



Hierarchical Data Modeling Framework

HDMF

Release 1.6.1

May 26, 2020

| | | |
|-----------|---|-----------|
| 1 | Dependencies | 3 |
| 2 | Installation | 5 |
| 2.1 | Install release from PyPI | 5 |
| 2.2 | Install release from conda-forge | 5 |
| 2.3 | Install latest pre-release | 6 |
| 3 | For developers | 7 |
| 3.1 | Install from Git repository | 7 |
| 3.2 | Run tests | 7 |
| 3.3 | Following the HDMF Style Guide | 8 |
| 4 | How to contribute to HDMF software and documents | 9 |
| 4.1 | Code of Conduct | 9 |
| 4.2 | Types of Contributions | 9 |
| 4.3 | Contributing Patches and Changes | 10 |
| 4.4 | Issue Labels, Projects, and Milestones | 11 |
| 4.5 | Styleguides | 11 |
| 4.6 | Endorsement | 11 |
| 5 | License and Copyright | 13 |
| 6 | Introduction | 15 |
| 7 | Software Architecture | 17 |
| 7.1 | Main Concepts | 20 |
| 7.2 | Additional Concepts | 22 |
| 8 | Tutorials | 25 |
| 9 | Extending standards | 27 |
| 9.1 | Creating new Extensions | 27 |
| 9.2 | Saving Extensions | 29 |
| 9.3 | Incorporating extensions | 30 |
| 9.4 | Documenting Extensions | 32 |
| 9.5 | Further Reading | 32 |
| 10 | Building API classes | 33 |

| | | |
|-----------|---|------------|
| 10.1 | The register_class function/decorator | 33 |
| 11 | Validating HDMF data | 35 |
| 12 | API Documentation | 37 |
| 12.1 | hdmf.common package | 37 |
| 12.2 | hdmf.container module | 44 |
| 12.3 | hdmf.build package | 45 |
| 12.4 | hdmf.spec package | 57 |
| 12.5 | hdmf.backends package | 73 |
| 12.6 | hdmf.data_utils module | 82 |
| 12.7 | hdmf.utils module | 85 |
| 12.8 | hdmf.validate package | 88 |
| 12.9 | hdmf.testing package | 92 |
| 12.10 | hdmf package | 93 |
| 13 | Software Process | 97 |
| 13.1 | Continuous Integration | 97 |
| 13.2 | Coverage | 97 |
| 13.3 | Requirement Specifications | 97 |
| 13.4 | Versioning and Releasing | 98 |
| 14 | How to Make a Roundtrip Test | 99 |
| 14.1 | H5RoundTripMixin | 99 |
| 15 | How to Make a Release | 101 |
| 15.1 | Prerequisites | 101 |
| 15.2 | Documentation conventions | 102 |
| 15.3 | Setting up environment | 102 |
| 15.4 | PyPI: Step-by-step | 102 |
| 15.5 | Conda: Step-by-step | 103 |
| 16 | How to Update Requirements Files | 105 |
| 16.1 | requirements.txt | 105 |
| 16.2 | requirements-(devldoc).txt | 105 |
| 16.3 | requirements-min.txt | 106 |
| 17 | Copyright | 107 |
| 18 | License | 109 |
| 19 | Indices and tables | 111 |
| | Python Module Index | 113 |
| | Index | 115 |

HDMF is a Python package for working with standardizing, reading, and writing hierarchical object data.

HDMF is a by product of the [Neurodata Without Borders: Neurophysiology \(NWB:N\)](#) project. The goal of NWB:N was to enable collaborative science within the neurophysiology and systems neuroscience communities through data standardization. The team of neuroscientists and software developers involved with NWB:N recognize that adoption of a unified data format is an important step toward breaking down the barriers to data sharing in neuroscience. HDMF was central the NWB:N development efforts, and has since been split off with the intention of providing it as an open-source tool for other scientific communities.

CHAPTER 1

Dependencies

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

1. Python 3.5, 3.6, 3.7, or 3.8
2. pip

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 the HDMF distribution from PyPI, simply run:

```
$ pip install -U hdmf
```

This will automatically install the following required dependencies:

1. h5py
2. numpy
3. scipy
4. pandas
5. ruamel.yaml

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 the HDMF distribution from conda-forge using conda, simply run:

```
$ conda install -c conda-forge hdmf
```

2.3 Install latest pre-release

To try out the latest features and also set up continuous integration of your own project against the latest version of HDMF, install the latest release from GitHub.

