeSpec('foo', column for foo', 'int'),
                          NWBDtypeSpec('bar', 'a column for bar', 'float')
                      ])
```

Compound data types can be nested.

```
from pynwb.spec import NWBDatasetSpec, NWBDtypeSpec

spec = NWBDatasetSpec('A custom NWB type',
                      attribute=[
                          NWBAttributeSpec('baz', 'a value for baz', 'str'),
                      ],
                      dtype=[
                          NWBDtypeSpec('foo', 'a column for foo', 'int'),
                          NWBDtypeSpec('bar', 'a column for bar', 'float')
                      ])
```

### Group Specifications

Specifying groups is done with the *NWBGroupSpec* class.

```
from pynwb.spec import NWBGroupSpec

# A list of NWBAttributeSpec objects to specify new attributes
addl_attributes = [...]
# A list of NWBDatasetSpec objects to specify new datasets
addl_datasets = [...]
# A list of NWBDatasetSpec objects to specify new groups
addl_groups = [...]
spec = NWBGroupSpec('A custom NWB type',
                    attributes = addl_attributes,
                    datasets = addl_datasets,
                    groups = addl_groups)
```

### Neurodata Type Specifications

*NWBGroupSpec* and *NWBDatasetSpec* use the arguments *neurodata_type_inc* and *neurodata_type_def* for declaring new types and extending existing types. New types are specified by setting the argument *neurodata_type_def*. New types can extend an existing type by specifying the argument *neurodata_type_inc*.

Create a new type

```
from pynwb.spec import NWBGroupSpec

# A list of NWBAttributeSpec objects to specify new attributes
addl_attributes = [...]
# A list of NWBDatasetSpec objects to specify new datasets
addl_datasets = [...]
# A list of NWBDatasetSpec objects to specify new groups
addl_groups = [...]
spec = NWBGroupSpec('A custom NWB type',
                    attributes = addl_attributes,
                    datasets = addl_datasets,
                    groups = addl_groups,
                    neurodata_type_def='MyNewNWBType')
```

Extend an existing type

```
from pynwb.spec import NWBGroupSpec

# A list of NWBAttributeSpec objects to specify additional attributes or attributes␣
↪to be overriden
addl_attributes = [...]
# A list of NWBDatasetSpec objects to specify additional datasets or datasets to be␣
↪overriden
addl_datasets = [...]
# A list of NWBGroupSpec objects to specify additional groups or groups to be␣
↪overriden
addl_groups = [...]
spec = NWBGroupSpec('An extended NWB type',
                    attributes = addl_attributes,
                    datasets = addl_datasets,
                    groups = addl_groups,
                    neurodata_type_inc='Clustering',
                    neurodata_type_def='MyExtendedClustering')
```

Existing types can be instantiate by specifying *neurodata_type_inc* alone.

```
from pynwb.spec import NWBGroupSpec

# use another NWBGroupSpec object to specify that a group of type
# ElectricalSeries should be present in the new type defined below
addl_groups = [ NWBGroupSpec('An included ElectricalSeries instance',
                             neurodata_type_inc='ElectricalSeries') ]

spec = NWBGroupSpec('An extended NWB type',
                    groups = addl_groups,
                    neurodata_type_inc='Clustering',
                    neurodata_type_def='MyExtendedClustering')
```

Datasets can be extended in the same manner (with regard to *neurodata_type_inc* and *neurodata_type_def*, by using the class *NWBDatasetSpec*.

### 10.2.2 Saving Extensions

Extensions are used by including them in a loaded namespace. Namespaces and extensions need to be saved to file for downstream use. The class *NWBNamespaceBuilder* can be used to create new namespace and specification files.

**Note:** When using *NWBNamespaceBuilder*, the core NWB namespace is automatically included

Create a new namespace with extensions

```
from pynwb.spec import NWBGroupSpec, NWBNamespaceBuilder

# create a builder for the namespace
ns_builder = NWBNamespaceBuilder("Extension for use in my laboratory", "mylab", ...)

# create extensions
ext1 = NWBGroupSpec('A custom Clustering interface',
                    attributes = [...]
                    datasets = [...],
                    groups = [...],
                    neurodata_type_inc='Clustering',
                    neurodata_type_def='MyExtendedClustering')

ext2 = NWBGroupSpec('A custom ClusterWaveforms interface',
                    attributes = [...]
                    datasets = [...],
                    groups = [...],
                    neurodata_type_inc='ClusterWaveforms',
                    neurodata_type_def='MyExtendedClusterWaveforms')


# add the extension
ext_source = 'mylab.specs.yaml'
ns_builder.add_spec(ext_source, ext1)
ns_builder.add_spec(ext_source, ext2)

# include an existing namespace - this will include all specifications in that␣
↪namespace
ns_builder.include_namespace('collab_ns')

# save the namespace and extensions
```

```
ns_path = 'mylab.namespace.yaml'
ns_builder.export(ns_path)
```

**Tip:** Using the API to generate extensions (rather than writing YAML sources directly) helps avoid errors in the specification (e.g., due to missing required keys or invalid values) and ensure compliance of the extension definition with the NWB specification language. It also helps with maintanence of extensions, e.g., if extensions have to be ported to newer versions of the specification language in the future.

### 10.2.3 Documenting Extensions

Using the same tools used to generate the documentation for the NWB-N core format one can easily generate documentation in HTML, PDF, ePub and many other format for extensions as well.

For the purpose of this example we assume that our current directory has the following structure.

```
- nwb_schema (cloned from `https://github.com/NeurodataWithoutBorders/nwb-schema`)
- my_extension/
    - my_extension_source/
        - mylab.namespace.yaml
        - mylab.specs.yaml
        - ...
        - docs/  (Optional)
            - mylab_description.rst
            - mylab_release_notes.rst
```

In addition to Python 3.x you will also need `sphinx` (including the `sphinx-quickstart` tool) installed. Sphinx is availble here http://www.sphinx-doc.org/en/stable/install.html .

We can now create the sources of our documentation as follows:

```
python3 nwb-schema/docs/utils/init_sphinx_extension_doc.py \
             --project test \
             --author "Dr. Master Expert" \
             --version "1.2.3" \
             --release alpha \
             --output my_extension_docs \
             --spec_dir my_extension_source \
             --namespace_filename mylab.namespace.yaml \
             --default_namespace mylab
             --external_description my_extension_source/docs/mylab_description.rst \ ␣
↪(Optional)
             --external_release_notes my_extension_source/docs/mylab_release_notes.
↪rst \  (Optional)
```

The new folder `my_extension_docs/` now contains the basic setup for the documentation. To automatically generate the RST documentation files from the YAML (or JSON) sources of the extension simply run:

```
cd my_extension_docs
make apidoc
```

Finally, to generate the HTML version of the docs run:

```
make html
```

**Tip:** Additional instructions for how to use and customize the extension documentations are also available in the `Readme.md` file that `init_sphinx_extension_doc.py` automatically adds to the docs.

**Tip:** See `make help` for a list of available options for building the documentation in many different output formats (e.g., PDF, ePub, LaTeX, etc.).

**Tip:** See `python3 init_sphinx_extension_doc.py --help` for a complete list of option to customize the documentation directly during initialization.

**Tip:** The above example included additional description and release note docs as part of the specification. These are included in the docs via `.. include` commands so that changes in those files are automatically picked up when rebuilding to docs. Alternatively, we can also add custom documentation directly to the docs. In this case the options `--custom_description format_description.rst` and `--custom_release_notes format_release_notes.rst` of the `init_sphinx_extension_doc.py` script are useful to automatically generate the basic setup for those files so that one can easily start to add content directly without having to worry about the additional setup.

### 10.2.4 Further Reading

- **Using Extensions:** See *Using Extensions* for an example on how to use extensions during read and write.
- **Specification Language:** For a detailed overview of the specification language itself see https://schema-language.readthedocs.io/en/latest/

## 10.3 Validating NWB files

Validating NWB files is handled by a command-line tool availble in *pynwb*. The validator can be invoked like so:

```
python -m pynwb.validate test.nwb
```

This will validate the file `test.nwb` against the *core* NWB specification. Validating against other specifications i.e. extensions can be done using the `-p` and `-n` flags. For example, the following command will validate against the specifications referenced in the namespace file `mylab.namespace.yaml` in addition to the core specification.

```
python -m pynwb.validate -p mylab.namespace.yaml test.nwb
```

CHAPTER 11

# Tutorials

## 11.1 Extensions

The NWB-N format was designed to be easily extendable. Here we will demonstrate how to extend NWB using the PyNWB API.

### 11.1.1 Defining extensions

Extensions should be defined separately from the code that uses the extensions. This design decision is based on the assumption that extension will be written once, and read or used multiple times. Here, we provide an example of how to create an extension for subsequent use. (For more information on the available tools for creating extensions, see *Extending NWB*).

The following block of code demonstrates how to create a new namespace, and then add a new *neurodata_type* to this namespace. Finally, it calls `export` to save the extensions to disk for downstream use.

```
from pynwb import NWBNamespaceBuilder, NWBGroupSpec, NWBAttributeSpec

ns_path = "mylab.namespace.yaml"
ext_source = "mylab.extensions.yaml"

ns_builder = NWBNamespaceBuilder('Extension for use in my Lab', "mylab")
ext = NWBGroupSpec('A custom ElectricalSeries for my lab',
                   attributes=[NWBAttributeSpec('trode_id', 'int', 'the tetrode id')],
                   neurodata_type_inc='ElectricalSeries',
                   neurodata_type_def='TetrodeSeries')

ns_builder.add_spec(ext_source, ext)
ns_builder.export(ns_path)
```

Running this block will produce two YAML files.

The first file contains the specification of the namespace.

```
# mylab.namespace.yaml
namespaces:
- doc: Extension for use in my Lab
  name: mylab
  schema:
  - namespace: core
  - source: fake_extension.yaml
```

The second file contains the details on newly defined types.

```
# mylab.extensions.yaml
groups:
- attributes:
  - doc: the tetrode id
    dtype: int
    name: trode_id
  doc: A custom ElectricalSeries for my lab
  neurodata_type_def: TetrodeSeries
  neurodata_type_inc: ElectricalSeries
```

**Tip:** Detailed documentation of all components and *neurodata_types* that are part of the core schema of NWB:N are available in the schema docs at http://nwb-schema.readthedocs.io . Before creating a new type from scratch, please have a look at the schema docs to see if using or extending an existing type may solve your problem. Also, the schema docs are helpful when extending an existing type to better understand the design and structure of the neurodata_type you are using.

### 11.1.2 Using extensions

After an extension has been created, it can be used by downstream codes for reading and writing data. There are two main mechanisms for reading and writing extension data with PyNWB. The first involves defining new *NWBContainer* classes that are then mapped to the neurodata types in the extension.

```
from pynwb import register_class, load_namespaces
from pynwb.ecephys import ElectricalSeries

ns_path = "mylab.namespace.yaml"
load_namespaces(ns_path)

@register_class('mylab', 'TetrodeSeries')
class TetrodeSeries(ElectricalSeries):
    __nwbfields__ = ('tetrode_id',)

    def __init__(self, ...):
        ...
```

**Note:** Although it is not used here, it is encouraged to use the `docval` decorator for documenting constructors, methods, and functions.

When extending *NWBContainer* or *NWBContainer* subclasses, you should defining the class field `__nwbfields__`. This will tell PyNWB the properties of the *NWBContainer* extension.

If you do not want to write additional code to read your extensions, PyNWB is able to dynamically create an

*NWBContainer* subclass for use within the PyNWB API. Dynamically created classes can be inspected using the built-in `help` or the `inspect` module.

```
from pynwb import get_class, load_namespaces

ns_path = "mylab.namespace.yaml"
load_namespaces(ns_path)

TetrodeSeries = get_class('TetrodeSeries', 'mylab')
```

**Note:** When defining your own *NWBContainer*, the subclass name does not need to be the same as the extension type name. However, it is encouraged to keep class and extension names the same for the purposes of readibility.

## 11.2 Converting data to NWB

The following are example Jupyter notebooks for converting custom lab data to NWB:

### 11.2.1 crcns-ret-1: Meister lab retina data

- **Notebook:** https://github.com/NeurodataWithoutBorders/pynwb/blob/dev/docs/notebooks/convert-crcns-ret-1-meisterlab.ipynb
- **Example:** This example shows:
  - Use of `UnitTimes`, `SpikeUnit`, `ImageSeries`, `ElectrodeGroup`, `EpochTimeSeries`, `Device`
  - Creation and use of custom namespace and extension to extend `ImageSeries` to add custom metadata attributes
  - Create external link for `ImageSeries.data`
  - Read of crcns-ret-1 dataset
  - Convert of the data to NWB
  - Comparison of H5Gate and PyNWB
- **Data:** Convert single-unit neural responses recorded from isolated retina from lab mice (Mus Musculus) using a 61-electrode array in response to various visual stimuli. Recordings were done by Yifeng Zhang in Markus Meister's lab at Harvard University in 2008. Further description of the data are available here: https://crcns.org/data-sets/retina/ret-1/about-ret-1
- **Data Terms of use** The data used on the notebook and, therefore also the NWB files generated by the notebook are governed by the terms-of-use of CRCNS.org described here https://crcns.org/terms .
- **Comparison to NWB 1.0.x':**
  - **Notebook:** https://github.com/NeurodataWithoutBorders/pynwb/blob/dev/docs/notebooks/convert-crcns-ret-1-meisterlab-compare-nwb-1.0.6.ipynb
  - **Description:** This notebook shows the convert of the same data using the orginal NWB 1.0.x API to allow for comparison of the NWB 1.0.x and NWB 2.x file.

# CHAPTER 12

# API Documentation

## 12.1 pynwb.file module

**class** pynwb.file.**Image**
Bases: *pynwb.form.container.Container*

**class** pynwb.file.**SpecFile**
Bases: *pynwb.form.container.Container*

**class** pynwb.file.**NWBFile**(*source*, *session_description*, *identifier*, *session_start_time*, *file_create_date=None*, *version=None*, *experimenter=None*, *experiment_description=None*, *session_id=None*, *institution=None*, *lab=None*, *acquisition=None*, *stimulus=None*, *stimulus_template=None*, *epochs=None*, *modules=None*, *ec_electrodes=None*, *ec_electrode_groups=None*, *ic_electrodes=None*, *imaging_planes=None*, *optogenetic_sites=None*, *devices=None*)
Bases: *pynwb.core.NWBContainer*

A representation of an NWB file.

**Parameters**

- **source** (*str*) – the source of the data
- **session_description** (*str*) – a description of the session where this data was generated
- **identifier** (*str*) – a unique text identifier for the file
- **session_start_time** (*datetime or str*) – the start time of the recording session
- **file_create_date** (*list or datetime or str*) – the time the file was created and subsequent modifications made
- **version** (*str*) – the NWB version
- **experimenter** (*str*) – name of person who performed experiment

- **experiment_description** (*str*) – general description of the experiment
- **session_id** (*str*) – lab-specific ID for the session
- **institution** (*str*) – institution(s) where experiment is performed
- **lab** (*str*) – lab where experiment was performed
- **acquisition** (*list or tuple*) – Raw TimeSeries objects belonging to this NWBFile
- **stimulus** (*list or tuple*) – Stimulus TimeSeries objects belonging to this NWBFile
- **stimulus_template** (*list or tuple*) – Stimulus template TimeSeries objects belonging to this NWBFile
- **epochs** (*list or tuple*) – Epoch objects belonging to this NWBFile
- **modules** (*list or tuple*) – ProcessingModule objects belonging to this NWBFile
- **ec_electrodes** (*ElectrodeTable or Iterable*) – the ElectrodeTable that belongs to this NWBFile
- **ec_electrode_groups** (*Iterable*) – the ElectrodeGroups that belong to this NWBFile
- **ic_electrodes** (*list or tuple*) – IntracellularElectrodes that belong to this NWBFile
- **imaging_planes** (*list or tuple*) – ImagingPlanes that belong to this NWBFile
- **optogenetic_sites** (*list or tuple*) – OptogeneticStimulusSites that belong to this NWBFile
- **devices** (*list or tuple*) – Device objects belonging to this NWBFile

**classmethod set_version**(*version*)

**devices**

**epochs**

**epoch_tags**

**modules**

**identifier**

**nwb_version**

**session_description**

**file_create_date**

**session_start_time**

**acquisition**

**stimulus**

**stimulus_template**

**ec_electrodes**

**add_electrode**(*id*, *x*, *y*, *z*, *imp*, *location*, *filtering*, *description*, *group*)

**Parameters**

- **id** (*int*) – a unique identifier for the electrode
- **x** (*float*) – the x coordinate of the position
- **y** (*float*) – the y coordinate of the position
- **z** (*float*) – the z coordinate of the position
- **imp** (*float*) – the impedance of the electrode
- **location** (*str*) – the location of electrode within the subject e.g. brain region
- **filtering** (*str*) – description of hardware filtering
- **description** (*str*) – a brief description of what this electrode is
- **group** (*ElectrodeGroup*) – the ElectrodeGroup object to add to this NWBFile

**create_electrode_table_region**(*region*, *description*, *name='electrodes'*)

**Parameters**

- **region** (*slice or list or tuple or RegionReference*) – the indices of the table
- **description** (*str*) – a brief description of what this electrode is
- **name** (*str*) – the name of this container

**ec_electrode_groups**

**ic_electrodes**

**imaging_planes**

**create_imaging_plane**(*name*, *source*, *optical_channel*, *description*, *device*, *excitation_lambda*, *imaging_rate*, *indicator*, *location*, *manifold=None*, *conversion=None*, *unit=None*, *reference_frame=None*)

Add metadata about an imaging plane

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **optical_channel** (*list or OpticalChannel*) – One of possibly many groups storing channelspecific data.
- **description** (*str*) – Description of this ImagingPlane.
- **device** (*str*) – Name of device in /general/devices
- **excitation_lambda** (*str*) – Excitation wavelength.
- **imaging_rate** (*str*) – Rate images are acquired, in Hz.
- **indicator** (*str*) – Calcium indicator
- **location** (*str*) – Location of image plane.
- **manifold** (*Iterable*) – Physical position of each pixel. height, weight, x, y, z.
- **conversion** (*float*) – Multiplier to get from stored values to specified unit (e.g., 1e-3 for millimeters)
- **unit** (*str*) – Base unit that coordinates are stored in (e.g., Meters).
- **reference_frame** (*str*) – Describes position and reference frame of manifold based on position of first element in manifold.

**Returns** the imaging plane

**Return type** ImagingPlane

**set_imaging_plane**(*imaging_plane*)
Add an ImagingPlane object to this file

**Parameters** **imaging_plane** (*ImagingPlane*) – the ImagingPlane object to add to this NWBFile

**get_imaging_plane**(*name*)
Get an ImagingPlane object from this file

**Parameters** **name** (*str*) – the name of the imaging plane

**is_acquisition**(*ts*)

**is_stimulus**(*ts*)

**is_stimulus_template**(*ts*)

**create_epoch**(*name*, *source*, *start*, *stop*, *tags=[]*, *description=None*)

**Creates a new Epoch object. Epochs are used to track intervals** in an experiment, such as exposure to a certain type of stimuli (an interval where orientation gratings are shown, or of sparse noise) or a different paradigm (a rat exploring an enclosure versus sleeping between explorations)

**Parameters**

- **name** (*str*) – the name of the epoch, as it will appear in the file
- **source** (*str*) – the source of the data
- **start** (*float*) – the starting time of the epoch
- **stop** (*float*) – the ending time of the epoch
- **tags** (*tuple or list*) – tags for this epoch
- **description** (*str*) – a description of this epoch

**get_epoch**(*name*)

**set_epoch_timeseries**(*epoch*, *timeseries*)
Add one or more TimSeries datasets to one or more Epochs

**Parameters**

- **epoch** (*str or Epoch or list or tuple*) – the name of an epoch or an Epoch object or a list of names of epochs or Epoch objects
- **timeseries** (*str or TimeSeries or list or tuple*) – the name of a timeseries or a TimeSeries object or a list of names of timeseries or TimeSeries objects

**add_acquisition**(*ts*, *epoch=None*)

**Parameters**

- **ts** (*TimeSeries*) – the TimeSeries object to add
- **epoch** (*str or Epoch or list or tuple*) – the name of an epoch or an Epoch object or a list of names of epochs or Epoch objects

**get_acquisition**(*name*)
Retrieve acquisition TimeSeries data

**Parameters** **name** (*str*) – the name of this TimeSeries

**add_stimulus**(*ts*, *epoch=None*)

**Parameters**

- **ts** (*TimeSeries*) – the TimeSeries object to add
- **epoch** (*str or Epoch*) – the name of an epoch or an Epoch object or a list of names of epochs or Epoch objects

**get_stimulus**(*name*)
Retrieve stimiulus TimeSeries data

**Parameters** **name** (*str*) – the name of this TimeSeries

**add_stimulus_template**(*ts*, *epoch=None*)

**Parameters**

- **ts** (*TimeSeries*) – the TimeSeries object to add
- **epoch** (*str or Epoch*) – the name of an epoch or an Epoch object or a list of names of epochs or Epoch objects

**get_stimulus_template**(*name*)
Retrieve stimiulus template TimeSeries data

**Parameters** **name** (*str*) – the name of this TimeSeries

**create_electrode_group**(*name*, *source*, *description*, *location*, *device*)
Add an electrode group (e.g. a probe, shank, tetrode).

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **description** (*str*) – description of this electrode group
- **location** (*str*) – description of location of this electrode group
- **device** (*Device*) – the device that was used to record from this electrode group

**Returns** the electrode group

**Return type** ElectrodeGroup

**set_electrode_group**(*electrode_grp*)

**Parameters** **electrode_grp** (*ElectrodeGroup*) – the ElectrodeGroup object to add to this NWBFile

**set_electrode_table**(*electrode_table*)

**Parameters** **electrode_table** (*ElectrodeTable*) – the ElectrodeTable for this file

**create_device**(*name*, *source*)

**Parameters**

- **name** (*str*) – the name of this device
- **source** (*str*) – the source of the data

**Returns** the recording device

**Return type** Device

**set_device**(*device*)

**Parameters** **device** (*Device*) – the Device object to add to this NWBFile

**get_electrode_group**(*name*)

**Parameters** **name** (*str*) – the name of the electrode group

**create_intracellular_electrode**(*name*, *source*, *slice*, *seal*, *description*, *location*, *resistance*, *filtering*, *initial_access_resistance*, *device*)

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **slice** (*str*) – Information about slice used for recording.
- **seal** (*str*) – Information about seal used for recording.
- **description** (*str*) – Recording description, description of electrode (e.g., whole-cell, sharp, etc) COMMENT: Free-form text (can be from Methods)
- **location** (*str*) – Area, layer, comments on estimation, stereotaxis coordinates (if in vivo, etc).
- **resistance** (*str*) – Electrode resistance COMMENT: unit: Ohm.
- **filtering** (*str*) – Electrode specific filtering.
- **initial_access_resistance** (*str*) – Initial access resistance.
- **device** (*str*) – Name(s) of devices in general/devices.

**Returns** the intracellular electrode

**Return type** IntracellularElectrode

**description**

**experiment_description**

**experimenter**

**institution**

**lab**

**optogenetic_sites**

**session_id**

**set_intracellular_electrode**(*ic_elec*)

**Parameters** **ic_elec** (*IntracellularElectrode*) – the IntracellularElectrode object to add to this NWBFile

**get_intracellular_electrode**(*name*)
Retrieve an IntracellularElectrode

**Parameters** **name** (*str*) – the name of the intracellular electrode

**create_processing_module**(*name*, *source*, *description*)

**Creates a ProcessingModule object of the specified name. NWBContainers can** be created by the module and will be stored inside it

**Parameters**

- **name** (*str*) – the name of the processing module

- **source** (*str*) – the source of the data
- **description** (*str*) – description of the processing module

**Returns** a processing module

**Return type** ProcessingModule

**set_processing_module**(*module*)
Add a ProcessingModule to this NWBFile

**Parameters** **module** (*ProcessingModule*) – the processing module to add to this file

**get_processing_module**(*name*)
Retrieve a ProcessingModule

**Parameters** **name** (*str*) – the name of the processing module

## 12.2 pynwb.ecephys module

**class** pynwb.ecephys.**Device**(*name*, *source*, *parent=None*)
Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this device
- **source** (*str*) – the source of the data
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**class** pynwb.ecephys.**ElectrodeGroup**(*name*, *source*, *description*, *location*, *device*, *parent=None*)
Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **description** (*str*) – description of this electrode group
- **location** (*str*) – description of location of this electrode group
- **device** (*Device*) – the device that was used to record from this electrode group
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**description**

**location**

**device**

**class** pynwb.ecephys.**ElectrodeTable**(*name*, *data=[]*)
Bases: *pynwb.core.NWBTable*

A table of all electrodes

**Parameters**

- **name** (*str*) – the name of this container
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – the source of the data

**add_row**(*id*, *x*, *y*, *z*, *imp*, *location*, *filtering*, *description*, *group*)

**Parameters**

- **id** (*int*) – a unique identifier for the electrode
- **x** (*float*) – the x coordinate of the position
- **y** (*float*) – the y coordinate of the position
- **z** (*float*) – the z coordinate of the position
- **imp** (*float*) – the impedance of the electrode
- **location** (*str*) – the location of electrode within the subject e.g. brain region
- **filtering** (*str*) – description of hardware filtering
- **description** (*str*) – a brief description of what this electrode is
- **group** (*ElectrodeGroup*) – the ElectrodeGroup object to add to this NWBFile

**class** pynwb.ecephys.**ElectrodeTableRegion**(*table*, *region*, *description*, *name='electrodes'*)

Bases: *pynwb.core.NWBTableRegion*

A subsetting of an ElectrodeTable

**Parameters**

- **table** (*ElectrodeTable*) – the ElectrodeTable this region applies to
- **region** (*slice or list or tuple or RegionReference*) – the indices of the table
- **description** (*str*) – a brief description of what this electrode is
- **name** (*str*) – the name of this container

**description**

**class** pynwb.ecephys.**ElectricalSeries**(*name*, *source*, *data*, *electrodes*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Stores acquired voltage data from extracellular recordings. The data field of an ElectricalSeries is an int or float array storing data in Volts. TimeSeries::data array structure: [num times] [num channels] (or [num_times] for single electrode).

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **electrodes** (*ElectrodeTableRegion*) – the table region corresponding to the electrodes from which this series was recorded
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data

- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**electrodes**

**class** pynwb.ecephys.**SpikeEventSeries**(*name*, *source*, *data*, *timestamps*, *electrodes*, *resolution=0.0*, *conversion=1.0*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.ecephys.ElectricalSeries*

Stores "snapshots" of spike events (i.e., threshold crossings) in data. This may also be raw data, as reported by ephys hardware. If so, the TimeSeries::description field should describing how events were detected. All SpikeEventSeries should reside in a module (under EventWaveform interface) even if the spikes were reported and stored by hardware. All events span the same recording channels and store snapshots of equal duration. TimeSeries::data array structure: [num events] [num channels] [num samples] (or [num events] [num samples] for single electrode).

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **electrodes** (*ElectrodeTableRegion*) – the table region corresponding to the electrodes from which this series was recorded
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value

- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**class** pynwb.ecephys.**EventDetection**(*source*, *detection_method*, *source_electricalseries*, *source_idx*, *times*, *name='EventDetection'*)

Bases: *pynwb.core.NWBContainer*

Detected spike events from voltage trace(s).

**Parameters**

- **source** (*str*) – the source of the data
- **detection_method** (*str*) – Description of how events were detected, such as voltage threshold, or dV/dT threshold, as well as relevant values.
- **source_electricalseries** (*ElectricalSeries*) – The source electrophysiology data
- **source_idx** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – Indices (zero-based) into source ElectricalSeries::data array corresponding to time of event. Module description should define what is meant by time of event (e.g., .25msec before action potential peak, zero-crossing time, etc). The index points to each event from the raw data
- **times** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – Timestamps of events, in Seconds
- **name** (*str*) – the name of this container

**detection_method**

**source_electricalseries**

**source_idx**

**times**

**class** pynwb.ecephys.**EventWaveform**(*source*, *spike_event_series*, *name='EventWaveform'*)

Bases: *pynwb.core.NWBContainer*

Spike data for spike events detected in raw data stored in this NWBFile, or events detect at acquisition

**Parameters**

- **source** (*str*) – the source of the data
- **spike_event_series** (*list or SpikeEventSeries*) – spiking event data
- **name** (*str*) – the name of this container

**spike_event_series**

**class** pynwb.ecephys.**Clustering**(*source*, *description*, *num*, *peak_over_rms*, *times*, *name='Clustering'*)

Bases: *pynwb.core.NWBContainer*

Specifies cluster event times and cluster metric for maximum ratio of waveform peak to RMS on any channel in cluster.

**Parameters**

- **source** (*str*) – The source of the data
- **description** (*str*) – Description of clusters or clustering, (e.g. cluster 0 is noise, clusters curated using Klusters, etc).

- **num** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – Cluster number of each event.
- **peak_over_rms** (*Iterable*) – Maximum ratio of waveform peak to RMS on any channel in the cluster (provides a basic clustering metric).
- **times** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – Times of clustered events, in seconds.
- **name** (*str*) – the name of this container

**class** pynwb.ecephys.**ClusterWaveforms**(*source*, *clustering_interface*, *waveform_filtering*, *waveform_mean*, *waveform_sd*, *name='ClusterWaveforms'*)

Bases: *pynwb.core.NWBContainer*

Describe cluster waveforms by mean and standard deviation for at each sample.

**Parameters**

- **source** (*str*) – the source of the data
- **clustering_interface** (*Clustering*) – the clustered spike data used as input for computing waveforms
- **waveform_filtering** (*str*) – filter applied to data before calculating mean and standard deviation
- **waveform_mean** (*Iterable*) – the mean waveform for each cluster
- **waveform_sd** (*Iterable*) – the standard deviations of waveforms for each cluster
- **name** (*str*) – the name of this container

**clustering_interface**

**waveform_filtering**

**waveform_mean**

**waveform_sd**

**class** pynwb.ecephys.**LFP**(*source*, *electrical_series*, *name='LFP'*)

Bases: *pynwb.core.NWBContainer*

LFP data from one or more channels. The electrode map in each published ElectricalSeries will identify which channels are providing LFP data. Filter properties should be noted in the ElectricalSeries description or comments field.

**Parameters**

- **source** (*str*) – the source of the data
- **electrical_series** (*ElectricalSeries*) – LFP electrophysiology data
- **name** (*str*) – the name of this container

**electrical_series**

**class** pynwb.ecephys.**FilteredEphys**(*source*, *electrical_series*, *name='FilteredEphys'*)

Bases: *pynwb.core.NWBContainer*

Ephys data from one or more channels that has been subjected to filtering. Examples of filtered data include Theta and Gamma (LFP has its own interface). FilteredEphys modules publish an ElectricalSeries for each filtered channel or set of channels. The name of each ElectricalSeries is arbitrary but should be informative. The source of the filtered data, whether this is from analysis of another time series or as acquired by hardware, should be noted in each's TimeSeries::description field. There is no assumed 1::1 correspondence between

filtered ephys signals and electrodes, as a single signal can apply to many nearby electrodes, and one electrode may have different filtered (e.g., theta and/or gamma) signals represented.

**Parameters**

- **source** (*str*) – the source of the data
- **electrical_series** (*ElectricalSeries*) – filtered electrophysiology data
- **name** (*str*) – the name of this container

**electrical_series**

**class** pynwb.ecephys.**FeatureExtraction**(*source*, *electrodes*, *description*, *times*, *features*, *name='FeatureExtraction'*)

Bases: *pynwb.core.NWBContainer*

Features, such as PC1 and PC2, that are extracted from signals stored in a SpikeEvent TimeSeries or other source.

**Parameters**

- **source** (*str*) – The source of the data
- **electrodes** (*ElectrodeTableRegion*) – the table region corresponding to the electrodes from which this series was recorded
- **description** (*list or tuple or ndarray or DataChunkIterator*) – A description for each feature extracted
- **times** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – The times of events that features correspond to
- **features** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO*) – Features for each channel
- **name** (*str*) – the name of this container

**description**

**electrode_group**

**features**

**times**

## 12.3 pynwb.icephys module

**class** pynwb.icephys.**IntracellularElectrode**(*name*, *source*, *slice*, *seal*, *description*, *location*, *resistance*, *filtering*, *initial_access_resistance*, *device*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **slice** (*str*) – Information about slice used for recording.
- **seal** (*str*) – Information about seal used for recording.
- **description** (*str*) – Recording description, description of electrode (e.g., whole-cell, sharp, etc) COMMENT: Free-form text (can be from Methods)

- **location** (*str*) – Area, layer, comments on estimation, stereotaxis coordinates (if in vivo, etc).
- **resistance** (*str*) – Electrode resistance COMMENT: unit: Ohm.
- **filtering** (*str*) – Electrode specific filtering.
- **initial_access_resistance** (*str*) – Initial access resistance.
- **device** (*str*) – Name(s) of devices in general/devices.

**slice**

**seal**

**description**

**location**

**resistance**

**filtering**

**initial_access_resistance**

**device**

**class** pynwb.icephys.**PatchClampSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Stores stimulus or response current or voltage. Superclass definition for patch-clamp data (this class should not be instantiated directly).

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz

- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**electrode**

**gain**

**class** pynwb.icephys.**CurrentClampSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *bias_current*, *bridge_balance*, *capacitance_compensation*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.icephys.PatchClampSeries*

Stores voltage data recorded from intracellular current-clamp recordings. A corresponding CurrentClampStimulusSeries (stored separately as a stimulus) is used to store the current injected.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **bias_current** (*float*) – Unit: Amp
- **bridge_balance** (*float*) – Unit: Ohm
- **capacitance_compensation** (*float*) – Unit: Farad
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **timestamps** – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset

- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**bias_current**

**bridge_balance**

**capacitance_compensation**

**class** pynwb.icephys.**IZeroClampSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *bias_current=0.0*, *bridge_balance=0.0*, *capacitance_compensation=0.0*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.icephys.CurrentClampSeries*

Stores recorded voltage data from intracellular recordings when all current and amplifier settings are off (i.e., CurrentClampSeries fields will be zero). There is no CurrentClampStimulusSeries associated with an IZero series because the amplifier is disconnected and no stimulus can reach the cell.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **bias_current** (*float*) – Unit: Amp
- **bridge_balance** (*float*) – Unit: Ohm
- **capacitance_compensation** (*float*) – Unit: Farad
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset

- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**class** pynwb.icephys.**CurrentClampStimulusSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.icephys.PatchClampSeries*

Aliases to standard PatchClampSeries. Its functionality is to better tag PatchClampSeries for machine (and human) readability of the file.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**class** pynwb.icephys.**VoltageClampSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *capacitance_fast*, *capacitance_slow*, *resistance_comp_bandwidth*, *resistance_comp_correction*, *resistance_comp_prediction*, *whole_cell_capacitance_comp*, *whole_cell_series_resistance_comp*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.icephys.PatchClampSeries*

Stores current data recorded from intracellular voltage-clamp recordings. A corresponding VoltageClampStimulusSeries (stored separately as a stimulus) is used to store the voltage injected.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **capacitance_fast** (*float*) – Unit: Farad
- **capacitance_slow** (*float*) – Unit: Farad
- **resistance_comp_bandwidth** (*float*) – Unit: Hz
- **resistance_comp_correction** (*float*) – Unit: %
- **resistance_comp_prediction** (*float*) – Unit: %
- **whole_cell_capacitance_comp** (*float*) – Unit: Farad
- **whole_cell_series_resistance_comp** (*float*) – Unit: Ohm
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data

- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**capacitance_fast**

**capacitance_slow**

**resistance_comp_bandwidth**

**resistance_comp_correction**

**resistance_comp_prediction**

**whole_cell_capacitance_comp**

**whole_cell_series_resistance_comp**

**class** pynwb.icephys.**VoltageClampStimulusSeries**(*name*, *source*, *data*, *unit*, *electrode*, *gain*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.icephys.PatchClampSeries*

Aliases to standard PatchClampSeries. Its functionality is to better tag PatchClampSeries for machine (and human) readability of the file.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **electrode** (*IntracellularElectrode*) – IntracellularElectrode group that describes the electrode that was used to apply or record this data.
- **gain** (*float*) – Units: Volt/Amp (v-clamp) or Volt/Volt (c-clamp)
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data

- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

## 12.4 pynwb.ophys module

**class** pynwb.ophys.**OpticalChannel**(*name*, *source*, *description*, *emission_lambda*, *parent=None*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **description** (*str*) – Any notes or comments about the channel.
- **emission_lambda** (*str*) – Emission lambda for channel.
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**description**

**emission_lambda**

**class** pynwb.ophys.**ImagingPlane**(*name*, *source*, *optical_channel*, *description*, *device*, *excitation_lambda*, *imaging_rate*, *indicator*, *location*, *manifold=None*, *conversion=None*, *unit=None*, *reference_frame=None*, *parent=None*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this electrode
- **source** (*str*) – the source of the data
- **optical_channel** (*list or OpticalChannel*) – One of possibly many groups storing channelspecific data.
- **description** (*str*) – Description of this ImagingPlane.
- **device** (*str*) – Name of device in /general/devices
- **excitation_lambda** (*str*) – Excitation wavelength.
- **imaging_rate** (*str*) – Rate images are acquired, in Hz.
- **indicator** (*str*) – Calcium indicator
- **location** (*str*) – Location of image plane.
- **manifold** (*Iterable*) – Physical position of each pixel. height, weight, x, y, z.
- **conversion** (*float*) – Multiplier to get from stored values to specified unit (e.g., 1e-3 for millimeters)
- **unit** (*str*) – Base unit that coordinates are stored in (e.g., Meters).
- **reference_frame** (*str*) – Describes position and reference frame of manifold based on position of first element in manifold.
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**optical_channel**

**description**

**device**

**excitation_lambda**

**imaging_rate**

**indicator**

**location**

**manifold**

**conversion**

**unit**

**reference_frame**

**class** pynwb.ophys.**TwoPhotonSeries**(*name, source, data, imaging_plane, unit=None, format=None, field_of_view=None, pmt_gain=None, scan_line_rate=None, external_file=None, starting_frame=None, bits_per_pixel=None, dimension=[nan], resolution=0.0, conversion=1.0, timestamps=None, starting_time=None, rate=None, comments='no comments', description='no description', control=None, control_description=None, parent=None*)

Bases: *pynwb.image.ImageSeries*

A special case of optical imaging.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **imaging_plane** (*ImagingPlane*) – Imaging plane class/pointer.
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **format** (*str*) – Format of image. Three types: 1) Image format; tiff, png, jpg, etc. 2) external 3) raw.
- **field_of_view** (*Iterable or TimeSeries*) – Width, height and depth of image, or imaged area (meters).
- **pmt_gain** (*float*) – Photomultiplier gain.
- **scan_line_rate** (*float*) – Lines imaged per second. This is also stored in /general/optophysiology but is kept here as it is useful information for analysis, and so good to be stored w/ the actual data.
- **external_file** (*Iterable*) – Path or URL to one or more external file(s). Field only present if format=external. Either external_file or data must be specified, but not both.
- **starting_frame** (*Iterable*) – Each entry is the frame number in the corresponding external_file variable. This serves as an index to what frames each file contains.
- **bits_per_pixel** (*int*) – Number of bit per image pixel

- **dimension** (*Iterable*) – Number of pixels on x, y, (and z) axes.
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**field_of_view**

**imaging_plane**

**pmt_gain**

**scan_line_rate**

**class** pynwb.ophys.**ROI**(*name*, *source*, *roi_description*, *pix_mask*, *pix_mask_weight*, *img_mask*)

Bases: *pynwb.core.NWBContainer*

A class for defining a region of interest (ROI)

**Parameters**

- **name** (*str*) – the name of this ROI
- **source** (*str*) – the source of the data
- **roi_description** (*str*) – Description of this ROI.
- **pix_mask** (*Iterable*) – List of pixels (x,y) that compose the mask.
- **pix_mask_weight** (*Iterable*) – Weight of each pixel listed in pix_mask.
- **img_mask** (*Iterable*) – ROI mask, represented in 2D ([y][x]) intensity image.

**roi_description**

**pix_mask**

**pix_mask_weight**

**img_mask**

**class** pynwb.ophys.**PlaneSegmentation**(*name*, *source*, *description*, *roi_list*, *imaging_plane*, *reference_images*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – name of PlaneSegmentation.
- **source** (*str*) – the source of the data

- **description** (*str*) – Description of image plane, recording wavelength, depth, etc.
- **roi_list** (*Iterable or ROI*) – List of ROIs in this imaging plane.
- **imaging_plane** (*ImagingPlane*) – link to ImagingPlane group from which this TimeSeries data was generated.
- **reference_images** (*ImageSeries*) – One or more image stacks that the masks apply to (can be oneelement stack).

**description**

**roi_list**

**imaging_plane**

**reference_images**

**class** pynwb.ophys.**ImageSegmentation**(*source*, *plane_segmentations*, *name='ImageSegmentation'*)

Bases: *pynwb.core.NWBContainer*

Stores pixels in an image that represent different regions of interest (ROIs) or masks. All segmentation for a given imaging plane is stored together, with storage for multiple imaging planes (masks) supported. Each ROI is stored in its own subgroup, with the ROI group containing both a 2D mask and a list of pixels that make up this mask. Segments can also be used for masking neuropil. If segmentation is allowed to change with time, a new imaging plane (or module) is required and ROI names should remain consistent between them.

**Parameters**

- **source** (*str*) – The source of the data represented in this Module Interface.
- **plane_segmentations** (*PlaneSegmentation or list*) – PlaneSegmentation with the description of the image plane.
- **name** (*str*) – the name of this ImageSegmentation container

**plane_segmentations**

**class** pynwb.ophys.**RoiResponseSeries**(*name*, *source*, *data*, *unit*, *roi_names*, *segmentation_interface*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

ROI responses over an imaging plane. Each row in data[] should correspond to the signal from one ROI.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **roi_names** (*Iterable*) – List of ROIs represented, one name for each row of data[].
- **segmentation_interface** (*ImageSegmentation*) – Link to ImageSegmentation.

- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**roi_names**

**segmentation_interface**

**class** pynwb.ophys.**DfOverF**(*source*, *roi_response_series*, *name='DfOverF'*)
Bases: *pynwb.core.NWBContainer*

dF/F information about a region of interest (ROI). Storage hierarchy of dF/F should be the same as for segmentation (ie, same names for ROIs and for image planes).

**Parameters**

- **source** (*str*) – The source of the data represented in this Module Interface.
- **roi_response_series** (*RoiResponseSeries or list*) – RoiResponseSeries or any subtype.
- **name** (*str*) – the name of this DfOverF container

**roi_response_series**

**class** pynwb.ophys.**Fluorescence**(*source*, *roi_response_series*, *name='Fluorescence'*)
Bases: *pynwb.core.NWBContainer*

Fluorescence information about a region of interest (ROI). Storage hierarchy of fluorescence should be the same as for segmentation (ie, same names for ROIs and for image planes).

**Parameters**

- **source** (*str*) – the source of the data represented in this Module Interface
- **roi_response_series** (*RoiResponseSeries or list*) – RoiResponseSeries or any subtype.
- **name** (*str*) – the name of this Fluorescence container

**roi_response_series**

## 12.5 pynwb.ogen module

**class** pynwb.ogen.**OptogeneticStimulusSite**(*name*, *source*, *device*, *description*, *excitation_lambda*, *location*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – The name of this stimulus site
- **source** (*str*) – the source of the data
- **device** (*str*) – Name of device in /general/devices
- **description** (*str*) – Description of site.
- **excitation_lambda** (*str*) – Excitation wavelength.
- **location** (*str*) – Location of stimulation site.

**device**

**description**

**excitation_lambda**

**location**

**class** pynwb.ogen.**OptogeneticSeries**(*name*, *source*, *data*, *site*, *unit='Watt'*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Optogenetic stimulus. The data field is in unit of watts.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **site** (*OptogeneticStimulusSite*) – Name of site description in general/optogentics.
- **unit** (*str*) – Value is the string "Watt".
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz

- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**site**

## 12.6 pynwb.retinotopy module

**class** pynwb.retinotopy.**AImage**(*name*, *source*, *data*, *bits_per_pixel*, *dimension*, *format*, *field_of_view*, *focal_depth*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this axis map
- **source** (*str*) – the source of the data
- **data** (*Iterable*) – Data field.
- **bits_per_pixel** (*int*) – Number of bits used to represent each value. This is necessary to determine maximum (white) pixel value.
- **dimension** (*Iterable*) – Number of rows and columns in the image.
- **format** (*Iterable*) – Format of image. Right now only "raw" supported.
- **field_of_view** (*Iterable*) – Size of viewing area, in meters.
- **focal_depth** (*float*) – Focal depth offset, in meters.

**data**

**bits_per_pixel**

**dimension**

**field_of_view**

**focal_depth**

**format**

**class** pynwb.retinotopy.**AxisMap**(*name*, *source*, *data*, *field_of_view*, *unit*, *dimension*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **name** (*str*) – the name of this axis map
- **source** (*str*) – the source of the data
- **data** (*Iterable*) – data field.
- **field_of_view** (*Iterable*) – Size of viewing area, in meters.
- **unit** (*str*) – Unit that axis data is stored in (e.g., degrees)
- **dimension** (*Iterable*) – Number of rows and columns in the image

**data**

**field_of_view**

**unit**

**dimension**

**class** pynwb.retinotopy.**ImagingRetinotopy**(*source*, *sign_map*, *axis_1_phase_map*, *axis_1_power_map*, *axis_2_phase_map*, *axis_2_power_map*, *axis_descriptions*, *focal_depth_image*, *vasculature_image*, *name='ImagingRetinotopy'*)

Bases: *pynwb.core.NWBContainer*

Intrinsic signal optical imaging or widefield imaging for measuring retinotopy. Stores orthogonal maps (e.g., altitude/azimuth; radius/theta) of responses to specific stimuli and a combined polarity map from which to identify visual areas. Note: for data consistency, all images and arrays are stored in the format [row][column] and [row, col], which equates to [y][x]. Field of view and dimension arrays may appear backward (i.e., y before x).

**Parameters**

- **source** (*str*) – The source of the data represented in this Module Interface.
- **sign_map** (*AxisMap*) – Sine of the angle between the direction of the gradient in axis_1 and axis_2.
- **axis_1_phase_map** (*AxisMap*) – Phase response to stimulus on the first measured axis.
- **axis_1_power_map** (*AxisMap*) – Power response on the first measured axis. Response is scaled so 0.0 is no power in the response and 1.0 is maximum relative power.
- **axis_2_phase_map** (*AxisMap*) – Phase response to stimulus on the second measured axis.
- **axis_2_power_map** (*AxisMap*) – Power response on the second measured axis. Response is scaled so 0.0 is no power in the response and 1.0 is maximum relative power.
- **axis_descriptions** (*Iterable*) – Two-element array describing the contents of the two response axis fields. Description should be something like ["altitude", "azimuth"] or ["radius", "theta"].
- **focal_depth_image** (*AImage*) – Gray-scale image taken with same settings/parameters (e.g., focal depth, wavelength) as data collection. Array format: [rows][columns].
- **vasculature_image** (*AImage*) – Gray-scale anatomical image of cortical surface. Array structure: [rows][columns].
- **name** (*str*) – the name of this container

**axis_1_phase_map**

**axis_1_power_map**

**axis_2_phase_map**

**axis_2_power_map**

**axis_descriptions**

**focal_depth_image**

**sign_map**

**vasculature_image**

## 12.7 pynwb.image module

**class** pynwb.image.**ImageSeries**(*name, source, data, unit='None', format='None', external_file=None, starting_frame=None, bits_per_pixel=None, dimension=[nan], resolution=0.0, conversion=1.0, timestamps=None, starting_time=None, rate=None, comments='no comments', description='no description', control=None, control_description=None, parent=None*)

Bases: *pynwb.base.TimeSeries*

General image data that is common between acquisition and stimulus time series. The image data can be stored in the HDF5 file or it will be stored as an external image file.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **format** (*str*) – Format of image. Three types: 1) Image format; tiff, png, jpg, etc. 2) external 3) raw.
- **external_file** (*Iterable*) – Path or URL to one or more external file(s). Field only present if format=external. Either external_file or data must be specified, but not both.
- **starting_frame** (*Iterable*) – Each entry is the frame number in the corresponding external_file variable. This serves as an index to what frames each file contains.
- **bits_per_pixel** (*int*) – Number of bit per image pixel
- **dimension** (*Iterable*) – Number of pixels on x, y, (and z) axes.
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**bits_per_pixel**

**dimension**

**external_file**

**starting_frame**

**format**

**class** pynwb.image.**IndexSeries**(*name*, *source*, *data*, *unit*, *indexed_timeseries*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Stores indices to image frames stored in an ImageSeries. The purpose of the ImageIndexSeries is to allow a static image stack to be stored somewhere, and the images in the stack to be referenced out-of-order. This can be for the display of individual images, or of movie segments (as a movie is simply a series of images). The data field stores the index of the frame in the referenced ImageSeries, and the timestamps array indicates when that image was displayed.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **indexed_timeseries** (*TimeSeries*) – HDF5 link to TimeSeries containing images that are indexed.
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**indexed_timeseries**

**class** pynwb.image.**ImageMaskSeries**(*name, source, data, unit, masked_imageseries, format, external_file=None, starting_frame=None, bits_per_pixel=None, dimension=[nan], resolution=0.0, conversion=1.0, timestamps=None, starting_time=None, rate=None, comments='no comments', description='no description', control=None, control_description=None, parent=None*)

Bases: *pynwb.image.ImageSeries*

An alpha mask that is applied to a presented visual stimulus. The data[] array contains an array of mask values that are applied to the displayed image. Mask values are stored as RGBA. Mask can vary with time. The timestamps array indicates the starting time of a mask, and that mask pattern continues until it's explicitly changed.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **masked_imageseries** (*ImageSeries*) – Link to ImageSeries that mask is applied to.
- **format** (*str*) – Format of image. Three types: 1) Image format; tiff, png, jpg, etc. 2) external 3) raw.
- **external_file** (*Iterable*) – Path or URL to one or more external file(s). Field only present if format=external. Either external_file or data must be specified, but not both.
- **starting_frame** (*Iterable*) – Each entry is the frame number in the corresponding external_file variable. This serves as an index to what frames each file contains.
- **bits_per_pixel** (*int*) – Number of bit per image pixel
- **dimension** (*Iterable*) – Number of pixels on x, y, (and z) axes.
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**masked_imageseries**

**class** pynwb.image.**OpticalSeries**(*name, source, data, unit, format, distance, field_of_view, orientation, external_file=None, starting_frame=None, bits_per_pixel=None, dimension=[nan], resolution=0.0, conversion=1.0, timestamps=None, starting_time=None, rate=None, comments='no comments', description='no description', control=None, control_description=None, parent=None*)

Bases: *pynwb.image.ImageSeries*

Image data that is presented or recorded. A stimulus template movie will be stored only as an image. When the image is presented as stimulus, additional data is required, such as field of view (eg, how much of the visual field the image covers, or how what is the area of the target being imaged). If the OpticalSeries represents acquired imaging data, orientation is also important.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **format** (*str*) – Format of image. Three types: 1) Image format; tiff, png, jpg, etc. 2) external 3) raw.
- **distance** (*float*) – Distance from camera/monitor to target/eye.
- **field_of_view** (*list or ndarray or TimeSeries*) – Width, height and depth of image, or imaged area (meters).
- **orientation** (*str*) – Description of image relative to some reference frame (e.g., which way is up). Must also specify frame of reference.
- **external_file** (*Iterable*) – Path or URL to one or more external file(s). Field only present if format=external. Either external_file or data must be specified, but not both.
- **starting_frame** (*Iterable*) – Each entry is the frame number in the corresponding external_file variable. This serves as an index to what frames each file contains.
- **bits_per_pixel** (*int*) – Number of bit per image pixel
- **dimension** (*Iterable*) – Number of pixels on x, y, (and z) axes.
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element by to conver to volts
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset

- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**distance**

**field_of_view**

**orientation**

## 12.8 pynwb.behavior module

**class** pynwb.behavior.**SpatialSeries**(*name*, *source*, *data*, *reference_frame*, *conversion=1.0*, *resolution=0.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *parent=None*, *control=None*, *control_description=None*)

Bases: *pynwb.base.TimeSeries*

Direction, e.g., of gaze or travel, or position. The TimeSeries::data field is a 2D array storing position or direction relative to some reference frame. Array structure: [num measurements] [num dimensions]. Each SpatialSeries has a text dataset reference_frame that indicates the zero-position, or the zero-axes for direction. For example, if representing gaze direction, "straight-ahead" might be a specific pixel on the monitor, or some other point in space. For position data, the 0,0 point might be the top-left corner of an enclosure, as viewed from the tracking camera. The unit of data will indicate how to interpret SpatialSeries values.

Create a SpatialSeries TimeSeries dataset

**Parameters**

- **name** (*str*) – The name of this SpatialSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **reference_frame** (*str*) – description defining what the zero-position is
- **conversion** (*float*) – Scalar to multiply each element by to conver to meters
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value

**reference_frame**

**class** pynwb.behavior.**BehavioralEpochs**(*source*, *interval_series*, *name='BehavioralEpochs'*)
Bases: *pynwb.core.NWBContainer*

TimeSeries for storing behavoioral epochs. The objective of this and the other two Behavioral interfaces (e.g. BehavioralEvents and BehavioralTimeSeries) is to provide generic hooks for software tools/scripts. This allows a tool/script to take the output one specific interface (e.g., UnitTimes) and plot that data relative to another data modality (e.g., behavioral events) without having to define all possible modalities in advance. Declaring one of these interfaces means that one or more TimeSeries of the specified type is published. These TimeSeries should reside in a group having the same name as the interface. For example, if a BehavioralTimeSeries interface is declared, the module will have one or more TimeSeries defined in the module sub-group "BehavioralTimeSeries". BehavioralEpochs should use IntervalSeries. BehavioralEvents is used for irregular events. BehavioralTimeSeries is for continuous data.

**Parameters**

- **source** (*str*) – The source of the data represented in this container.
- **interval_series** (*list or IntervalSeries*) – IntervalSeries or any subtype.
- **name** (*str*) – The name of this BehavioralEpochs container

**interval_series**

**class** pynwb.behavior.**BehavioralEvents**(*source*, *time_series*, *name='BehavioralEvents'*)
Bases: *pynwb.core.NWBContainer*

TimeSeries for storing behavioral events. See description of BehavioralEpochs for more details.

**Parameters**

- **source** (*str*) – the source of the data
- **time_series** (*TimeSeries*) – TimeSeries or any subtype.
- **name** (*str*) – The name of this BehavioralEvents container

**time_series**

**class** pynwb.behavior.**BehavioralTimeSeries**(*source*, *time_series*, *name='BehavioralTimeSeries'*)
Bases: *pynwb.core.NWBContainer*

TimeSeries for storing Behavoioral time series data. See description of BehavioralEpochs for more details.

**Parameters**

- **source** (*str*) – the source of the data
- **time_series** (*TimeSeries*) – <TimeSeries> or any subtype.
- **name** (*str*) – The name of this BehavioralTimeSeries

**time_series**

**class** pynwb.behavior.**PupilTracking**(*source*, *time_series*, *name='PupilTracking'*)
Bases: *pynwb.core.NWBContainer*

Eye-tracking data, representing pupil size.

**Parameters**

- **source** (*str*) – the source of the data

- **time_series** (*TimeSeries*) –
- **name** (*str*) – The name of this PupilTracking container

**time_series**

**class** pynwb.behavior.**EyeTracking**(*source*, *spatial_series*, *name='EyeTracking'*)
Bases: *pynwb.core.NWBContainer*

Eye-tracking data, representing direction of gaze.

**Parameters**

- **source** (*str*) – the source of the data
- **spatial_series** (*list or SpatialSeries*) –
- **name** (*str*) – The name of this EyeTracking container

**spatial_series**

**class** pynwb.behavior.**CompassDirection**(*source*, *spatial_series*, *name='CompassDirection'*)
Bases: *pynwb.core.NWBContainer*

With a CompassDirection interface, a module publishes a SpatialSeries object representing a floating point value for theta. The SpatialSeries::reference_frame field should indicate what direction corresponds to 0 and which is the direction of rotation (this should be clockwise). The si_unit for the SpatialSeries should be radians or degrees.

**Parameters**

- **source** (*str*) – the source of the data
- **spatial_series** (*list or SpatialSeries*) – SpatialSeries or any subtype.
- **name** (*str*) – The name of this CompassDirection container

**spatial_series**

**class** pynwb.behavior.**Position**(*source*, *spatial_series*, *name='Position'*)
Bases: *pynwb.core.NWBContainer*

Position data, whether along the x, x/y or x/y/z axis.

**Parameters**

- **source** (*str*) – the source of the data
- **spatial_series** (*list or SpatialSeries*) –
- **name** (*str*) – The name of this Position container