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

3.1 Install from Git repository

For development, an editable install in a virtual environment is recommended. First, create a new virtual environment located at `~/hdmf` using the `virtualenv` tool.

```
$ pip install -U virtualenv
$ virtualenv ~/hdmf
$ source ~/hdmf/bin/activate
```

Alternatively, you can use the `conda` environment and package manager to create your virtual environment. This may work better on Windows.

```
$ conda create --name hdmf-dev python=3.8
$ conda activate hdmf-dev
```

Then clone the git repository for HDMF, install the HDMF package requirements using the `pip` Python package manager, and install HDMF.

```
$ git clone --recurse-submodules git@github.com:hdmf-dev/hdmf.git
$ cd hdmf
$ pip install -r requirements.txt
$ pip install -e .
```

3.2 Run tests

For running the tests, it is required to install the development requirements. Within a virtual environment, run the following code, which will clone the git repository for HDMF, install the HDMF package requirements using `pip`, install HDMF, and run tests using the `tox` automated testing tool.

```
$ cd hdmf
$ pip install -r requirements.txt -r requirements-dev.txt
$ pip install -e .
$ tox
```

3.3 Following the HDMF Style Guide

Before you create a Pull Request, make sure you are following the HDMF style guide ([PEP8](#)). To check whether your code conforms to the HDMF style guide, make sure you have the development requirements installed (see above) and then simply run the [flake8](#) tool in the project's root directory.

```
$ flake8
```

How to contribute to HDMF 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?

- **Submit issues and requests** using our [issue tracker](#)
- **Ensure the feature or change was not already reported** by searching on GitHub under [HDMF Issues](#)
- If you are unable to find an open issue addressing the problem then open a new issue on the respective repository. Be sure to use our issue templates and 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 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.
- HDMF 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 HDMF?

See our hdmf-dev.github.io website for details.

4.2.5 Informal discussions between developers and users?

The <https://nwb-users.slack.com> slack is currently used for informal discussions between developers and users.

4.3 Contributing Patches and Changes

To contribute to HDMF 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. Be sure to use our [pull request template](#) and:
 - 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 respective 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:

- [HDMF 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:

- [HDMF milestones](#)

4.4.3 Projects

Projects are currently used mainly on the HDMF 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 reStructuredText (RST) using Sphinx.

4.5.3 Python Code Styleguide

Python coding style is checked via `flake8` for automatic checking of PEP8 style during pull requests.

4.6 Endorsement

Please don't take the fact that working with an organization (e.g., during a hackathon or via GitHub) as an endorsement of your work or your organization. It's okay to say e.g., “We worked with XXXXX to advance science” but not e.g., “XXXXX supports our work on HDMF”.

License and Copyright

See the [license](#) files for details about the copyright and license.

As indicated in the HDMF license: *“You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code (“Enhancements”) to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.”*

Contributors to the HDMF code base are expected to use a permissive, non-copyleft open source license. Typically 3-clause BSD is used, but any compatible license is allowed, the MIT and Apache 2.0 licenses being good alternative choices. The GPL and other copyleft licenses are not allowed due to the consternation it generates across many organizations.

Also, make sure that you are permitted to contribute code. Some organizations, even academic organizations, have agreements in place that discuss IP ownership in detail (i.e., address IP rights and ownership that you create while under the employ of the organization). These are typically signed documents that you looked at on your first day of work and then promptly forgot. We don’t want contributed code to be yanked later due to IP issues.

CHAPTER 6

Introduction

HDMF provides a high-level Python API for specifying, reading, writing and manipulating hierarchical object data. This section provides a broad overview of the software architecture of HDMF (see Section *Software Architecture*) and its functionality.

CHAPTER 7

Software Architecture

The main goal of HDMF is to enable users and developers to efficiently interact with the hierarchical object data. The following figures provide an overview of the high-level architecture of HDMF and functionality of the various components.

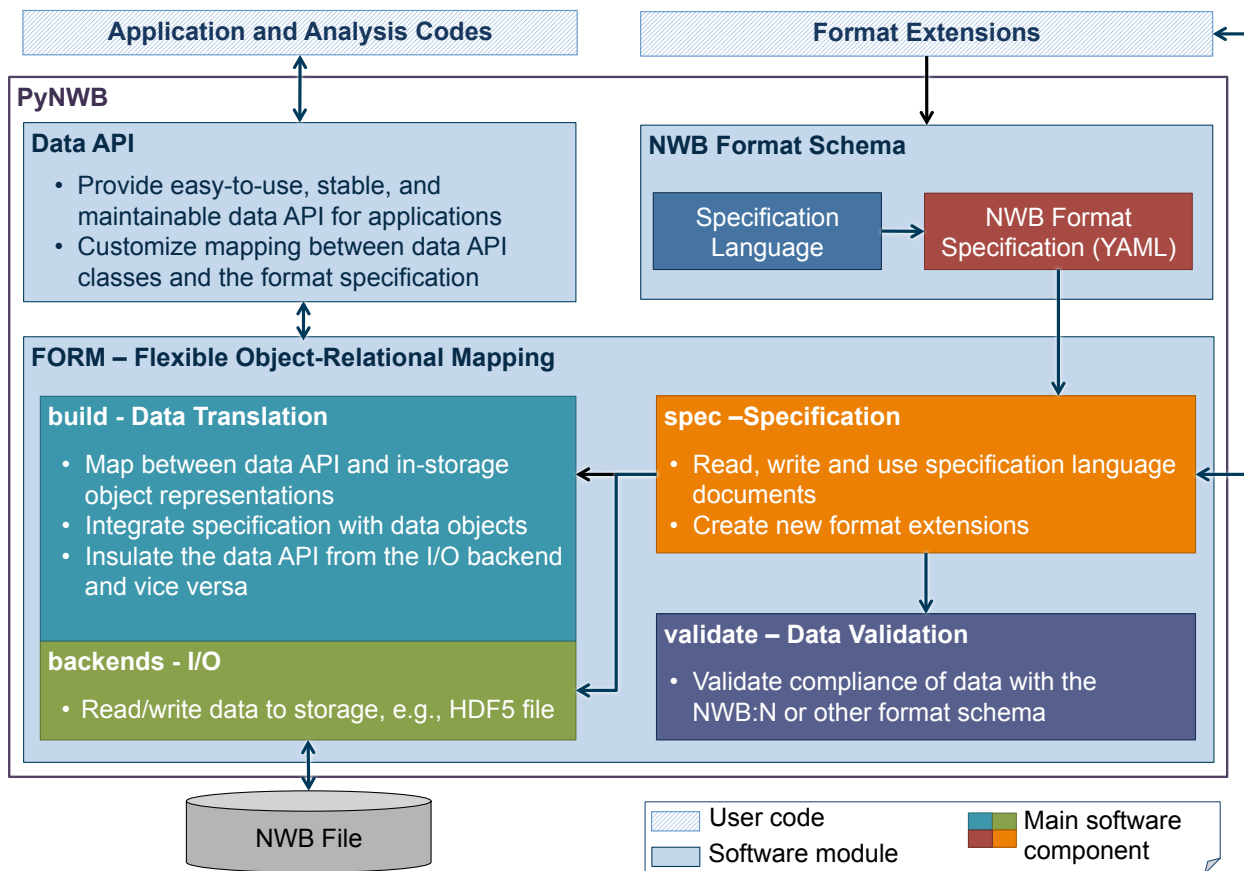


Fig. 1: Overview of the high-level software architecture of HDMF (click to enlarge).