**spatial_series**

**class** pynwb.behavior.**CorrectedImageStack**(*source*, *corrected*, *original*, *xy_translation*, *name='CorrectedImageStack'*)
Bases: *pynwb.core.NWBContainer*

**Parameters**

- **source** (*str*) – the source of the data
- **corrected** (*ImageSeries*) – Image stack with frames shifted to the common coordinates.
- **original** (*ImageSeries*) – Link to image series that is being registered.
- **xy_translation** (*TimeSeries*) – Stores the x,y delta necessary to align each frame to the common coordinates, for example, to align each frame to a reference image.

- **name** (*str*) – The name of this CorrectedImageStack container

**corrected**

**original**

**xy_translation**

**class** pynwb.behavior.**MotionCorrection**(*source*, *corrected_image_stacks*, *name='MotionCorrection '*)

Bases: *pynwb.core.NWBContainer*

An image stack where all frames are shifted (registered) to a common coordinate system, to account for movement and drift between frames. Note: each frame at each point in time is assumed to be 2-D (has only x & y dimensions).

**Parameters**

- **source** (*str*) – the source of the data
- **corrected_image_stacks** (*list or CorrectedImageStack*) – the corrected image stack in this Motion Correction analysis
- **name** (*str*) – The name of this MotionCorrection container

**corrected_image_stacks**

## 12.9 pynwb.base module

**class** pynwb.base.**ProcessingModule**(*name*, *source*, *description*, *containers=None*, *parent=None*)

Bases: *pynwb.core.NWBContainer*

Processing module. This is a container for one or more containers that provide data at intermediate levels of analysis

ProcessingModules should be created through calls to NWB.create_module(). They should not be instantiated directly

**Parameters**

- **name** (*str*) – The name of this processing module
- **source** (*str*) – the source of the data
- **description** (*str*) – Description of this processing module
- **containers** (*list or dict*) – NWBContainers that belong to this ProcessingModule
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**description**

**containers**

**add_container**(*container*)

Add an NWBContainer to this ProcessingModule

**Parameters** **container** (*NWBContainer*) – the NWBContainer to add to this Module

**get_container**(*container_name*)

Retrieve an NWBContainer from this ProcessingModule

**Parameters** **container_name** (*str*) – the name of the NWBContainer to retrieve

**class** pynwb.base.**TimeSeries**(*name*, *source*, *data*, *unit*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.core.NWBContainer*

A generic base class for time series data

Create a TimeSeries object

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **unit** (*str*) – The base unit of measurement (should be SI unit)
- **resolution** (*str or float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*str or float*) – Scalar to multiply each element in data to convert it to the specified unit
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**timestamps_unit**

**interval**

**starting_time**

**rate**

**rate_unit**

**help**

**data**

**data_link**

**timestamps**

**ancestry**

**comments**

**control**

**control_description**

**conversion**

**description**

**neurodata_type**

**num_samples**

**resolution**

**timestamp_link**

**unit**

**time_unit**

## 12.10 pynwb.misc module

**class** pynwb.misc.**AnnotationSeries**(*name*, *source*, *data=[]*, *timestamps=None*, *comments='no comments'*, *description='no description'*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Stores text-based records about the experiment. To use the AnnotationSeries, add records individually through add_annotation() and then call finalize(). Alternatively, if all annotations are already stored in a list, use set_data() and set_timestamps()

All time series are created by calls to NWB.create_timeseries(). They should not not be instantiated directly

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**add_annotation**(*time*, *annotation*)

Add an annotation

**Parameters**

- **time** (*float*) – The time for the anotation
- **annotation** (*str*) – the annotation

**class** pynwb.misc.**AbstractFeatureSeries**(*name*, *source*, *feature_units*, *features*, *data=[]*, *resolution=0.0*, *conversion=1.0*, *timestamps=None*, *starting_time=None*, *rate=None*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Represents the salient features of a data stream. Typically this will be used for things like a visual grating stimulus, where the bulk of data (each frame sent to the graphics card) is bulky and not of high value, while the salient characteristics (eg, orientation, spatial frequency, contrast, etc) are what important and are what are used for analysis

All time series are created by calls to NWB.create_timeseries(). They should not not be instantiated directly

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **feature_units** (*str or Iterable*) – The unit of each feature
- **features** (*str or Iterable*) – Description of each feature
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – The data this TimeSeries dataset stores. Can also store binary data e.g. image frames
- **resolution** (*float*) – The smallest meaningful difference (in specified unit) between values in data
- **conversion** (*float*) – Scalar to multiply each element in data to convert it to the specified unit
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **starting_time** (*float*) – The timestamp of the first sample
- **rate** (*float*) – Sampling rate in Hz
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**features**

**feature_units**

**add_features**(*time*, *features*)

**Parameters**

- **time** (*float*) – the time point of this feature
- **features** (*list or ndarray*) – the feature values for this time point

**class** pynwb.misc.**IntervalSeries**(*name*, *source*, *data=[]*, *timestamps=[]*, *comments='no comments'*, *description='no description'*, *control=None*, *control_description=None*, *parent=None*)

Bases: *pynwb.base.TimeSeries*

Stores intervals of data. The timestamps field stores the beginning and end of intervals. The data field stores whether the interval just started (>0 value) or ended (<0 value). Different interval types can be represented in the same series by using multiple key values (eg, 1 for feature A, 2 for feature B, 3 for feature C, etc). The field data stores an 8-bit integer. This is largely an alias of a standard TimeSeries but that is identifiable as representing time intervals in a machinereadable way.

**Parameters**

- **name** (*str*) – The name of this TimeSeries dataset
- **source** (*str*) – Name of TimeSeries or Modules that serve as the source for the data contained here. It can also be the name of a device, for stimulus or acquisition data
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – >0 if interval started, <0 if interval ended.
- **timestamps** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Timestamps for samples stored in data
- **comments** (*str*) – Human-readable comments about this TimeSeries dataset
- **description** (*str*) – Description of this TimeSeries dataset
- **control** (*Iterable*) – Numerical labels that apply to each element in data
- **control_description** (*Iterable*) – Description of each control value
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**add_interval**(*start*, *stop*)

**Parameters**

- **start** (*float*) – The name of this TimeSeries dataset
- **stop** (*float*) – The name of this TimeSeries dataset

**data**

**timestamps**

**class** pynwb.misc.**SpikeUnit**(*name*, *times*, *unit_description*, *source*)

Bases: *pynwb.core.NWBContainer*

For use in the UnitTimes class.

**Parameters**

- **name** (*str*) – Name of the SpikeUnit
- **times** (*ndarray or list or tuple or Dataset or DataChunkIterator or DataIO or TimeSeries*) – Spike time for the units (exact or estimated)
- **unit_description** (*str*) – Description of the unit (eg, cell type).
- **source** (*str*) – Name, path or description of where unit times originated. This is necessary only if the info here differs from or is more fine-grained than the interfaces source field.

**times**

**unit_description**

**class** pynwb.misc.**UnitTimes**(*source*, *spike_units*, *name='UnitTimes'*)
Bases: *pynwb.core.NWBContainer*

Event times of observed units (e.g. cell, synapse, etc.). The UnitTimes group contains a group for each unit. The name of the group should match the value in the source module, if that is possible/relevant (e.g., name of ROIs from Segmentation module).

**Parameters**

- **source** (*str*) – the source of the data represented in this Module Interface
- **spike_units** (*Iterable*) – The SpikeUnits contained in this Interface
- **name** (*str*) – the name of this Container

**spike_units**

**unit_list**

## 12.11 pynwb.epoch module

**class** pynwb.epoch.**Epoch**(*name*, *source*, *start*, *stop*, *description=None*, *tags=[]*, *parent=None*)
Bases: *pynwb.core.NWBContainer*

Epoch object Epochs represent specific experimental intervals and store references to desired time series that overlap with the interval. The references to those time series indicate the first index in the time series that overlaps with the interval, and the duration of that overlap.

Epochs should be created through NWBFile.create_epoch(). They should not be instantiated directly

**Parameters**

- **name** (*str*) – the name of the epoch, as it will appear in the file
- **source** (*str*) – the source of the data
- **start** (*float*) – the starting time of the epoch
- **stop** (*float*) – the ending time of the epoch
- **description** (*str*) – a description of this epoch
- **tags** (*tuple or list*) – tags for this epoch
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**start_time**

**stop_time**

**description**

**tags**

**timeseries**

**get_timeseries**(*name*)

**Parameters** **name** (*str*) – The name of this TimeSeries dataset

**set_description**(*desc*)
Convenience function to set the value of the 'description' dataset in the epoch

**Parameters** ***desc*** (*text*) –

**Returns** *nothing*

**add_tag**(*tag*)

Append string annotation to epoch. This will be stored in the epoch's 'tags' dataset. Additionally, it will be added to a master tag list stored as an attribute on the root 'epoch/' group. Each epoch can have multiple tags. The root epoch keeps a list of unique tags

**Parameters** ***tag*** (*text*) –

**Returns** *nothing*

**add_ignore_interval**(*start*, *stop*)

Each epoch has a list of intervals that can be flagged to be ignored, for example due electrical noise or poor behavior of the subject. Intervals are trimmed to fit within epoch boundaries, but no further logic is performed (eg, if overlapping intervals are specified, those overlaps will be stored)

**Parameters**

- ***start*** (*float*) –
- ***stop*** (*float*) –

**Returns** *nothing*

**add_timeseries**(*timeseries*, *in_epoch_name=None*)

Associates time series with epoch. This will create a link to the specified time series within the epoch and will calculate its overlaps.

**Parameters**

- ***in_epoch_name*** (*text*) –
- **the epoch (this can be different than the actual** (*in*) –
- **series name)** (*time*) –
- ***timeseries*** (*text or TimeSeries object*) –
- **hdf5 path to time series that's being added,** (*Full*) –
- **the TimeSeries object itself** (*or*) –

**Returns** *nothing*

**class** pynwb.epoch.**EpochTimeSeries**(*source*, *ts*, *idx_start*, *count*, *name=None*, *parent=None*)

Bases: *pynwb.core.NWBContainer*

**Parameters**

- **source** (*str*) – the source of the data
- **ts** (*TimeSeries*) – the TimeSeries object
- **idx_start** (*int*) – the index of the start time in this TimeSeries
- **count** (*int*) – the number of samples available in the TimeSeries
- **name** (*str*) – the name of this alignment
- **parent** (*NWBContainer*) – The parent NWBContainer for this NWBContainer

**timeseries**

**count**

**idx_start**

## 12.12 pynwb package

### 12.12.1 Subpackages

#### pynwb.form package

#### Subpackages

#### pynwb.form.backends package

#### Subpackages

#### pynwb.form.backends.hdf5 package

#### Submodules

#### pynwb.form.backends.hdf5.h5_utils module

**class** pynwb.form.backends.hdf5.h5_utils.**H5DataIO**(*data*, *compress=False*)
Bases: *pynwb.form.data_utils.DataIO*

**Parameters**

- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator*) – the data to be written
- **compress** (*bool*) – Flag to use gzip compression filter on dataset

**compress**

**class** pynwb.form.backends.hdf5.h5_utils.**H5RegionSlicer**(*dataset*, *region*)
Bases: *pynwb.form.data_utils.RegionSlicer*

**Parameters**

- **dataset** (*Dataset*) – the HDF5 dataset to slice
- **region** (*RegionReference*) – the region reference to use to slice

#### pynwb.form.backends.hdf5.h5tools module

**class** pynwb.form.backends.hdf5.h5tools.**H5SpecReader**(*group*)
Bases: *pynwb.form.spec.namespace.SpecReader*

**Parameters** **group** (*Group*) – the HDF5 file to read specs from

**read_namespace**(*ns_path*)

**read_spec**(*spec_path*)

**class** pynwb.form.backends.hdf5.h5tools.**H5SpecWriter**(*group*)
Bases: *pynwb.form.spec.write.SpecWriter*

**Parameters** **group** (*Group*) – the HDF5 file to write specs to

**static stringify**(*spec*)
Converts a spec into a JSON string to write to a dataset

**write_namespace**(*namespace*, *path*)

**write_spec**(*spec*, *path*)

**class** pynwb.form.backends.hdf5.h5tools.**HDF5IO**(*path*, *manager=None*, *mode='a'*)
Bases: *pynwb.form.backends.io.FORMIO*

Open an HDF5 file for IO

For *mode*, see *Write an NWBFile*

**Parameters**

- **path** (*str*) – the path to the HDF5 file to write to
- **manager** (*BuildManager*) – the BuildManager to use for I/O
- **mode** (*str*) – the mode to open the HDF5 file with, one of ("w", "r", "r+", "a", "w-")

**close**()

**classmethod get_type**(*data*)

**classmethod isinstance_inmemory_array**(*data*)
Check if an object is a common in-memory data structure

**classmethod load_namespaces**(*namespace_catalog*, *path*, *namespaces=<class 'list'>*)
Load cached namespaces from a file.

**Parameters**

- **namespace_catalog** (*NamespaceCatalog*) – the NamespaceCatalog to load namespaces into
- **path** (*str*) – the path to the HDF5 file
- **namespaces** (*list*) – the namespaces to load

**open**()

**read_builder**()

**Returns** a GroupBuilder representing the NWB Dataset

**Return type** GroupBuilder

**classmethod set_attributes**(*obj*, *attributes*)

**Parameters**

- **obj** (*Group or Dataset*) – the HDF5 object to add attributes to
- **attributes** (*dict*) – a dict containing the attributes on the Group, indexed by attribute name

**write**(*container*, *cache_spec=False*)

**Parameters**

- **container** (*Container*) – the Container object to write
- **cache_spec** (*bool*) – cache specification to file

**write_builder**(*builder*)

**Parameters** **builder** (*GroupBuilder*) – the GroupBuilder object representing the NWBFile

**write_dataset**(*parent*, *builder*)
Write a dataset to HDF5

The function uses other dataset-dependent write functions, e.g, __scalar_fill__, __list_fill__ and __chunked_iter_fill__ to write the data.

**Parameters**

- **parent** (*Group*) – the parent HDF5 object
- **builder** (*DatasetBuilder*) – the DatasetBuilder to write

**Returns** the Dataset that was created

**Return type** Dataset

**write_group**(*parent*, *builder*)

**Parameters**

- **parent** (*Group*) – the parent HDF5 object
- **builder** (*GroupBuilder*) – the GroupBuilder to write

**Returns** the Group that was created

**Return type** Group

**write_link**(*parent*, *builder*)

**Parameters**

- **parent** (*Group*) – the parent HDF5 object
- **builder** (*LinkBuilder*) – the LinkBuilder to write

**Returns** the Link that was created

**Return type** Link

### Module contents

### Submodules

### pynwb.form.backends.io module

**class** pynwb.form.backends.io.**FORMIO**(*manager=None*, *source=None*)
Bases: object

**Parameters**

- **manager** (*BuildManager*) – the BuildManager to use for I/O
- **source** (*str*) – the source of container being built i.e. file path

**manager**
The BuildManager this FORMIO is using

**source**
The source of the container being read/written i.e. file path

**read**()

**Returns** the Container object that was read in

**Return type** Container

**write**(*container*)

**Parameters** **container** (*Container*) – the Container object to write

**read_builder**()
Read data and return the GroupBuilder represention

**Returns** a GroupBuilder representing the read data

**Return type** GroupBuilder

**write_builder**(*builder*)
Write a GroupBuilder representing an Container object

**Parameters** **builder** (*GroupBuilder*) – the GroupBuilder object representing the Container

**open**()
Open this FORMIO object for writing of the builder

**close**()
Close this FORMIO object to further reading/writing

### Module contents

### pynwb.form.build package

### Submodules

### pynwb.form.build.builders module

**class** pynwb.form.build.builders.**Builder**(*name*, *parent=None*, *source=None*)
Bases: dict

**Parameters**

- **name** (*str*) – the name of the group
- **parent** (*Builder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data in this builder e.g. file name

**name**
The name of this Builder

**source**
The source of this Builder

**parent**
The parent Builder of this Builder

**class** pynwb.form.build.builders.**BaseBuilder**(*name*, *attributes={}*, *parent=None*, *source=None*)
Bases: *pynwb.form.build.builders.Builder*

**Parameters**

- **name** (*str*) – the name of the group
- **attributes** (*dict*) – a dictionary of attributes to create in this group

- **parent** (*GroupBuilder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data represented in this Builder

**attributes**
The attributes stored in this Builder object

**set_attribute**(*name*, *value*)
Set an attribute for this group.

**Parameters**

- **name** (*str*) – the name of the attribute
- **value** (*None*) – the attribute value

**deep_update**(*builder*)
Merge attributes from the given BaseBuilder into this builder

**Parameters** **builder** (*BaseBuilder*) – the BaseBuilder to merge attributes from

**class** pynwb.form.build.builders.**GroupBuilder**(*name*, *groups={}*, *datasets={}*, *attributes={}*, *links={}*, *parent=None*, *source=None*)
Bases: *pynwb.form.build.builders.BaseBuilder*

Create a GroupBuilder object

**Parameters**

- **name** (*str*) – the name of the group
- **groups** (*dict or list*) – a dictionary of subgroups to create in this group
- **datasets** (*dict or list*) – a dictionary of datasets to create in this group
- **attributes** (*dict*) – a dictionary of attributes to create in this group
- **links** (*dict or list*) – a dictionary of links to create in this group
- **parent** (*GroupBuilder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data represented in this Builder

**source**
The source of this Builder

**groups**
The subgroups contained in this GroupBuilder

**datasets**
The datasets contained in this GroupBuilder

**links**
The datasets contained in this GroupBuilder

**set_attribute**(*name*, *value*)
Set an attribute for this group

**Parameters**

- **name** (*str*) – the name of the attribute
- **value** (*None*) – the attribute value

**add_dataset**(*name*, *data=None*, *dtype=None*, *attributes={}*, *maxshape=None*, *chunks=False*)
Create a dataset and add it to this group

**Parameters**

- **name** (*str*) – the name of this dataset
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or str or int or float or DataIO or DatasetBuilder or Iterable*) – a dictionary of datasets to create in this dataset
- **dtype** (*type or dtype or str or list*) – the datatype of this dataset
- **attributes** (*dict*) – a dictionary of attributes to create in this dataset
- **maxshape** (*int or tuple*) – the shape of this dataset. Use None for scalars
- **chunks** (*bool*) – whether or not to chunk this dataset

**Returns** the DatasetBuilder object for the dataset

**Return type** DatasetBuilder

**set_dataset**(*builder*)
Add a dataset to this group

**Parameters** **builder** (*DatasetBuilder*) – the DatasetBuilder that represents this dataset

**add_group**(*name*, *groups={}*, *datasets={}*, *attributes={}*, *links={}*)
Add a subgroup with the given data to this group

**Parameters**

- **name** (*str*) – the name of this subgroup
- **groups** (*dict*) – a dictionary of subgroups to create in this subgroup
- **datasets** (*dict*) – a dictionary of datasets to create in this subgroup
- **attributes** (*dict*) – a dictionary of attributes to create in this subgroup
- **links** (*dict*) – a dictionary of links to create in this subgroup

**Returns** the GroupBuilder object for the subgroup

**Return type** GroupBuilder

**set_group**(*builder*)
Add a subgroup to this group

**Parameters** **builder** (*GroupBuilder*) – the GroupBuilder that represents this subgroup

**add_link**(*name*, *target*)
Create a soft link and add it to this group

**Parameters**

- **name** (*str*) – the name of this link
- **target** (*GroupBuilder or DatasetBuilder*) – the target Builder

**Returns** the builder object for the soft link

**Return type** LinkBuilder

**set_link**(*builder*)
Add a link to this group

**Parameters** **builder** (*LinkBuilder*) – the LinkBuilder that represents this link

**deep_update**(*builder*)
Recursively update subgroups in this group

**is_empty**()
Returns true if there are no datasets, attributes, links or subgroups that contain datasets, attributes or links. False otherwise.

**get**(*key*, *default=None*)
Like dict.get, but looks in groups, datasets, attributes, and links sub-dictionaries.

**items**()
Like dict.items, but iterates over key-value pairs in groups, datasets, attributes, and links sub-dictionaries.

**keys**()
Like dict.keys, but iterates over keys in groups, datasets, attributes, and links sub-dictionaries.

**values**()
Like dict.values, but iterates over values in groups, datasets, attributes, and links sub-dictionaries.

**class** pynwb.form.build.builders.**DatasetBuilder**(*name*, *data=None*, *dtype=None*, *attributes={}*, *maxshape=None*, *chunks=False*, *parent=None*, *source=None*)
Bases: *pynwb.form.build.builders.BaseBuilder*

Create a Builder object for a dataset

**Parameters**

- **name** (*str*) – the name of the dataset
- **data** (*ndarray or list or tuple or Dataset or DataChunkIterator or str or int or float or DataIO or DatasetBuilder or Iterable*) – the data in this dataset
- **dtype** (*type or dtype or str or list*) – the datatype of this dataset
- **attributes** (*dict*) – a dictionary of attributes to create in this dataset
- **maxshape** (*int or tuple*) – the shape of this dataset. Use None for scalars
- **chunks** (*bool*) – whether or not to chunk this dataset
- **parent** (*GroupBuilder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data in this builder

**OBJECT_REF_TYPE = 'object'**

**REGION_REF_TYPE = 'region'**

**data**
The data stored in the dataset represented by this builder

**chunks**
Whether or not this dataset is chunked

**maxshape**
The max shape of this object

**dtype**
The data type of this object

**deep_update**(*dataset*)
Merge data and attributes from given DatasetBuilder into this DatasetBuilder

**Parameters** **dataset** (*DatasetBuilder*) – the DatasetBuilder to merge into this DatasetBuilder

**class** pynwb.form.build.builders.**LinkBuilder**(*name*, *builder*, *parent=None*, *source=None*)
Bases: *pynwb.form.build.builders.Builder*

**Parameters**

- **name** (*str*) – the name of the dataset
- **builder** (*DatasetBuilder or GroupBuilder*) – the target of this link
- **parent** (*GroupBuilder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data in this builder

**builder**
The target builder object

**class** pynwb.form.build.builders.**RegionBuilder**(*name*, *region*, *builder*, *attributes={}*, *parent=None*, *source=None*)
Bases: *pynwb.form.build.builders.DatasetBuilder*

**Parameters**

- **name** (*str*) – the name of the dataset
- **region** (*slice or tuple or list or RegionReference*) – the region i.e. slice or indices into the target Dataset
- **builder** (*DatasetBuilder*) – the Dataset this region applies to
- **attributes** (*dict*) – a dictionary of attributes to create in this dataset
- **parent** (*GroupBuilder*) – the parent builder of this Builder
- **source** (*str*) – the source of the data in this builder

**region**
The target builder object

### pynwb.form.build.map module

**class** pynwb.form.build.map.**BuildManager**(*type_map*)
Bases: object

A class for managing builds of Containers

**namespace_catalog**

**build**(*container*, *source=None*)
Build the GroupBuilder for the given Container

**Parameters**

- **container** (*Container*) – the container to convert to a Builder
- **source** (*str*) – the source of container being built i.e. file path

**prebuilt**(*container*, *builder*)
Save the Builder for a given Container for future use

**Parameters**

- **container** (*Container*) – the Container to save as prebuilt

- **builder** (*DatasetBuilder or GroupBuilder*) – the Builder representation of the given container

**construct**(*builder*)
Construct the Container represented by the given builder

**Parameters** **builder** (*DatasetBuilder or GroupBuilder*) – the builder to construct the Container from

**get_cls**(*builder*)
Get the class object for the given Builder

**Parameters** **builder** (*Builder*) – the Builder to get the class object for

**get_builder_name**(*container*)
Get the name a Builder should be given