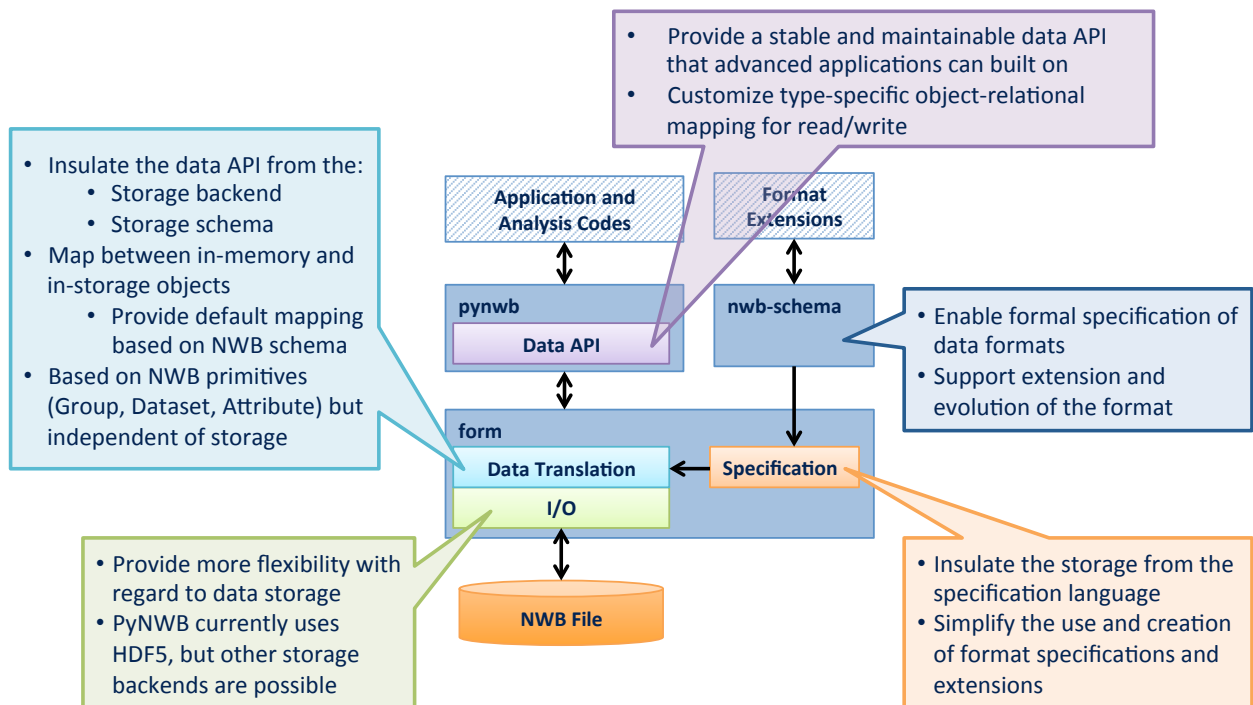


Fig. 2: We choose a modular design for HDMF to enable flexibility and separate the various levels of standardizing hierarchical data (click to enlarge).

7.1 Main Concepts

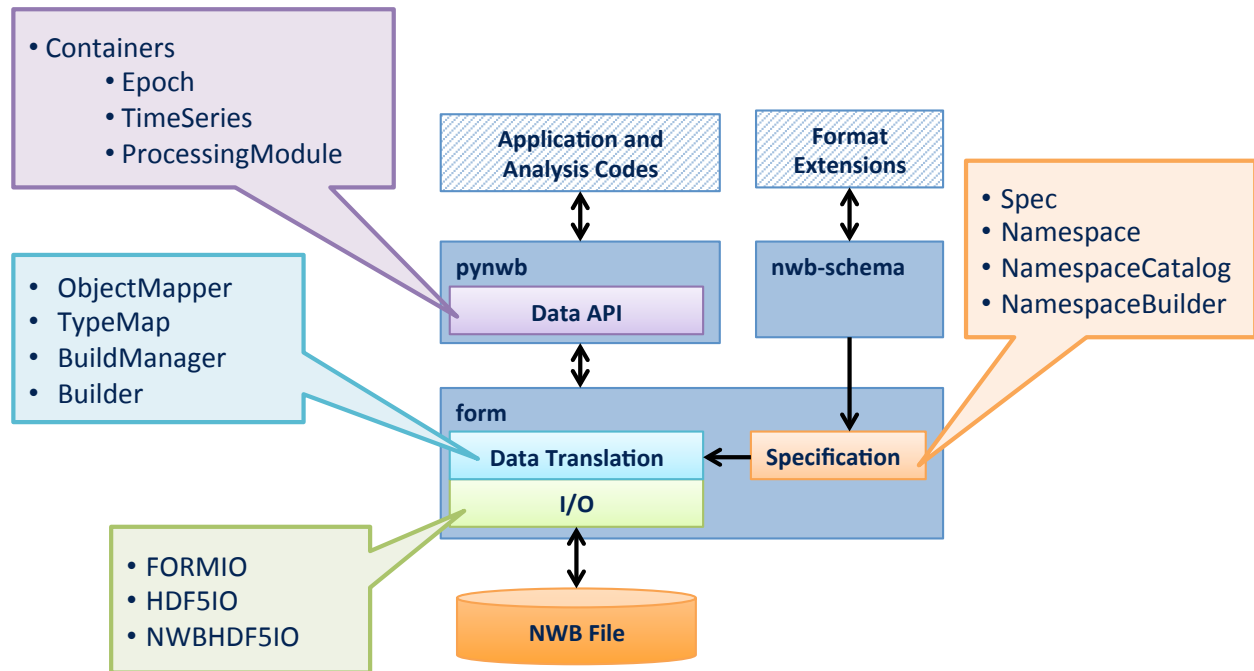


Fig. 3: Overview of the main concepts/classes in HDMF and their location in the overall software architecture (click to enlarge).

7.1.1 Container

- In memory objects
- Interface for (most) applications
- Similar to a table row
- HDMF does not provide these. They are left for standards developers to define how users interact with data.
- There are two Container base classes:
 - *Container* - represents a collection of objects
 - *Data* - represents data
- **Main Module:** `hdmf.container`

7.1.2 Builder

- Intermediary objects for I/O
- Interface for I/O
- Backend readers and writers must return and accept these
- There are different kinds of builders for different base types:
 - *GroupBuilder* - represents a collection of objects

- *DatasetBuilder* - represents data
- *LinkBuilder* - represents soft-links
- *RegionBuilder* - represents a slice into data (Subclass of *DatasetBuilder*)

- **Main Module:** *hdmf.build.builders*

7.1.3 Spec

- Interact with format specifications
- Data structures to specify data types and what said types consist of
- Python representation for YAML specifications
- Interface for writing extensions or custom specification
- There are several main specification classes:
 - *AttributeSpec* - specification for metadata
 - *GroupSpec* - specification for a collection of objects (i.e. subgroups, datasets, link)
 - *DatasetSpec* - specification for dataset (like and n-dimensional array). Specifies data type, dimensions, etc.
 - *LinkSpec* - specification for link (like a POSIX soft link)
 - *RefSpec* - specification for references (References are like links, but stored as data)
 - *DtypeSpec* - specification for compound data types. Used to build complex data type specification, e.g., to define tables (used only in *DatasetSpec* and correspondingly *DatasetSpec*)
- **Main Modules:** *hdmf.spec*

Note: A *data_type* defines a reusable type in a format specification that can be referenced and used elsewhere in other specifications. The specification of the standard is basically a collection of *data_types*,

- *data_type_inc* is used to include an existing type and
- *data_type_def* is used to define a new type

i.e, if both keys are defined then we create a new type that uses/inherits an existing type as a base.

7.1.4 ObjectMapper

- Maintains the mapping between *Container* attributes and *Spec* components
- Provides a way of converting between *Container* and *Builder*, while leaving standards developers with the flexibility of presenting data to users in a user-friendly manner, while storing data in an efficient manner
- ObjectMappers are constructed using a *Spec*
- Ideally, one ObjectMapper for each data type
- Things an ObjectMapper should do:
 - Given a *Builder*, return a Container representation
 - Given a *Container*, return a Builder representation
- **Main Module:** *hdmf.build.objectmapper*

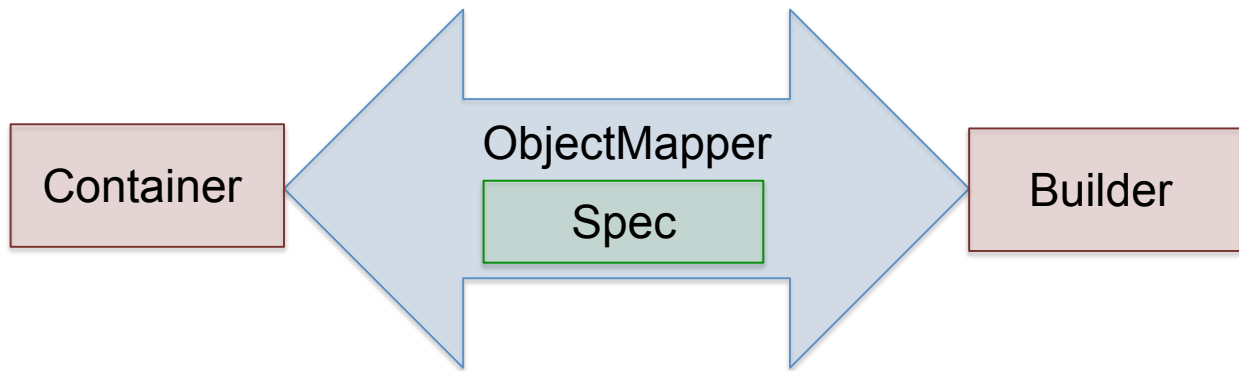


Fig. 4: Relationship between *Container*, *Builder*, *ObjectMapper*, and *Spec*

7.2 Additional Concepts

7.2.1 Namespace, NamespaceCatalog, NamespaceBuilder

- **Namespace**
 - A namespace for specifications
 - Necessary for making standards extensions and standard core specification
 - Contains basic info about who created extensions
- *NamespaceCatalog* – A class for managing namespaces
- *NamespaceBuilder* – A utility for building extensions

7.2.2 TypeMap

- Map between data types, Container classes (i.e. a Python class object) and corresponding ObjectMapper classes
- Constructed from a NamespaceCatalog
- Things a TypeMap does:
 - Given a *data_type*, return the associated Container class
 - Given a Container class, return the associated ObjectMapper
- HDMF has one of these classes:
 - the base class (i.e. *TypeMap*)
- TypeMaps can be merged, which is useful when combining extensions

7.2.3 BuildManager

- Responsible for memoizing *Builder* and *Container*
- Constructed from a *TypeMap*
- HDMF only has one of these: `hdmf.build.manager.BuildManager`