**Parameters** **container** (*Container*) – the container to convert to a Builder

**Returns** The name a Builder should be given when building this container

**Return type** str

**get_subspec**(*spec*, *builder*)
Get the specification from this spec that corresponds to the given builder

**Parameters**

- **spec** (*DatasetSpec or GroupSpec*) – the parent spec to search
- **builder** (*DatasetBuilder or GroupBuilder or LinkBuilder*) – the builder to get the sub-specification for

**class** pynwb.form.build.map.**ObjectMapper**(*spec*)
Bases: object

A class for mapping between Spec objects and Container attributes

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**static constructor_arg**(*name*)
Decorator to override the default mapping scheme for a given constructor argument.

Decorate ObjectMapper methods with this function when extending ObjectMapper to override the default scheme for mapping between Container and Builder objects. The decorated method should accept as its first argument the Builder object that is being mapped. The method should return the value to be passed to the target Container class constructor argument given by *name*.

**Parameters** **name** (*str*) – the name of the constructor argument

**static object_attr**(*name*)
Decorator to override the default mapping scheme for a given object attribute.

Decorate ObjectMapper methods with this function when extending ObjectMapper to override the default scheme for mapping between Container and Builder objects. The decorated method should accept as its first argument the Container object that is being mapped. The method should return the child Builder object (or scalar if the object attribute corresponds to an AttributeSpec) that represents the attribute given by *name*.

**Parameters** **name** (*str*) – the name of the constructor argument

**hack_get_subspec_values**(**args*, ***kwargs*)

**spec**
the Spec used in this ObjectMapper

**get_container_name**(**args*)

**classmethod convert_dt_name**(*spec*)
Get the attribute name corresponding to a specification

**Parameters** **spec** (*Spec*) – the specification to get the name for

**classmethod get_attr_names**(*spec*)
Get the attribute names for each subspecification in a Spec

**Parameters** **spec** (*Spec*) – the specification to get the object attribute names for

**map_attr**(*attr_name*, *spec*)
Map an attribute to spec. Use this to override default behavior

**Parameters**

- **attr_name** (*str*) – the name of the object to map
- **spec** (*Spec*) – the spec to map the attribute to

**get_attr_spec**(*attr_name*)
Return the Spec for a given attribute

**Parameters** **attr_name** (*str*) – the name of the attribute

**get_carg_spec**(*carg_name*)
Return the Spec for a given constructor argument

**Parameters** **carg_name** (*str*) – the name of the constructor argument

**map_const_arg**(*const_arg*, *spec*)
Map an attribute to spec. Use this to override default behavior

**Parameters**

- **const_arg** (*str*) – the name of the constructor argument to map
- **spec** (*Spec*) – the spec to map the attribute to

**unmap**(*spec*)
Removing any mapping for a specification. Use this to override default mapping

**Parameters** **spec** (*Spec*) – the spec to map the attribute to

**map_spec**(*attr_carg*, *spec*)
Map the given specification to the construct argument and object attribute

**Parameters**

- **attr_carg** (*str*) – the constructor argument/object attribute to map this spec to
- **spec** (*Spec*) – the spec to map the attribute to

**get_attribute**(*spec*)
Get the object attribute name for the given Spec

**Parameters** **spec** (*Spec*) – the spec to get the attribute for

**Returns** the attribute name

**Return type** str

**get_attr_value**(*spec*, *container*)
Get the value of the attribute corresponding to this spec from the given container

**Parameters**

- **spec** (*Spec*) – the spec to get the attribute value for
- **container** (*Container*) – the container to get the attribute value from

**classmethod convert_dtype**(*dtype_spec*)

**get_const_arg**(*spec*)
Get the constructor argument for the given Spec

**Parameters** **spec** (*Spec*) – the spec to get the constructor argument for

**Returns** the name of the constructor argument

**Return type** str

**build**(*container*, *manager*, *parent=None*, *source=None*)
Convert an Container to a Builder representation

**Parameters**

- **container** (*Container*) – the container to convert to a Builder
- **manager** (*BuildManager*) – the BuildManager to use for managing this build
- **parent** (*Builder*) – the parent of the resulting Builder
- **source** (*str*) – the source of container being built i.e. file path

**Returns** the Builder representing the given Container

**Return type** Builder

**construct**(*builder*, *manager*)
Construct an Container from the given Builder

**Parameters**

- **builder** (*DatasetBuilder or GroupBuilder*) – the builder to construct the Container from
- **manager** (*BuildManager*) – the BuildManager for this build

**get_builder_name**(*container*)
Get the name of a Builder that represents a Container

**Parameters** **container** (*Container*) – the Container to get the Builder name for

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

**class** pynwb.form.build.map.**TypeSource**(*namespace*, *data_type*)
Bases: object

A class to indicate the source of a data_type in a namespace.

This class should only be used by TypeMap

**Parameters**

- **namespace** (*str*) – the namespace the from, which the data_type originated
- **data_type** (*str*) – the name of the type

**namespace**

**data_type**

**class** pynwb.form.build.map.**TypeMap**(*namespaces*, *mapper_cls=<class 'pynwb.form.build.map.ObjectMapper'>*)

Bases: object

A class to maintain the map between ObjectMappers and Container classes

**Parameters**

- **namespaces** (*NamespaceCatalog*) – the NamespaceCatalog to use
- **mapper_cls** (*type*) – the ObjectMapper class to use

**namespace_catalog**

**merge**(*type_map*)

**load_namespaces**(*namespace_path*, *resolve=True*)

Load namespaces from a namespace file.

This method will call load_namespaces on the NamespaceCatalog used to construct this TypeMap. Additionally, it will process the return value to keep track of what types were included in the loaded namespaces. Calling load_namespaces here has the advantage of being able to keep track of type dependencies across namespaces.

**Parameters**

- **namespace_path** (*str*) – the path to the file containing the namespaces(s) to load
- **resolve** (*bool*) – whether or not to include objects from included/parent spec objects

**Returns** the namespaces loaded from the given file

**Return type** tuple

**get_container_cls**(*namespace*, *data_type*)

Get the container class from data type specification

If no class has been associated with the data_type from namespace, a class will be dynamically created and returned.

**Parameters**

- **namespace** (*str*) – the namespace containing the data_type
- **data_type** (*str*) – the data type to create a Container class for

**Returns** the class for the given namespace and data_type

**Return type** type

**get_builder_dt**(*builder*)

**get_builder_ns**(*builder*)

**get_cls**(*builder*)

Get the class object for the given Builder

**Parameters** **builder** (*Builder*) – the Builder object to get the corresponding Container class for

**get_subspec**(*spec*, *builder*)

Get the specification from this spec that corresponds to the given builder

**Parameters**

- **spec** (*DatasetSpec or GroupSpec*) – the parent spec to search
- **builder** (*DatasetBuilder or GroupBuilder or LinkBuilder*) – the builder to get the sub-specification for

**get_container_classes**(*namespace=None*)

**Parameters** **namespace** (*str*) – the namespace to get the container classes for

**get_map**(*obj*)
Return the ObjectMapper object that should be used for the given container

**Parameters** **obj** (*Container or Builder*) – the object to get the ObjectMapper for

**Returns** the ObjectMapper to use for mapping the given object

**Return type** ObjectMapper

**register_container_type**(*namespace*, *data_type*, *container_cls*)
Map a container class to a data_type

**Parameters**

- **namespace** (*str*) – the namespace containing the data_type to map the class to
- **data_type** (*str*) – the data_type to mape the class to
- **container_cls** (*TypeSource or type*) – the class to map to the specified data_type

**register_map**(*container_cls*, *mapper_cls*)
Map a container class to an ObjectMapper class

**Parameters**

- **container_cls** (*type*) – the Container class for which the given ObjectMapper class gets used for
- **mapper_cls** (*type*) – the ObjectMapper class to use to map

**build**(*container*, *manager=None*, *source=None*)
Build the GroupBuilder for the given Container

**Parameters**

- **container** (*Container*) – the container to convert to a Builder
- **manager** (*BuildManager*) – the BuildManager to use for managing this build
- **source** (*str*) – the source of container being built i.e. file path

**construct**(*builder*, *build_manager=None*)
Construct the Container represented by the given builder

**Parameters**

- **builder** (*DatasetBuilder or GroupBuilder*) – the builder to construct the Container from
- **build_manager** (*BuildManager*) – the BuildManager for constructing

**get_builder_name**(*container*)
Get the name a Builder should be given

**Parameters** **container** (*Container*) – the container to convert to a Builder

**Returns** The name a Builder should be given when building this container

**Return type** str

### Module contents

### pynwb.form.spec package

### Submodules

### pynwb.form.spec.catalog module

**class** pynwb.form.spec.catalog.**SpecCatalog**

Bases: object

Create a new catalog for storing specifications

** Private Instance Variables **

**Variables**

- **__specs** – Dict with the specification of each registered type
- **__parent_types** – Dict with parent types for each registered type
- **__spec_source_files** – Dict with the path to the source files (if available) for each registered type
- **__hierarchy** – Dict describing the hierarchy for each registered type. NOTE: Always use SpecCatalog.get_hierarchy(...) to retrieve the hierarchy as this dictionary is used like a cache, i.e., to avoid repeated calcuation of the hierarchy but the contents are computed on first request by SpecCatalog.get_hierarchy(...)

**register_spec**(*spec*, *source_file=None*)

Associate a specified object type with an HDF5 specification

**Parameters**

- **spec** (*BaseStorageSpec*) – a Spec object
- **source_file** (*str*) – path to the source file from which the spec was loaded

**get_spec**(*data_type*)

Get the Spec object for the given type

**Parameters** **data_type** (*str*) – the data_type to get the Spec for

**Returns** the specification for writing the given object type to HDF5

**Return type** Spec

**get_registered_types**()

Return all registered specifications

**get_spec_source_file**(*data_type*)

**Return the path to the source file from which the spec for the given** type was loaded from. None is returned if no file path is available for the spec. Note: The spec in the file may not be identical to the object in case teh spec is modified after load.

**Parameters** **data_type** (*str*) – the data_type of the spec to get the source file for

**Returns** the path to source specification file from which the spec was orignially loaded or None

**Return type** str

**auto_register**(*spec*, *source_file=None*)
Register this specification and all sub-specification using data_type as object type name

**Parameters**

- **spec** (*BaseStorageSpec*) – the Spec object to register
- **source_file** (*str*) – path to the source file from which the spec was loaded

**Returns** the types that were registered with this spec

**Return type** tuple

**get_hierarchy**(*data_type*)
Get the extension hierarchy for the given data_type

**Parameters** **data_type** (*str or type*) – the data_type to get the hierarchy of

### pynwb.form.spec.namespace module

**class** pynwb.form.spec.namespace.**SpecNamespace**(*doc*, *name*, *schema*, *full_name=None*, *version=None*, *date=None*, *author=None*, *contact=None*, *catalog=None*)

Bases: dict

A namespace for specifications

**Parameters**

- **doc** (*str*) – a description about what this namespace represents
- **name** (*str*) – the name of this namespace
- **schema** (*list*) – location of schema specification files or other Namespaces
- **full_name** (*str*) – extended full name of this namespace
- **version** (*str or tuple or list*) – Version number of the namespace
- **date** (*datetime or str*) – Date last modified or released. Formatting is %Y-%m-%d %H:%M:%S, e.g, 2017-04-25 17:14:13
- **author** (*str or list*) – Author or list of authors.
- **contact** (*str or list*) – List of emails. Ordering should be the same as for author
- **catalog** (*SpecCatalog*) – The SpecCatalog object for this SpecNamespace

**classmethod types_key**()
Get the key used for specifying types to include from a file or namespace

Override this method to use a different name for 'data_types'

**full_name**
String with full name or None

**contact**
String or list of strings with the contacts or None

**author**
String or list of strings with the authors or None

**version**
String, list, or tuple with the version or None

**date**
Date last modified or released.

**Returns** datetime object, string, or None

**name**
String with short name or None

**doc**

**catalog**
The SpecCatalog containing all the Specs

**get_spec**(*data_type*)
Get the Spec object for the given data type

**Parameters** **data_type** (*str or type*) – the data_type to get the spec for

**get_registered_types**()
Get the available types in this namespace

**Returns** the a tuple of the available data types

**Return type** tuple

**get_hierarchy**(*data_type*)
Get the extension hierarchy for the given data_type in this namespace

**Parameters** **data_type** (*str or type*) – the data_type to get the hierarchy of

**Returns** a tuple with the type hierarchy

**Return type** tuple

**classmethod build_namespace**(***spec_dict*)

**class** pynwb.form.spec.namespace.**SpecReader**
Bases: object

**read_spec**()

**read_namespace**()

**class** pynwb.form.spec.namespace.**YAMLSpecReader**(*indir='.'*)
Bases: *pynwb.form.spec.namespace.SpecReader*

**Parameters** **indir** (*str*) – the path spec files are relative to

**read_namespace**(*namespace_path*)

**read_spec**(*spec_path*)

**class** pynwb.form.spec.namespace.**NamespaceCatalog**(*group_spec_cls=<class 'pynwb.form.spec.spec.GroupSpec'>, dataset_spec_cls=<class 'pynwb.form.spec.spec.DatasetSpec'>, spec_namespace_cls=<class 'pynwb.form.spec.namespace.SpecNamespace'>*)
Bases: object

Create a catalog for storing multiple Namespaces

**Parameters**

- **group_spec_cls** (*type*) – the class to use for group specifications
- **dataset_spec_cls** (*type*) – the class to use for dataset specifications
- **spec_namespace_cls** (*type*) – the class to use for specification namespaces

**namespaces**
The namespaces in this NamespaceCatalog

**Returns** a tuple of the availble namespaces

**Return type** tuple

**dataset_spec_cls**
The DatasetSpec class used in this NamespaceCatalog

**group_spec_cls**
The GroupSpec class used in this NamespaceCatalog

**spec_namespace_cls**
The SpecNamespace class used in this NamespaceCatalog

**add_namespace**(*name*, *namespace*)
Add a namespace to this catalog

**Parameters**

- **name** (*str*) – the name of this namespace
- **namespace** (*SpecNamespace*) – the SpecNamespace object

**get_namespace**(*name*)
Get the a SpecNamespace

**Parameters** **name** (*str*) – the name of this namespace

**Returns** the SpecNamespace with the given name

**Return type** SpecNamespace

**get_spec**(*namespace*, *data_type*)
Get the Spec object for the given type from the given Namespace

**Parameters**

- **namespace** (*str*) – the name of the namespace
- **data_type** (*str or type*) – the data_type to get the spec for

**Returns** the specification for writing the given object type to HDF5

**Return type** Spec

**get_hierarchy**(*namespace*, *data_type*)
Get the type hierarchy for a given data_type in a given namespace

**Parameters**

- **namespace** (*str*) – the name of the namespace
- **data_type** (*str or type*) – the data_type to get the spec for

**Returns** a tuple with the type hierarchy

**Return type** tuple

**get_sources**()
Get all the source specification files that were loaded in this catalog

**get_namespace_sources**(*namespace*)
Get all the source specifications that were loaded for a given namespace

**Parameters** **namespace** (*str*) – the name of the namespace

**get_types**(*source*)
Get the types that were loaded from a given source

**Parameters** **source** (*str*) – the name of the source

**load_namespaces**(*namespace_path*, *resolve=True*, *reader=None*)
Load the namespaces in the given file

**Parameters**

- **namespace_path** (*str*) – the path to the file containing the namespaces(s) to load
- **resolve** (*bool*) – whether or not to include objects from included/parent spec objects
- **reader** (*SpecReader*) – the class to user for reading specifications

**Returns** a dictionary describing the dependencies of loaded namespaces

**Return type** dict

### pynwb.form.spec.spec module

**class** pynwb.form.spec.spec.**ConstructableDict**
Bases: dict

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this ConstructableDict class from a dictionary

**classmethod build_spec**(*spec_dict*)
Build a Spec object from the given Spec dict

**class** pynwb.form.spec.spec.**Spec**(*doc*, *name=None*, *required=True*, *parent=None*)
Bases: *pynwb.form.spec.spec.ConstructableDict*

A base specification class

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – The name of this attribute
- **required** (*bool*) – whether or not this attribute is required
- **parent** (*Spec*) – the parent of this spec

**doc**
Documentation on what this Spec is specifying

**name**
The name of the object being specified

**parent**
The parent specification of this specification

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this Spec class from a dictionary

**class** pynwb.form.spec.spec.**AttributeSpec**(*name*, *doc*, *dtype*, *shape=None*, *dims=None*, *required=True*, *parent=None*, *value=None*, *default_value=None*)

Bases: *pynwb.form.spec.spec.Spec*

Specification for attributes

**Parameters**

- **name** (*str*) – The name of this attribute
- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str*) – The data type of this attribute
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **required** (*bool*) – whether or not this attribute is required. ignored when "value" is specified
- **parent** (*AttributeSpec*) – the parent of this spec
- **value** (*None*) – a constant value for this attribute
- **default_value** (*None*) – a default value for this attribute

**dtype**
The data type of the attribute

**value**
The constant value of the attribute. "None" if this attribute is not constant

**default_value**
The default value of the attribute. "None" if this attribute has no default value

**required**
True if this attribute is required, False otherwise.

**dims**
The dimensions of this attribute's value

**shape**
The shape of this attribute's value

**class** pynwb.form.spec.spec.**BaseStorageSpec**(*doc*, *name=None*, *default_name=None*, *attributes=[]*, *linkable=True*, *quantity=1*, *data_type_def=None*, *data_type_inc=None*)

Bases: *pynwb.form.spec.spec.Spec*

A specification for any object that can hold attributes.

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – the name of this base storage container
- **default_name** (*str*) – The default name of this base storage container
- **attributes** (*list*) – the attributes on this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **data_type_def** (*str*) – the NWB type this specification represents

- **data_type_inc** (*str or BaseStorageSpec*) – the NWB type this specification extends

**default_name**
The default name for this spec

**resolved**

**required**
Whether or not the this spec represents a required field

**resolve_spec**(*inc_spec*)

**Parameters** **inc_spec** (*BaseStorageSpec*) – the data type this specification represents

**is_inherited_spec**(*spec*)
Return True if this spec was inherited from the parent type, False otherwise

**Parameters** **spec** (*Spec or str*) – the specification to check

**is_overridden_spec**(*spec*)
Return True if this spec overrides a specification from the parent type, False otherwise

**Parameters** **spec** (*Spec or str*) – the specification to check

**is_inherited_attribute**(*name*)
Return True if the attribute was inherited from the parent type, False otherwise

**Parameters** **name** (*str*) – the name of the attribute to the Spec for

**is_overridden_attribute**(*name*)
Return True if the given attribute overrides the specification from the parent, False otherwise

**Parameters** **name** (*str*) – the name of the attribute to the Spec for

**is_many**()

**classmethod get_data_type_spec**(*data_type_def*)

**classmethod get_namespace_spec**()

**attributes**
The attributes for this specification

**linkable**
True if object can be a link, False otherwise

**classmethod type_key**()
Get the key used to store data type on an instance

Override this method to use a different name for 'data_type'

**classmethod inc_key**()
Get the key used to define a data_type include.

Override this method to use a different keyword for 'data_type_inc'

**classmethod def_key**()
Get the key used to define a data_type definition.

Override this method to use a different keyword for 'data_type_def'

**data_type_inc**
The data type of this specification

**data_type_def**
The data type this specification defines

**quantity**
The number of times the object being specified should be present

**add_attribute**(*name*, *doc*, *dtype*, *shape=None*, *dims=None*, *required=True*, *parent=None*, *value=None*, *default_value=None*)
Add an attribute to this specification

**Parameters**

- **name** (*str*) – The name of this attribute
- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str*) – The data type of this attribute
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **required** (*bool*) – whether or not this attribute is required. ignored when "value" is specified
- **parent** (*AttributeSpec*) – the parent of this spec
- **value** (*None*) – a constant value for this attribute
- **default_value** (*None*) – a default value for this attribute

**set_attribute**(*spec*)
Set an attribute on this specification

**Parameters** **spec** (*AttributeSpec*) – the specification for the attribute to add

**get_attribute**(*name*)
Get an attribute on this specification

**Parameters** **name** (*str*) – the name of the attribute to the Spec for

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this Spec class from a dictionary

**class** pynwb.form.spec.spec.**RefSpec**(*target_type*, *reftype*)
Bases: *pynwb.form.spec.spec.ConstructableDict*

**Parameters**

- **target_type** (*str*) – the target type GroupSpec or DatasetSpec
- **reftype** (*str*) – the type of references this is i.e. region or object

**target_type**
The data_type of the target of the reference

**reftype**
The type of reference

**is_region**()

**Returns** True if this RefSpec specifies a region reference, False otherwise

**Return type** bool

**class** pynwb.form.spec.spec.**DtypeSpec**(*name*, *doc*, *dtype*)
Bases: *pynwb.form.spec.spec.ConstructableDict*

A class for specifying a component of a compound type

**Parameters**

- **name** (*str*) – the name of this column
- **doc** (*str*) – a description about what this data type is
- **dtype** (*str or list or RefSpec*) – the data type of this column

**doc**
Documentation about this component

**name**
The name of this component

**dtype**
The data type of this component

**static assertValidDtype**(*dtype*)

**static is_ref**(*spec*)

**Parameters** **spec** (*str or dict*) – the spec object to check

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this Spec class from a dictionary

**class** pynwb.form.spec.spec.**DatasetSpec**(*doc*, *dtype=None*, *name=None*, *default_name=None*, *shape=None*, *dims=None*, *attributes=[]*, *linkable=True*, *quantity=1*, *default_value=None*, *data_type_def=None*, *data_type_inc=None*)

Bases: *pynwb.form.spec.spec.BaseStorageSpec*

Specification for datasets

To specify a table-like dataset i.e. a compound data type.

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str or list or RefSpec*) – The data type of this attribute. Use a list of DtypeSpecs to specify a compound data type.
- **name** (*str*) – The name of this dataset
- **default_name** (*str*) – The default name of this dataset
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **attributes** (*list*) – the attributes on this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **default_value** (*None*) – a default value for this dataset
- **data_type_def** (*str*) – the NWB type this specification represents
- **data_type_inc** (*str or DatasetSpec*) – the NWB type this specification extends

**resolve_spec**(*inc_spec*)

**Parameters** **inc_spec** (*DatasetSpec*) – the data type this specification represents

**dims**
The dimensions of this Dataset

**dtype**
The data type of the Dataset

**shape**
The shape of the dataset

**classmethod dtype_spec_cls**()
The class to use when constructing DtypeSpec objects

Override this if extending to use a class other than DtypeSpec to build dataset specifications