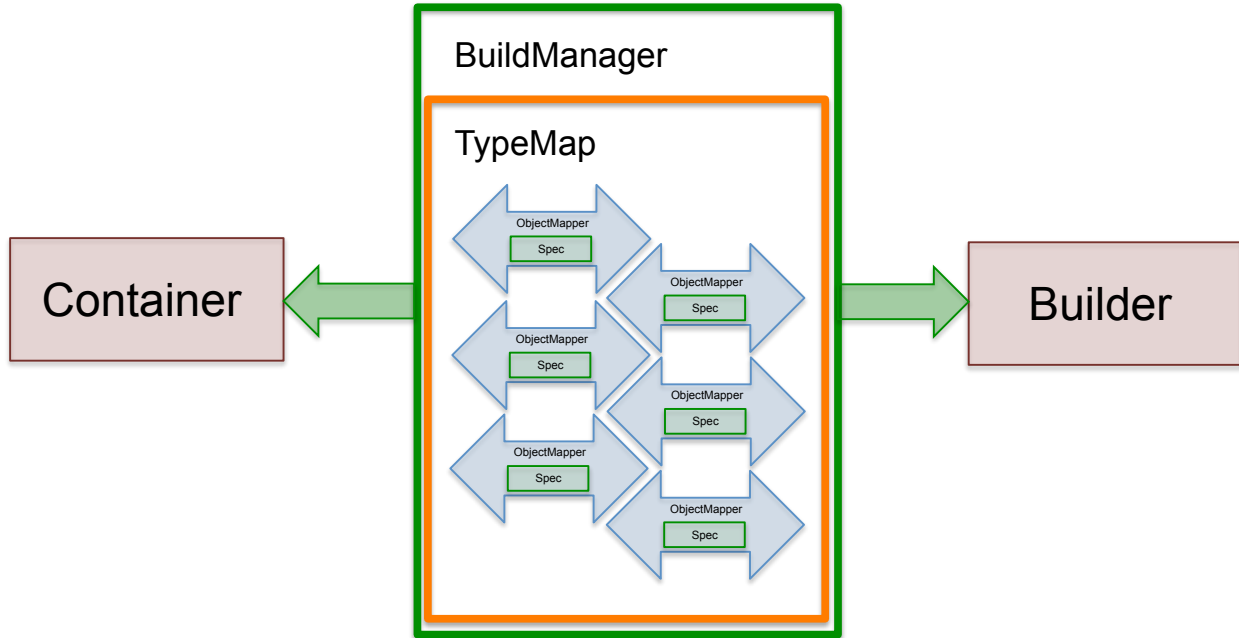


Fig. 5: Overview of *BuildManager* (and *TypeMap*) (click to enlarge).

7.2.4 HDMFIO

- Abstract base class for I/O
- *HDMFIO* has two key abstract methods:
 - *write_builder* – given a builder, write data to storage format
 - *read_builder* – given a handle to storage format, return builder representation
 - Others: *open* and *close*
- Constructed with a *BuildManager*
- Extend this for creating a new I/O backend
- HDMF has one concrete form of this:
 - *HDF5IO* - reading and writing HDF5

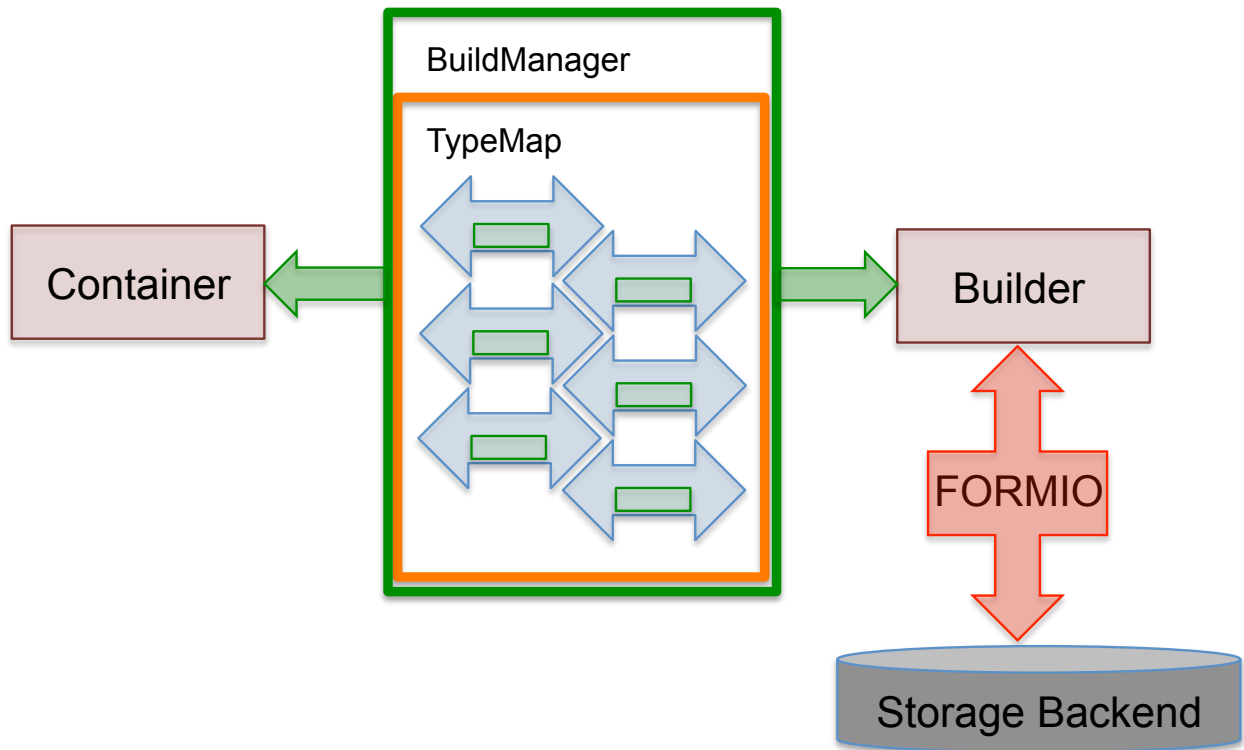


Fig. 6: Overview of *HDMFIO* (click to enlarge).

CHAPTER 8

Tutorials

The following page will discuss how to extend a standard using HDMF.

9.1 Creating new Extensions

Standards specified using HDMF are designed to be extended. Extension for a standard can be done so using classes provided in the *hdmf.spec* module. The classes *GroupSpec*, *DatasetSpec*, *AttributeSpec*, and *LinkSpec* can be used to define custom types.

9.1.1 Attribute Specifications

Specifying attributes is done with *AttributeSpec*.

```
from hdmf.spec import AttributeSpec

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

9.1.2 Dataset Specifications

Specifying datasets is done with *DatasetSpec*.

```
from hdmf.spec import DatasetSpec

spec = DatasetSpec('A custom data type',
                  name='qux',
                  attribute=[
                      AttributeSpec('baz', 'a value for baz', 'str'),
                  ],
                  shape=(None, None))
```

Using datasets to specify tables

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

```
from hdmf.spec import DatasetSpec, DtypeSpec

spec = DatasetSpec('A custom data type',
                   name='quux',
                   attribute=[
                       AttributeSpec('baz', 'a value for baz', 'str'),
                   ],
                   dtype=[
                       DtypeSpec('foo', 'column for foo', 'int'),
                       DtypeSpec('bar', 'a column for bar', 'float')
                   ])

```

9.1.3 Group Specifications

Specifying groups is done with the *GroupSpec* class.

```
from hdmf.spec import GroupSpec

spec = GroupSpec('A custom data type',
                 name='quux',
                 attributes=[...],
                 datasets=[...],
                 groups=[...])

```

9.1.4 Data Type Specifications

GroupSpec and *DatasetSpec* use the arguments *data_type_inc* and *data_type_def* for declaring new types and extending existing types. New types are specified by setting the argument *data_type_def*. New types can extend an existing type by specifying the argument *data_type_inc*.

Create a new type

```
from hdmf.spec import GroupSpec

# A list of AttributeSpec objects to specify new attributes
addl_attributes = [...]
# A list of DatasetSpec objects to specify new datasets
addl_datasets = [...]
# A list of DatasetSpec objects to specify new groups
addl_groups = [...]
spec = GroupSpec('A custom data type',
                 attributes=addl_attributes,
                 datasets=addl_datasets,
                 groups=addl_groups,
                 data_type_def='MyNewType')

```

Extend an existing type


```

from hdmf.spec import GroupSpec

# A list of AttributeSpec objects to specify additional attributes or attributes to_
↳be overridden
addl_attributes = [...]
# A list of DatasetSpec objects to specify additional datasets or datasets to be_
↳overridden
addl_datasets = [...]
# A list of GroupSpec objects to specify additional groups or groups to be overridden
addl_groups = [...]
spec = GroupSpec('An extended data type',
                 attributes=addl_attributes,
                 datasets=addl_datasets,
                 groups=addl_groups,
                 data_type_inc='SpikeEventSeries',
                 data_type_def='MyExtendedSpikeEventSeries')

```

Existing types can be instantiated by specifying *data_type_inc* alone.

```

from hdmf.spec import GroupSpec

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

spec = GroupSpec('An extended data type',
                 groups=addl_groups,
                 data_type_inc='SpikeEventSeries',
                 data_type_def='MyExtendedSpikeEventSeries')

```

Datasets can be extended in the same manner (with regard to *data_type_inc* and *data_type_def*, by using the class *DatasetSpec*.

9.2 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 *NamespaceBuilder* can be used to create new namespace and specification files.

Create a new namespace with extensions

```

from hdmf.spec import GroupSpec, NamespaceBuilder

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

# create extensions
ext1 = GroupSpec('A custom SpikeEventSeries interface',
                 attributes=[...]
                 datasets=[...],
                 groups=[...],
                 data_type_inc='SpikeEventSeries',
                 data_type_def='MyExtendedSpikeEventSeries')

ext2 = GroupSpec('A custom EventDetection interface',

```

(continues on next page)

(continued from previous page)

```

        attributes=[...]
        datasets=[...],
        groups=[...],
        data_type_inc='EventDetection',
        data_type_def='MyExtendedEventDetection')

# 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 HDMF specification language. It also helps with maintenance of extensions, e.g., if extensions have to be ported to newer versions of the specification language in the future.