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this Spec class from a dictionary

**class** pynwb.form.spec.spec.**LinkSpec**(*doc*, *target_type*, *quantity=1*, *name=None*)
Bases: *pynwb.form.spec.spec.Spec*

**Parameters**

- **doc** (*str*) – a description about what this link represents
- **target_type** (*str*) – the target type GroupSpec or DatasetSpec
- **quantity** (*str or int*) – the required number of allowed instance
- **name** (*str*) – the name of this link

**target_type**
The data type of target specification

**data_type_inc**
The data type of target specification

**is_many**()

**quantity**
The number of times the object being specified should be present

**required**
Whether or not the this spec represents a required field

**class** pynwb.form.spec.spec.**GroupSpec**(*doc*, *name=None*, *default_name=None*, *groups=[]*, *datasets=[]*, *attributes=[]*, *links=[]*, *linkable=True*, *quantity=1*, *data_type_def=None*, *data_type_inc=None*)
Bases: *pynwb.form.spec.spec.BaseStorageSpec*

Specification for groups

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – the name of this group
- **default_name** (*str*) – The default name of this group
- **groups** (*list*) – the subgroups in this group
- **datasets** (*list*) – the datasets in this group
- **attributes** (*list*) – the attributes on this group
- **links** (*list*) – the links in this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance

- **data_type_def** (*str*) – the NWB type this specification represents
- **data_type_inc** (*str or GroupSpec*) – the NWB type this specification data_type_inc

**resolve_spec**(*inc_spec*)

**Parameters** **inc_spec** (*GroupSpec*) – the data type this specification represents

**is_inherited_dataset**(*name*)
Return true if a dataset with the given name was inherited

**Parameters** **name** (*str*) – the name of the dataset

**is_overridden_dataset**(*name*)
Return true if a dataset with the given name overrides a specification from the parent type

**Parameters** **name** (*str*) – the name of the dataset

**is_inherited_group**(*name*)
Return true if a group with the given name was inherited

**Parameters** **name** (*str*) – the name of the group

**is_overridden_group**(*name*)
Return true if a group with the given name overrides a specification from the parent type

**Parameters** **name** (*str*) – the name of the group

**is_inherited_link**(*name*)
Return true if a link with the given name was inherited

**Parameters** **name** (*str*) – the name of the link

**is_overridden_link**(*name*)
Return true if a link with the given name overrides a specification from the parent type

**Parameters** **name** (*str*) – the name of the link

**is_inherited_spec**(*spec*)
Returns 'True' if specification was inherited from a parent type

**Parameters** **spec** (*Spec or str*) – the specification to check

**is_overridden_spec**(*spec*)
Returns 'True' if specification was inherited from a parent type

**Parameters** **spec** (*Spec or str*) – the specification to check

**is_inherited_type**(*spec*)
Returns True if *spec* represents a spec that was inherited from an included data_type

**Parameters** **spec** (*BaseStorageSpec or str*) – the specification to check

**get_data_type**(*data_type*)
Get a specification by "data_type"

**Parameters** **data_type** (*str*) – the data_type to retrieve

**groups**
The groups specificed in this GroupSpec

**datasets**
The datasets specificed in this GroupSpec

**links**
The links specificed in this GroupSpec

**add_group**(*doc*, *name=None*, *default_name=None*, *groups=[]*, *datasets=[]*, *attributes=[]*, *links=[]*, *linkable=True*, *quantity=1*, *data_type_def=None*, *data_type_inc=None*)
Add a new specification for a subgroup to this group specification

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – the name of this group
- **default_name** (*str*) – The default name of this group
- **groups** (*list*) – the subgroups in this group
- **datasets** (*list*) – the datasets in this group
- **attributes** (*list*) – the attributes on this group
- **links** (*list*) – the links in this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **data_type_def** (*str*) – the NWB type this specification represents
- **data_type_inc** (*str or GroupSpec*) – the NWB type this specification data_type_inc

**set_group**(*spec*)
Add the given specification for a subgroup to this group specification

**Parameters** **spec** (*GroupSpec*) – the specification for the subgroup

**get_group**(*name*)
Get a specification for a subgroup to this group specification

**Parameters** **name** (*str*) – the name of the group to the Spec for

**add_dataset**(*doc*, *dtype=None*, *name=None*, *default_name=None*, *shape=None*, *dims=None*, *attributes=[]*, *linkable=True*, *quantity=1*, *default_value=None*, *data_type_def=None*, *data_type_inc=None*)
Add a new specification for a dataset to this group specification

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str or list or RefSpec*) – The data type of this attribute. Use a list of DtypeSpecs to specify a compound data type.
- **name** (*str*) – The name of this dataset
- **default_name** (*str*) – The default name of this dataset
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **attributes** (*list*) – the attributes on this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **default_value** (*None*) – a default value for this dataset
- **data_type_def** (*str*) – the NWB type this specification represents

- **data_type_inc** (*str or DatasetSpec*) – the NWB type this specification extends

**set_dataset**(*spec*)
Add the given specification for a dataset to this group specification

**Parameters** **spec** (*DatasetSpec*) – the specification for the dataset

**get_dataset**(*name*)
Get a specification for a dataset to this group specification

**Parameters** **name** (*str*) – the name of the dataset to the Spec for

**add_link**(*doc*, *target_type*, *quantity=1*, *name=None*)
Add a new specification for a link to this group specification

**Parameters**

- **doc** (*str*) – a description about what this link represents
- **target_type** (*str*) – the target type GroupSpec or DatasetSpec
- **quantity** (*str or int*) – the required number of allowed instance
- **name** (*str*) – the name of this link

**set_link**(*spec*)
Add a given specification for a link to this group specification

**Parameters** **spec** (*LinkSpec*) – the specification for the object to link to

**get_link**(*name*)
Get a specification for a link to this group specification

**Parameters** **name** (*str*) – the name of the link to the Spec for

**classmethod dataset_spec_cls**()
The class to use when constructing DatasetSpec objects

Override this if extending to use a class other than DatasetSpec to build dataset specifications

**classmethod link_spec_cls**()
The class to use when constructing LinkSpec objects

Override this if extending to use a class other than LinkSpec to build link specifications

**classmethod build_const_args**(*spec_dict*)
Build constructor arguments for this Spec class from a dictionary

### pynwb.form.spec.write module

**class** pynwb.form.spec.write.**SpecWriter**
Bases: object

**write_spec**(*spec_file_dict*, *path*)

**write_namespace**(*namespace*, *path*)

**class** pynwb.form.spec.write.**YAMLSpecWriter**(*outdir='.'*)
Bases: *pynwb.form.spec.write.SpecWriter*

**Parameters** **outdir** (*str*) – the path to write the directory to output the namespace and specs too

**write_spec**(*spec_file_dict*, *path*)

**write_namespace**(*namespace*, *path*)

**class** pynwb.form.spec.write.**NamespaceBuilder**(*doc*, *name*, *full_name=None*, *version=None*, *author=None*, *contact=None*, *namespace_cls=<class 'pynwb.form.spec.namespace.SpecNamespace'>*)

Bases: object

A class for building namespace and spec files

**Parameters**

- **doc** (*str*) – a description about what name namespace represents
- **name** (*str*) – the name of namespace
- **full_name** (*str*) – extended full name of name namespace
- **version** (*str or tuple or list*) – Version number of the namespace
- **author** (*str or list*) – Author or list of authors.
- **contact** (*str or list*) – List of emails. Ordering should be the same as for author
- **namespace_cls** (*type*) – the SpecNamespace type

**add_spec**(*source*, *spec*)
Add a Spec to the namespace

**Parameters**

- **source** (*str*) – the path to write the spec to
- **spec** (*GroupSpec or DatasetSpec*) – the Spec to add

**add_source**(*source*)
Add a source file to the namespace

**Parameters** **source** (*str*) – the path to write the spec to

**include_type**(*data_type*, *source=None*, *namespace=None*)
Include a data type from an existing namespace or source

**Parameters**

- **data_type** (*str*) – the data type to include
- **source** (*str*) – the source file to include the type from
- **namespace** (*str*) – the namespace from which to include the data type

**include_namespace**(*namespace*)
Include an entire namespace

**Parameters** **namespace** (*str*) – the namespace to include

**export**(*path*, *outdir='.'*, *writer=None*)
Export the namespace to the given path.

All new specification source files will be written in the same directory as the given path.

**Parameters**

- **path** (*str*) – the path to write the spec to
- **outdir** (*str*) – the path to write the directory to output the namespace and specs too
- **writer** (*SpecWriter*) – the SpecWriter to use to write the namespace

**class** pynwb.form.spec.write.**SpecFileBuilder**
Bases: dict

**add_spec**(*spec*)

**Parameters** **spec** (*GroupSpec or DatasetSpec*) – the Spec to add

### Module contents

### pynwb.form.validate package

### Submodules

### pynwb.form.validate.errors module

**class** pynwb.form.validate.errors.**Error**(*name*, *reason*)
Bases: object

**Parameters**

- **name** (*str*) – the name of the component that is erroneous
- **reason** (*str*) – the reason for the error

**name**

**reason**

**class** pynwb.form.validate.errors.**DtypeError**(*name*, *expected*, *received*)
Bases: *pynwb.form.validate.errors.Error*

**Parameters**

- **name** (*str*) – the name of the component that is erroneous
- **expected** (*type or str*) – the expected dtype
- **received** (*type or str*) – the received dtype

**class** pynwb.form.validate.errors.**MissingError**(*name*)
Bases: *pynwb.form.validate.errors.Error*

**Parameters** **name** (*str*) – the name of the component that is erroneous

**class** pynwb.form.validate.errors.**MissingDataType**(*name*, *data_type*)
Bases: *pynwb.form.validate.errors.Error*

**Parameters**

- **name** (*str*) – the name of the component that is erroneous
- **data_type** (*str*) – the missing data type

**data_type**

**class** pynwb.form.validate.errors.**ShapeError**(*name*, *expected*, *received*)
Bases: *pynwb.form.validate.errors.Error*

**Parameters**

- **name** (*str*) – the name of the component that is erroneous
- **expected** (*tuple or list*) – the expected shape

- **received** (*tuple or list*) – the received shape

### pynwb.form.validate.validator module

pynwb.form.validate.validator.**check_type**(*expected*, *received*)

pynwb.form.validate.validator.**check_shape**(*expected*, *received*)

**class** pynwb.form.validate.validator.**Validator**(*spec*)
Bases: object

A base class for classes that will be used to validate against Spec subclasses

**Parameters** **spec** (*Spec*) – the specification to use to validate

**spec**

**validate**(*value*)

**Parameters** **value** (*None*) – either in the form of a value or a Builder

**Returns** a list of Errors

**Return type** list

**classmethod get_spec_loc**(*spec*)

**classmethod get_builder_loc**(*builder*)

**class** pynwb.form.validate.validator.**AttributeValidator**(*spec*)
Bases: *pynwb.form.validate.validator.Validator*

A class for validating values against AttributeSpecs

**Parameters** **spec** (*AttributeSpec*) – the specification to use to validate

**validate**(*value*)

**Parameters** **value** (*None*) – the value to validate

**Returns** a list of Errors

**Return type** list

**class** pynwb.form.validate.validator.**BaseStorageValidator**(*spec*)
Bases: *pynwb.form.validate.validator.Validator*

A base class for validating against Spec objects that have attributes i.e. BaseStorageSpec

**Parameters** **spec** (*BaseStorageSpec*) – the specification to use to validate

**validate**(*builder*)

**Parameters** **builder** (*BaseBuilder*) – the builder to validate

**Returns** a list of Errors

**Return type** list

**class** pynwb.form.validate.validator.**ValidatorMap**(*namespace*)
Bases: object

A class for keeping track of Validator objects for all data types in a namespace

**Parameters** **namespace** (*SpecNamespace*) – the namespace to builder map for

**valid_types**(*spec*)
Get all valid types for a given data type

**Parameters** **spec** (*Spec or str*) – the specification to use to validate

**Returns** all valid sub data types for the given spec

**Return type** tuple

**get_validator**(*data_type*)
Return the validator for a given data type

**Parameters** **data_type** (*BaseStorageSpec or str*) – the data type to get the validator for

**Returns** the validator `data_type`

**Return type** BaseStorageValidator

**validate**(*builder*)
Validate a builder against a Spec

`builder` must have the attribute used to specifying data type by the namespace used to construct this ValidatorMap.

**Parameters** **builder** (*BaseBuilder*) – the builder to validate

**Returns** a list of errors found

**Return type** list

**class** pynwb.form.validate.validator.**DatasetValidator**(*spec*)
Bases: *pynwb.form.validate.validator.BaseStorageValidator*

A class for validating DatasetBuilders against DatasetSpecs

**Parameters** **spec** (*DatasetSpec*) – the specification to use to validate

**validate**(*builder*)

**Parameters** **builder** (*DatasetBuilder*) – the builder to validate

**Returns** a list of Errors

**Return type** list

**class** pynwb.form.validate.validator.**GroupValidator**(*spec*, *validator_map*)
Bases: *pynwb.form.validate.validator.BaseStorageValidator*

A class for validating GroupBuilders against GroupSpecs

**Parameters**

- **spec** (*GroupSpec*) – the specification to use to validate
- **validator_map** (*ValidatorMap*) – the ValidatorMap to use during validation

**validate**(*builder*)

**Parameters** **builder** (*GroupBuilder*) – the builder to validate

**Returns** a list of Errors

**Return type** list

### Module contents

### Submodules

### pynwb.form.container module

**class** pynwb.form.container.**Container**
Bases: object

**classmethod type_hierarchy**()

**class** pynwb.form.container.**Data**
Bases: *pynwb.form.container.Container*

**data**
The data that is held by this Container

**class** pynwb.form.container.**DataRegion**
Bases: *pynwb.form.container.Data*

**data**
The target data that this region applies to

**region**
The region that indexes into data e.g. slice or list of indices

### pynwb.form.data_utils module

pynwb.form.data_utils.**get_shape**(*data*)

pynwb.form.data_utils.**get_type**(*data*)

**class** pynwb.form.data_utils.**AbstractDataChunkIterator**
Bases: object

**recommended_chunk_shape**()

**recommended_data_shape**()

**class** pynwb.form.data_utils.**DataChunkIterator**(*data=None*, *max_shape=None*, *dtype=None*, *buffer_size=1*)
Bases: *pynwb.form.data_utils.AbstractDataChunkIterator*

Custom iterator class used to iterate over chunks of data.

Derived classes must ensure that self.shape and self.dtype are set properly. define the self.max_shape property describing the maximum shape of the array. In addition, derived classes must implement the __next__ method (or overwrite _read_next_chunk if the default behavior of __next__ should be reused). The __next__ method must return in each iteration 1) a numpy array with the data values for the chunk and 2) a numpy-compliant index tuple describing where the chunk is located within the complete data. HINT: numpy.s provides a convenient way to generate index tuples using standard array slicing. There are a number of additional functions that one can overwrite to customize behavior, e.g, the `recommended_chunk_shape` or `recommended_data_shape`

The default implementation accepts any iterable and assumes that we iterate over the first dimension of the data array. The default implemention supports buffered read, i.e., multiple values from the input iterator can be combined to a single chunk. This is useful for buffered I/O operations, e.g., to improve performance by accumulating data in memory and writing larger blocks at once.

Initalize the DataChunkIterator

**Parameters**

- **data** (*None*) – The data object used for iteration
- **max_shape** (*tuple*) – The maximum shape of the full data array. Use None to indicate unlimited dimensions
- **dtype** (*dtype*) – The Numpy data type for the array
- **buffer_size** (*int*) – Number of values to be buffered in a chunk

**next**()

Return the next data chunk or raise a StopIteration exception if all chunks have been retrieved.

HINT: numpy.s provides a convenient way to generate index tuples using standard array slicing. This is often useful to define the DataChunkk.selection of the current chunk

**Returns** DataChunk object with the data and selection of the current chunk

**Return type** DataChunk

**recommended_chunk_shape**()

Recommend a chunk shape.

To optimize iterative write the chunk should be aligned with the common shape of chunks returned by __next__ or if those chunks are too large, then a well-aligned subset of those chunks. This may also be any other value in case one wants to recommend chunk shapes to optimize read rather than write. The default implementation returns None, indicating no preferential chunking option.

**recommended_data_shape**()

**Recommend an initial shape of the data. This is useful when progressively writing data and** we want to recommend and initial size for the dataset

**class** pynwb.form.data_utils.**DataChunk**(*data=None*, *selection=None*)

Bases: object

Class used to describe a data chunk. Used in DataChunkIterator to describe

**Variables**

- **data** – Numpy ndarray with the data value(s) of the chunk
- **selection** – Numpy index tuple describing the location of the chunk

**Parameters**

- **data** (*ndarray*) – Numpy array with the data value(s) of the chunk
- **selection** (*None*) – Numpy index tuple describing the location of the chunk

**class** pynwb.form.data_utils.**ShapeValidator**

Bases: object

Helper class used to compare dimensions.

This class consists mainly of a set of static helper functions used to check that the shape of arrays is compliant.

The typical use of this class is by calling static methods that return instances of ShapeValidatorResult

**static get_data_shape**(*data*, *strict_no_data_load=False*)

Helper function used to determine the shape of the given array.

**Parameters**

- **data** (*List, numpy.ndarray, DataChunkIterator, any object that support __len__ or shape.*) – Array for which we should determine the shape.
- **strict_no_data_load** – In order to determin the shape of nested tuples and lists, this function recursively inspects elements along the dimensions, assuming that the data has a regular, rectangular shape. In the case of out-of-core iterators this means that the first item along each dimensions would potentially be loaded into memory. By setting this option we enforce that this does not happen, at the cost that we may not be able to determine the shape of the array.

**Returns** Tuple of ints indicating the size of known dimensions. Dimenions for which the size is unknown will be set to None.

**static assertEqualShape**(*data1*, *data2*, *axes1=None*, *axes2=None*, *name1=None*, *name2=None*, *ignore_undetermined=True*)

Ensure that the shape of data1 and data2 match along the given dimensions

**Parameters**

- **data1** (*List, Tuple, np.ndarray, DataChunkIterator etc.*) – The first input array
- **data2** (*List, Tuple, np.ndarray, DataChunkIterator etc.*) – The second input array
- **name1** – Optional string with the name of data1
- **name2** – Optional string with the name of data2
- **axes1** (*int, Tuple of ints, List of ints, or None*) – The dimensions of data1 that should be matched to the dimensions of data2. Set to None to compare all axes in order.
- **axes2** – The dimensions of data2 that should be matched to the dimensions of data1. Must have the same length as axes1. Set to None to compare all axes in order.
- **ignore_undetermined** – Boolean indicating whether non-matching unlimited dimensions should be ignored, i.e., if two dimension don't match because we can't determine the shape of either one, then should we ignore that case or treat it as no match

**Returns** Bool indicating whether the check passed and a string with a message about the matching process

**class** pynwb.form.data_utils.**ShapeValidatorResult**(*result=False*, *message=None*, *ignored=()*, *unmatched=()*, *error=None*, *shape1=()*, *shape2=()*, *axes1=()*, *axes2=()*)

Bases: object

Class for storing results from validating the shape of multi-dimensional arrays.

This class is used to store results generated by ShapeValidator

**Variables**

- **result** – Boolean indicating whether results matched or not
- **message** – Message indicating the result of the matching procedure

**Parameters**

- **result** (*bool*) – Result of the shape validation
- **message** (*str*) – Message describing the result of the shape validation

- **ignored** (*tuple*) – Axes that have been ignored in the validaton process
- **unmatched** (*tuple*) – List of axes that did not match during shape validation
- **error** (*str*) – Error that may have occurred. One of ERROR_TYPE
- **shape1** (*tuple*) – Shape of the first array for comparison
- **shape2** (*tuple*) – Shape of the second array for comparison
- **axes1** (*tuple*) – Axes for the first array that should match
- **axes2** (*tuple*) – Axes for the second array that should match

**SHAPE_ERROR = {None: 'All required axes matched', 'NUM_AXES_ERROR': 'Unequal number o**

Dict where the Keys are the type of errors that may have occured during shape comparison and the values are strings with default error messages for the type.

**class** pynwb.form.data_utils.**DataIO**(*data*)

Bases: object

**Parameters** **data** (*ndarray or list or tuple or Dataset or DataChunkIterator*) – the data to be written

**data**

**class** pynwb.form.data_utils.**RegionSlicer**

Bases: object

A class to control getting using a region

**class** pynwb.form.data_utils.**ListSlicer**(*dataset*, *region*)

Bases: *pynwb.form.data_utils.RegionSlicer*

**Parameters**

- **dataset** (*list or tuple*) – the HDF5 dataset to slice
- **region** (*list or tuple or slice*) – the region reference to use to slice

pynwb.form.data_utils.**get_region_slicer**(*dataset*, *region*)

**Parameters**

- **dataset** (*None*) – the HDF5 dataset to slice
- **region** (*None*) – the region reference to use to slice

### pynwb.form.monitor module

**exception** pynwb.form.monitor.**NotYetExhausted**

Bases: Exception

**class** pynwb.form.monitor.**DataChunkProcessor**(*data*)

Bases: *pynwb.form.data_utils.AbstractDataChunkIterator*

Initalize the DataChunkIterator

**Parameters** **data** (*DataChunkIterator*) – the DataChunkIterator to analyze

**recommended_chunk_shape**()

**recommended_data_shape**()

**get_final_result**(***kwargs*)

Return the result of processing data fed by this DataChunkIterator

**process_data_chunk**(*data_chunk*)

**This method should take in a DataChunk,** and process it.

**Parameters** **data_chunk** (*DataChunk*) – a chunk to process

**compute_final_result**()

**Return the result of processing this stream** Should raise NotYetExhaused exception

**class** pynwb.form.monitor.**NumSampleCounter**(***kwargs*)

Bases: *pynwb.form.monitor.DataChunkProcessor*

**process_data_chunk**(*data_chunk*)

**Parameters** **data_chunk** (*DataChunk*) – a chunk to process

**compute_final_result**()

### pynwb.form.utils module

pynwb.form.utils.**docval_macro**(*macro*)

pynwb.form.utils.**get_docval**(*func*)

Get a copy of docval arguments for a function

pynwb.form.utils.**fmt_docval_args**(*func*, *kwargs*)

Separate positional and keyword arguments

Useful for methods that wrap other methods

pynwb.form.utils.**call_docval_func**(*func*, *kwargs*)

pynwb.form.utils.**get_docval_args**(*func*)

Like get_docval, but return only positional arguments

pynwb.form.utils.**get_docval_kwargs**(*func*)

Like get_docval, but return only keyword arguments

pynwb.form.utils.**docval**(**validator*, ***options*)

A decorator for documenting and enforcing type for instance method arguments.

This decorator takes a list of dictionaries that specify the method parameters. These dictionaries are used for enforcing type and building a Sphinx docstring.

The first arguments are dictionaries that specify the positional arguments and keyword arguments of the decorated function. These dictionaries must contain the following keys: `'name'`, `'type'`, and `'doc'`. This will define a positional argument. To define a keyword argument, specify a default value using the key `'default'`. To validate the number of dimensions of an input array add the optional `'ndim'` parameter.

The decorated method must take `self` and `**kwargs` as arguments.

When using this decorator, the functions *getargs* and *popargs* can be used for easily extracting arguments from kwargs.

The following code example demonstrates the use of this decorator:

```
@docval({'name': 'arg1':,   'type': str,           'doc': 'this is the first␣
↪positional argument'},
        {'name': 'arg2':,   'type': int,           'doc': 'this is the second␣
↪positional argument'},
        {'name': 'kwarg1':, 'type': (list, tuple), 'doc': 'this is a keyword␣
↪argument', 'default': list()},
```

```
        returns='foo object', rtype='Foo'))
def foo(self, **kwargs):
    arg1, arg2, kwarg1 = getargs('arg1', 'arg2', 'kwarg1', **kwargs)
    ...
```

**Parameters**

- **enforce_type** – Enforce types of input parameters (Default=True)
- **returns** – String describing the return values
- **rtype** – String describing the data type of the return values
- **is_method** – True if this is decorating an instance or class method, False otherwise (Default=True)
- **enforce_ndim** – Enforce the number of dimensions of input arrays (Default=True)
- **validator** – `dict` objects specifying the method parameters
- **options** – additional options for documenting and validating method parameters

pynwb.form.utils.**getargs**(**argnames*, *argdict*)
Convenience function to retrieve arguments from a dictionary in batch

pynwb.form.utils.**popargs**(**argnames*, *argdict*)
Convenience function to retrieve and remove arguments from a dictionary in batch

**class** pynwb.form.utils.**ExtenderMeta**(*name*, *bases*, *classdict*)
Bases: abc.ABCMeta

A metaclass that will extend the base class initialization routine by executing additional functions defined in classes that use this metaclass

In general, this class should only be used by core developers.

**classmethod pre_init**(*func*)

**classmethod post_init**(*func*)
A decorator for defining a routine to run after creation of a type object.

An example use of this method would be to define a classmethod that gathers any defined methods or attributes after the base Python type construction (i.e. after `type` has been called)

**class** pynwb.form.utils.**frozendict**(*somedict*)
Bases: collections.abc.Mapping

An immutable dict

This will be useful for getter of dicts that we don't want to support

**get**(*key*, *default=None*)

**keys**()

**values**()

**items**()

### Module contents

### pynwb.io package

### Submodules

### pynwb.io.base module

**class** pynwb.io.base.**ModuleMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**name**(*builder*, *manager*)

**constructor_args = {'name': <function ModuleMap.name at 0x7f47d5a15378>}**

**obj_attrs = {}**

**class** pynwb.io.base.**TimeSeriesMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**name**(*builder*, *manager*)

**constructor_args = {'name': <function TimeSeriesMap.name at 0x7f47d5a15620>}**

**obj_attrs = {}**

### pynwb.io.core module

**class** pynwb.io.core.**NWBDataMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_name**(*builder*, *manager*)

**carg_data**(*builder*, *manager*)

**constructor_args = {'data': <function NWBDataMap.carg_data at 0x7f47d5a158c8>, 'name'**

**obj_attrs = {}**

**class** pynwb.io.core.**NWBTableRegionMap**(*spec*)
Bases: *pynwb.io.core.NWBDataMap*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_table**(*builder*, *manager*)

**carg_region**(*builder*, *manager*)

**constructor_args = {'data': <function NWBDataMap.carg_data at 0x7f47d5a158c8>, 'name'**

**obj_attrs = {}**

### pynwb.io.ecephys module

**class** pynwb.io.ecephys.**ElectrodeTableMap**(*spec*)
Bases: *pynwb.io.core.NWBDataMap*

**constructor_args = {'data': <function NWBDataMap.carg_data at 0x7f47d5a158c8>, 'name'**

**obj_attrs = {}**

**class** pynwb.io.ecephys.**ElectrodeTableRegionMap**(*spec*)
Bases: *pynwb.io.core.NWBTableRegionMap*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**constructor_args = {'data': <function NWBDataMap.carg_data at 0x7f47d5a158c8>, 'name'**

**obj_attrs = {}**

### pynwb.io.epoch module

**class** pynwb.io.epoch.**EpochMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**name**(*builder*, *manager*)

**constructor_args = {'name': <function EpochMap.name at 0x7f47d5999bf8>}**

**obj_attrs = {}**

**class** pynwb.io.epoch.**EpochTimeSeriesMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

### pynwb.io.file module

**class** pynwb.io.file.**NWBFileMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**name**(*builder*, *manager*)

**constructor_args = {'file_name': <function NWBFileMap.name at 0x7f47d5999598>, 'name'**

**obj_attrs = {}**

### pynwb.io.ophys module

**class** pynwb.io.ophys.**PlaneSegmentationMap**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

### Module contents

### pynwb.legacy package

### Subpackages

### pynwb.legacy.io package

### Submodules

### pynwb.legacy.io.base module

**class** pynwb.legacy.io.base.**ModuleMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**name**(**args*)

**carg_description**(**args*)

**constructor_args = {'description': <function ModuleMap.carg_description at 0x7f47d594**

**obj_attrs = {}**

**class** pynwb.legacy.io.base.**TimeSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**carg_name**(**args*)

**carg_starting_time**(**args*)

**carg_rate**(**args*)

**constructor_args = {'name': <function TimeSeriesMap.carg_name at 0x7f47d5945a60>, 'ra**

**obj_attrs = {}**

### pynwb.legacy.io.behavior module

**class** pynwb.legacy.io.behavior.**BehavioralTimeSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_time_series**(**args*)

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

**class** pynwb.legacy.io.behavior.**PupilTrackingMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_time_series**(**args*)

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

### pynwb.legacy.io.epoch module

**class** pynwb.legacy.io.epoch.**EpochMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**name**(*builder*)

**constructor_args = {'name': <function EpochMap.name at 0x7f47d583a2f0>, 'source': <f**

**obj_attrs = {}**

**class** pynwb.legacy.io.epoch.**EpochTimeSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

### pynwb.legacy.io.file module

**class** pynwb.legacy.io.file.**NWBFileMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**name**(*builder*)

**constructor_args = {'file_name': <function NWBFileMap.name at 0x7f47d583a7b8>, 'name'**

**obj_attrs = {}**

### pynwb.legacy.io.icephys module

**class** pynwb.legacy.io.icephys.**PatchClampSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_electrode**(**args*)

**constructor_args = {'electrode': <function PatchClampSeriesMap.carg_electrode at 0x7f**

**obj_attrs = {}**

### pynwb.legacy.io.image module

**class** pynwb.legacy.io.image.**ImageSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_data**(**args*)

**constructor_args = {'data': <function ImageSeriesMap.carg_data at 0x7f47d583abf8>, 'na**

**obj_attrs = {}**

### pynwb.legacy.io.misc module

**class** pynwb.legacy.io.misc.**AbstractFeatureSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_feature_units**(**args*)

**constructor_args = {'feature_units': <function AbstractFeatureSeriesMap.carg_feature_**

**obj_attrs = {}**

### pynwb.legacy.io.ogen module

**class** pynwb.legacy.io.ogen.**OptogeneticSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_site**(**args*)

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

### pynwb.legacy.io.ophys module

**class** pynwb.legacy.io.ophys.**PlaneSegmentationMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

**carg_imaging_plane**(**args*)

**constructor_args = {'imaging_plane': <function PlaneSegmentationMap.carg_imaging_plan**

**obj_attrs = {}**

**class** pynwb.legacy.io.ophys.**TwoPhotonSeriesMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_data**(**args*)

**carg_unit**(**args*)

**carg_imaging_plane**(**args*)

**constructor_args = {'data': <function TwoPhotonSeriesMap.carg_data at 0x7f47d583f730>**

**obj_attrs = {}**

**class** pynwb.legacy.io.ophys.**ROIMap**(*spec*)
Bases: *pynwb.legacy.map.ObjectMapperLegacy*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**carg_reference_images**(**args*)

**carg_name**(**args*)

**constructor_args = {'name': <function ROIMap.carg_name at 0x7f47d583fae8>, 'reference**

**obj_attrs = {}**

### Module contents

### Submodules

### pynwb.legacy.map module

pynwb.legacy.map.**decode**(*val*)

**class** pynwb.legacy.map.**ObjectMapperLegacy**(*spec*)
Bases: *pynwb.form.build.map.ObjectMapper*

Create a map from Container attributes to NWB specifications

**Parameters** **spec** (*DatasetSpec or GroupSpec*) – The specification for mapping objects to builders

**source_gettr**(*builder*, *manager*)

**constructor_args = {'name': <function ObjectMapper.get_container_name at 0x7f47d5b364**

**obj_attrs = {}**

**class** pynwb.legacy.map.**TypeMapLegacy**(*namespaces*, *mapper_cls=<class 'pynwb.form.build.map.ObjectMapper'>*)
Bases: *pynwb.form.build.map.TypeMap*

**Parameters**

- **namespaces** (*NamespaceCatalog*) – the NamespaceCatalog to use
- **mapper_cls** (*type*) – the ObjectMapper class to use

**get_builder_dt**(*builder*)
For a given builder, return the neurodata_type. In this legacy TypeMap, the builder may have out-of-spec neurodata_type; this function coerces this to a 2.0 compatible version.

**get_builder_ns**(*builder*)

### Module contents

pynwb.legacy.**get_type_map**(***kwargs*)
Get a TypeMap to use for I/O for Allen Institute Brain Observatory files (NWB v1.0.6)

pynwb.legacy.**register_map**(*container_cls*, *mapper_cls=None*)

**Register an ObjectMapper to use for a Container class type** If mapper_cls is not specified, returns a decorator for registering an ObjectMapper class as the mapper for container_cls. If mapper_cls specified, register the class as the mapper for container_cls

**Parameters**

- **container_cls** (*type*) – the Container class for which the given ObjectMapper class gets used for
- **mapper_cls** (*type*) – the ObjectMapper class to use to map

## 12.12.2 Submodules

### pynwb.core module

pynwb.core.**set_parents**(*container*, *parent*)

**class** pynwb.core.**LabelledDict**(*label*)
Bases: dict

A dict wrapper class for aggregating Timeseries from the standard locations

**Parameters** **label** (*str*) – the TimeSeries type (

**label**

**class** pynwb.core.**NWBBaseType**(*name*, *parent=None*, *container_source=None*)
Bases: object

The base class to any NWB types.

The purpose of this class is to provide a mechanism for representing hierarchical relationships in neurodata.

**Parameters**

- **name** (*str*) – the name of this container
- **parent** (*NWBContainer*) – the parent Container for this Container
- **container_source** (*object*) – the source of this Container e.g. file name

**name**

**container_source**
The source of this Container e.g. file name or table

**fields**

**parent**
The parent NWBContainer of this NWBContainer

**class** pynwb.core.**NWBContainer**(*source*, *name*, *parent=None*, *container_source=None*)
Bases: *pynwb.core.NWBBaseType*, *pynwb.form.container.Container*

**Parameters**

- **source** (*str*) – a description of where this NWBContainer came from

- **name** (*str*) – the name of this container
- **parent** (*NWBContainer*) – the parent Container for this Container
- **container_source** (*object*) – the source of this Container e.g. file name

**source**

**help**

**class** pynwb.core.**NWBData**(*name*, *data*, *parent=None*, *container_source=None*)
Bases: *pynwb.core.NWBBaseType*, *pynwb.form.container.Data*

**Parameters**

- **name** (*str*) – the name of this container
- **data** (*Iterable or Data*) – the source of the data
- **parent** (*NWBContainer*) – the parent Container for this Container
- **container_source** (*object*) – the source of this Container e.g. file name

**data**

**help**

**class** pynwb.core.**NWBTable**(*columns*, *name*, *data=[]*, *parent=None*, *container_source=None*)
Bases: *pynwb.core.NWBData*

**Parameters**

- **columns** (*list or tuple*) – a list of the columns in this table
- **name** (*str*) – the name of this container
- **data** (*Iterable*) – the source of the data
- **parent** (*NWBContainer*) – the parent Container for this Container
- **container_source** (*object*) – the source of this Container e.g. file name

**columns**

**add_row**(*values*)

**Parameters** **values** (*dict*) – the values for each column

**query**(*kwargs*)
Query a table

**Parameters** **kwargs** (*dict*) – the column to query by

**class** pynwb.core.**NWBTableRegion**(*name*, *table*, *region*)
Bases: *pynwb.core.NWBData*, *pynwb.form.container.DataRegion*

A class for representing regions i.e. slices or indices into an NWBTable

**Parameters**

- **name** (*str*) – the name of this container
- **table** (*NWBTable*) – the ElectrodeTable this region applies to
- **region** (*slice or list or tuple or RegionReference*) – the indices of the table

**table**
The ElectrodeTable this region applies to

**region**
The indices into table

### pynwb.spec module

**class** pynwb.spec.**NWBAttributeSpec**(*name*, *doc*, *dtype*, *shape=None*, *dims=None*, *required=True*, *parent=None*, *value=None*, *default_value=None*)
Bases: *pynwb.form.spec.spec.AttributeSpec*

**Parameters**

- **name** (*str*) – The name of this attribute
- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str*) – The data type of this attribute
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **required** (*bool*) – whether or not this attribute is required. ignored when "value" is specified
- **parent** (*AttributeSpec*) – the parent of this spec
- **value** (*None*) – a constant value for this attribute
- **default_value** (*None*) – a default value for this attribute

**class** pynwb.spec.**NWBLinkSpec**(*doc*, *target_type*, *quantity=1*, *name=None*)
Bases: *pynwb.form.spec.spec.LinkSpec*

**Parameters**

- **doc** (*str*) – a description about what this link represents
- **target_type** (*str*) – the target type GroupSpec or DatasetSpec
- **quantity** (*str or int*) – the required number of allowed instance
- **name** (*str*) – the name of this link

**neurodata_type_inc**
The neurodata type of target specification

**class** pynwb.spec.**BaseStorageOverride**
Bases: object

This class is used for the purpose of overriding BaseStorageSpec classmethods, without creating diamond inheritance hierarchies.

**classmethod type_key**()
Get the key used to store data type on an instance

Override this method to use a different name for 'data_type'

**classmethod inc_key**()
Get the key used to define a data_type include.

Override this method to use a different keyword for 'data_type_inc'

**classmethod def_key**()
Get the key used to define a data_type definition.

Override this method to use a different keyword for 'data_type_def'

**neurodata_type_inc**

**neurodata_type_def**

**class** pynwb.spec.**NWBDtypeSpec**(*name*, *doc*, *dtype*)
Bases: *pynwb.form.spec.spec.DtypeSpec*

**Parameters**

- **name** (*str*) – the name of this column
- **doc** (*str*) – a description about what this data type is
- **dtype** (*str or list or RefSpec*) – the data type of this column

**class** pynwb.spec.**NWBDatasetSpec**(*doc*, *dtype=None*, *name=None*, *default_name=None*, *shape=None*, *dims=None*, *attributes=[]*, *linkable=True*, *quantity=1*, *default_value=None*, *neurodata_type_def=None*, *neurodata_type_inc=None*)
Bases: *pynwb.spec.BaseStorageOverride*, *pynwb.form.spec.spec.DatasetSpec*

The Spec class to use for NWB specifications

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str or list or RefSpec*) – The data type of this attribute. Use a list of DtypeSpecs to specify a compound data type.
- **name** (*str*) – The name of this dataset
- **default_name** (*str*) – The default name of this dataset
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset
- **attributes** (*list*) – the attributes on this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **default_value** (*None*) – a default value for this dataset
- **neurodata_type_def** (*str*) – the NWB data type this spec defines
- **neurodata_type_inc** (*NWBDatasetSpec or str*) – the NWB data type this spec includes

**class** pynwb.spec.**NWBGroupSpec**(*doc*, *name=None*, *default_name=None*, *groups=[]*, *datasets=[]*, *attributes=[]*, *links=[]*, *linkable=True*, *quantity=1*, *neurodata_type_def=None*, *neurodata_type_inc=None*)
Bases: *pynwb.spec.BaseStorageOverride*, *pynwb.form.spec.spec.GroupSpec*

The Spec class to use for NWB specifications

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – the name of this group
- **default_name** (*str*) – The default name of this group
- **groups** (*list*) – the subgroups in this group
- **datasets** (*list*) – the datasets in this group

- **attributes** (*list*) – the attributes on this group
- **links** (*list*) – the links in this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **neurodata_type_def** (*str*) – the NWB data type this spec defines
- **neurodata_type_inc** (*NWBGroupSpec or str*) – the NWB data type this spec includes

**classmethod dataset_spec_cls**()

**get_neurodata_type**(*neurodata_type*)
Get a specification by "data_type"

**Parameters** **neurodata_type** (*str*) – the neurodata_type to retrieve

**add_group**(*doc*, *name=None*, *default_name=None*, *groups=[]*, *datasets=[]*, *attributes=[]*, *links=[]*, *linkable=True*, *quantity=1*, *neurodata_type_def=None*, *neurodata_type_inc=None*)
Add a new specification for a subgroup to this group specification

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **name** (*str*) – the name of this group
- **default_name** (*str*) – The default name of this group
- **groups** (*list*) – the subgroups in this group
- **datasets** (*list*) – the datasets in this group
- **attributes** (*list*) – the attributes on this group
- **links** (*list*) – the links in this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **neurodata_type_def** (*str*) – the NWB data type this spec defines
- **neurodata_type_inc** (*NWBGroupSpec or str*) – the NWB data type this spec includes

**add_dataset**(*doc*, *dtype=None*, *name=None*, *default_name=None*, *shape=None*, *dims=None*, *attributes=[]*, *linkable=True*, *quantity=1*, *default_value=None*, *neurodata_type_def=None*, *neurodata_type_inc=None*)
Add a new specification for a subgroup to this group specification

**Parameters**

- **doc** (*str*) – a description about what this specification represents
- **dtype** (*str or list or RefSpec*) – The data type of this attribute. Use a list of DtypeSpecs to specify a compound data type.
- **name** (*str*) – The name of this dataset
- **default_name** (*str*) – The default name of this dataset
- **shape** (*list or tuple*) – the shape of this dataset
- **dims** (*list or tuple*) – the dimensions of this dataset

- **attributes** (*list*) – the attributes on this group
- **linkable** (*bool*) – whether or not this group can be linked
- **quantity** (*str or int*) – the required number of allowed instance
- **default_value** (*None*) – a default value for this dataset
- **neurodata_type_def** (*str*) – the NWB data type this spec defines
- **neurodata_type_inc** (*NWBDatasetSpec or str*) – the NWB data type this spec includes

**class** pynwb.spec.**NWBNamespace**(*doc*, *name*, *schema*, *full_name=None*, *version=None*, *date=None*, *author=None*, *contact=None*, *catalog=None*)

Bases: *pynwb.form.spec.namespace.SpecNamespace*

A Namespace class for NWB

**Parameters**

- **doc** (*str*) – a description about what this namespace represents
- **name** (*str*) – the name of this namespace
- **schema** (*list*) – location of schema specification files or other Namespaces
- **full_name** (*str*) – extended full name of this namespace
- **version** (*str or tuple or list*) – Version number of the namespace
- **date** (*datetime or str*) – Date last modified or released. Formatting is %Y-%m-%d %H:%M:%S, e.g, 2017-04-25 17:14:13
- **author** (*str or list*) – Author or list of authors.
- **contact** (*str or list*) – List of emails. Ordering should be the same as for author
- **catalog** (*SpecCatalog*) – The SpecCatalog object for this SpecNamespace

**classmethod types_key**()

**class** pynwb.spec.**NWBNamespaceBuilder**(*doc*, *name*, *full_name=None*, *version=None*, *author=None*, *contact=None*)

Bases: *pynwb.form.spec.write.NamespaceBuilder*

A class for writing namespace and spec files for extensions of types in the NWB core namespace

Create a NWBNamespaceBuilder

**Parameters**

- **doc** (*str*) – a description about what this namespace represents
- **name** (*str*) – the name of this namespace
- **full_name** (*str*) – extended full name of this namespace
- **version** (*str or tuple or list*) – Version number of the namespace
- **author** (*str or list*) – Author or list of authors.
- **contact** (*str or list*) – List of emails. Ordering should be the same as for author

### pynwb.validate module

pynwb.validate.**main**()

### 12.12.3 Module contents

This ackage will contain functions, classes, and objects for reading and writing data in NWB format

pynwb.**get_type_map**()

pynwb.**get_manager**(*extensions=None*)

**Get a BuildManager to use for I/O using the given extensions. If no extensions are provided,** return a BuildManager that uses the core namespace

**Parameters** **extensions** (*str or TypeMap or list*) – a path to a namespace, a TypeMap, or a list consisting paths to namespaces and TypeMaps

**Returns** the namespaces loaded from the given file

**Return type** tuple

pynwb.**load_namespaces**(*namespace_path*)

Load namespaces from file

**Parameters** **namespace_path** (*str*) – the path to the YAML with the namespace definition

**Returns** the namespaces loaded from the given file

**Return type** tuple

pynwb.**available_namespaces**()

**Returns** a tuple of the available namespaces

**Return type** tuple

pynwb.**register_class**(*neurodata_type*, *namespace*, *container_cls=None*)

**Register an NWBContainer class to use for reading and writing a neurodata_type from a specification** If container_cls is not specified, returns a decorator for registering an NWBContainer subclass as the class for neurodata_type in namespace.

**Parameters**

- **neurodata_type** (*str*) – the neurodata_type to get the spec for
- **namespace** (*str*) – the name of the namespace
- **container_cls** (*type*) – the class to map to the specified neurodata_type

pynwb.**register_map**(*container_cls*, *mapper_cls=None*)

**Register an ObjectMapper to use for a Container class type** If mapper_cls is not specified, returns a decorator for registering an ObjectMapper class as the mapper for container_cls. If mapper_cls specified, register the class as the mapper for container_cls

**Parameters**

- **container_cls** (*type*) – the Container class for which the given ObjectMapper class gets used for
- **mapper_cls** (*type*) – the ObjectMapper class to use to map

pynwb.**get_class**(*neurodata_type*, *namespace*)

Get the class object of the NWBContainer subclass corresponding to a given neurdata_type.

**Parameters**

- **neurodata_type** (*str*) – the neurodata_type to get the NWBConatiner class for
- **namespace** (*str*) – the namespace the neurodata_type is defined in

pynwb.**validate**(*io*, *namespace='core'*)
Validate an NWB file against a namespace

**Parameters**

- **io** (*FORMIO*) – the FORMIO object to read from
- **namespace** (*str*) – the namespace to validate against

**Returns** errors in the file

**Return type** list

**class** pynwb.**NWBHDF5IO**(*path*, *manager=None*, *extensions=None*, *mode='a'*)
Bases: *pynwb.form.backends.hdf5.h5tools.HDF5IO*

**Parameters**

- **path** (*str*) – the path to the HDF5 file to write to
- **manager** (*BuildManager*) – the BuildManager to use for I/O
- **extensions** (*str or TypeMap or list*) – a path to a namespace, a TypeMap, or a list consisting paths to namespaces and TypeMaps
- **mode** (*str*) – the mode to open the HDF5 file with, one of ("w", "r", "r+", "a", "w-")

modindex

# CHAPTER 13

# Software Process

## 13.1 Continuous Integration

PyNWB is tested against Ubuntu, macOS and Windows operating systems. The project has both unit and integration tests.

- CircleCI runs all PyNWB tests on Ubuntu
- Appveyor runs all PyNWB tests on Windows
- Travis runs all PyNWB tests on macOS

Each time a PR is published or updated, the project is built, packaged and tested on all support operating systems and python distributions. That way, as a contributor you know if you introduced regressions or coding style inconsistencies.

There are badges in the README file which shows the current condition of the dev branch.

## 13.2 Coverage

Coverage is computed and reported using the coverage tool. There is a badge in the README file which shows percentage coverage. A detailed report can be found on codecov which shows line by line which lines are covered by the tests.

## 13.3 Requirement Specifications

There are 2 kinds of requirements specification in PyNWB.

### 13.3.1 Setup.py Dependencies

The first one is the dependencies in the *setup.py* file which lists the abstract dependencies for the PyNWB project. Note that there should not be specific versions of packages in the *setup.py* file.

### 13.3.2 Requirements.txt Dependencies

The second one is *requirements.txt* which contain a list of pinned (concrete) dependencies to reproduce an entire development environment to work with PyNWB.

In order to check the status of the required packages requires.io is used to create a badge on the project README. If all the required packages are up to date, a green badge appears.

If some of the packages are outdated, see *How to Update Requirements Files*.

## 13.4 Versioning and Releasing

PyNWB uses versioneer for versioning source and wheel distributions. Versioneer creates a semi-unique release name for the wheels that are created. It requires a version control system (git in PyNWB's case) to generate a release name. After all the tests pass, circleci creates both wheels(*.whl) and source distribution(*.tgz) for both Python 2 and Python 3 and uploads them back to github as a release. Versioneer makes it possible to get the source distribution from github and create wheels directly without having to use a version control system because it hardcodes versions in the source distribution.

CHAPTER 14

# How to Make a Release

A core developer should use the following steps to create a release *X.Y.Z* of **pynwb**.

**Note:** Since the pynwb wheels do not include compiled code, they are considered *pure* and could be generated on any supported platform.

That said, considering the instructions below have been tested on a Linux system, they may have to be adapted to work on macOS or Windows.

## 14.1 Prerequisites

- All CI tests are passing on all platforms.
- You have a GPG signing key.
- You have the API key associated with https://github.com/nwb-bot.

## 14.2 Documentation conventions

The commands reported below should be evaluated in the same terminal session.

Commands to evaluate starts with a dollar sign. For example:

```
$ echo "Hello"
Hello
```

means that `echo "Hello"` should be copied and evaluated in the terminal.

## 14.3 Setting up environment

1. First, register for an account on PyPI.
2. If not already the case, ask to be added as a `Package Index Maintainer`.
3. Create a `~/.pypirc` file with your login credentials:

```
[distutils]
index-servers =
  pypi
  pypitest

[pypi]
username=<your-username>
password=<your-password>

[pypitest]
repository=https://test.pypi.org/legacy/
username=<your-username>
password=<your-password>
```

   where `<your-username>` and `<your-password>` correspond to your PyPI account.

## 14.4 PyPI: Step-by-step

1. Choose the next release version number:

```
$ release=X.Y.Z
```

2. Download latest sources:

```
$ cd /tmp && git clone git@github.com:NeurodataWithoutBorders/pynwb && cd pynwb
```

3. Tag the release:

```
$ git tag -s -m "pynwb ${release}" ${release} origin/dev
```

   *Requires a GPG signing key*

4. Create a new virtual environment and install release requirements:

```
$ mkvirtualenv pynwb-${release}-release && \
  pip install tox twine
```

5. Create source distribution and wheels:

```
$ rm -rf dist/
$ tox
  [...]
  py35: commands succeeded
  py27: commands succeeded
  congratulations :)
```

**Warning:** Move forward **only** if the `tox` command completed successfully. If not, abort the release process and report the problem.

6. Confirm that expected packages have been generated:

```
$ ls -1 dist/*
pynwb-X.Y.Z-py2-none-any.whl
pynwb-X.Y.Z-py3-none-any.whl
pynwb-X.Y.Z.tar.gz
```

**Note:** `X.Y.Z` correspond to the release version selected earlier.

7. Sign and upload the packages to the PyPI testing server:

```
$ twine upload --sign -r pypitest dist/*
```

**Warning:** Confirm that the packages are available on both testing servers:

- new: https://test.pypi.org/project/pynwb/
- legacy: https://testpypi.python.org/pypi/pynwb.

8. Upload the packages to the production PyPI server:

```
$ twine upload --sign dist/*
```

**Warning:** Confirm that the packages are available on both servers:

- new: https://pypi.org/project/pynwb/
- legacy: https://pypi.python.org/pypi/pynwb

9. Create a clean testing environment to test installation:

```
$ mkvirtualenv pynwb-${release}-install-test && \
  pip install pynwb
```

10. Publish the release tag:

```
$ git push origin ${release}
```

11. Create GitHub release and upload packages:

```
$ pip install githubrelease
$ export GITHUB_TOKEN=<NWBOT_API_KEY>
$ githubrelease release NeurodataWithoutBorders/pynwb create ${release} --
↪name ${release} --publish ./dist/*
```

12. Cleanup

```
$ deactivate  && \
  rm -rf dist/* && \
```

```
  rmvirtualenv pynwb-${release}-release && \
  rmvirtualenv pynwb-${release}-install-test
```

## 14.5 Conda: Step-by-step

In order to release a new version on conda-forge, follow the steps below:

1. Clone feedstock

```
$ cd /tmp && \
  git clone git@github.com:conda-forge/pynwb-feedstock.git
```

2. Create a new branch

```
$ cd pynwb-feedstok && \
  git checkout -b $release
```

3. Modify meta.yaml

   Update the version string and sha256.

```
$ sed -i "2s/.*/{% set version = \"$release\" %}/" recipe/meta.yaml
$ sha=$(openssl sha256 ../pynwb/dist/*.tar.gz | awk '{print $2}')
$ sed -i "3s/.*/{$ set sha256 = \"$sha\" %}/" recipe/meta.yaml
```

4. Push the changes

```
$ git push origin $release
```

5. Create a Pull Request

   Create a pull request against the main repository. If the tests are passed a new release will be published on Anaconda cloud.

CHAPTER 15

# How to Update Requirements Files

The different requirements files introduced in *Software Process* section are the following:

- requirements.txt
- requirements-dev.txt
- requirements-doc.txt

## 15.1 requirements.txt

*Requirements.txt* of the project can be created or updated and then captured using the following script:

```
mkvirtualenv pynwb-requirements

cd pynwb
pip install .
pip freeze > requirements.txt

deactivate
rmvirtualenv pynwb-requirements
```

## 15.2 requirements-(dev|doc).txt

Any of these requirements files can be updated updated using the following scripts:

```
cd pynwb

# Set the requirements file to update
target_requirements=requirements-dev.txt

mkvirtualenv pynwb-requirements
```

```
# Install updated requirements
pip install -U -r $target_requirements

# If relevant, you could pip install new requirements now
# pip install -U <name-of-new-requirement>

# Update list of pinned requirements
pip freeze > $target_requirements

deactivate
rmvirtualenv pynwb-requirements
```

CHAPTER 16

# Copyright

CHAPTER 17

# License

CHAPTER 18

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## A

## B



## C

## D

## E



## F



## G

## H

## I



## K



## L

## M



## N



## O

## P

## Q



## R

## S

## T



## U



## V

## W



## X



## Y