9.3 Incorporating extensions

HDMF supports extending existing data types. Extensions must be registered with HDMF to be used for reading and writing of custom data types.

The following code demonstrates how to load custom namespaces.

```

from hdmf 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'.

9.3.1 Container : Representing custom data

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

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

```

from hdmf.common import register_class
from hdmf.container import Container

@register_class('MyExtension', 'my_namespace')

```

(continues on next page)

(continued from previous page)

```
class MyExtensionContainer(Container):
    ...
```

`register_class` can also be used as a function.

```
from hdmf.common import register_class
from hdmf.container import Container

class MyExtensionContainer(Container):
    ...

register_class(data_type='MyExtension', namespace='my_namespace', container_
↳cls=MyExtensionContainer)
```

If you do not have an *Container* 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 hdmf.common 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.

9.3.2 ObjectMapper : Customizing the mapping between Container and the Spec

If your *Container* extension requires custom mapping of the *Container* 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 hdmf.common import register_map
from hdmf.build import ObjectMapper

@register_map(MyExtensionContainer)
class MyExtensionMapper(ObjectMapper)
    ...
```

`register_map` can also be used as a function.

```
from hdmf.common import register_map
from hdmf.build import ObjectMapper

class MyExtensionMapper(ObjectMapper)
    ...

register_map(MyExtensionContainer, MyExtensionMapper)
```

Tip: *ObjectMappers* allow you to customize how objects in the spec are mapped to attributes of your *Container* in Python. This is useful, e.g., in cases where you want to customize the default mapping. For an overview of the concepts of containers, spec, builders, object mappers in HDMF see also *Software Architecture*

9.4 Documenting Extensions

Coming soon!

9.5 Further Reading

- **Specification Language:** For a detailed overview of the specification language itself see <https://schema-language.readthedocs.io/en/latest/>

Building API classes

After you have written an extension, you will need a Pythonic way to interact with the data model. To do this, you will need to write some classes that represent the data you defined in your specification extensions.

The `hdmf.container` module defines two base classes that represent the primitive structures supported by the schema. `Data` represents datasets and `Container` represents groups. See the classes in the `:py:mod:hdmf.common` package for examples.

10.1 The `register_class` function/decorator

When defining a class that represents a `data_type` (i.e. anything that has a `data_type_def`) from your extension, you can tell HDMF which `data_type` it represents using the function `register_class`. This class can be called on its own, or used as a class decorator. The first argument should be the `data_type` and the second argument should be the `namespace` name.

The following example demonstrates how to register a class as the Python class representation of the `data_type` “MyContainer” from the `namespace` “my_ns”. The namespace must be loaded prior to the below code using the `load_namespaces` function.

```
from hdmf.common import register_class
from hdmf.container import Container

class MyContainer(Container):
    ...

register_class(data_type='MyContainer', namespace='my_ns', container_cls=MyContainer)
```

Alternatively, you can use `register_class` as a decorator.

```
from hdmf.common import register_class
from hdmf.container import Container

@type_map.register_class('MyContainer', 'my_ns')
```

(continues on next page)

(continued from previous page)

```
class MyContainer(Container):  
    ...
```

register_class is used with *Data* the same way it is used with *Container*.

CHAPTER 11

Validating HDMF data

Validating HDMF structured data is handled by a command-line tool available in *hdmf*. The validator can be invoked like so:

```
python -m hdmf.validate -p namespace.yaml test.h5
```

This will validate the file `test.h5` against the specification in `namespace.yaml`.

12.1 hdmf.common package

12.1.1 Subpackages

hdmf.common.io package

Submodules

hdmf.common.io.table module

class hdmf.common.io.table.**DynamicTableMap** (*spec*)

Bases: *hdmf.build.objectmapper.ObjectMapper*

attr_columns (*container, manager*)

get_attr_value (*spec, container, manager*)

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** (*DynamicTable*) – the container to get the attribute value from
- **manager** (*BuildManager*) – the BuildManager used for managing this build

constructor_args = {'name': <function ObjectMapper.get_container_name>}

obj_attrs = {'colnames': <function DynamicTableMap.attr_columns>}

Module contents

12.1.2 Submodules

hdmf.common.sparse module

```
class hdmf.common.sparse.CSRMatrix(data, indices=None, indptr=None, shape=None,
                                   name='csr_matrix')
```

Bases: *hdmf.container.Container*

Parameters

- **data** (*csr_matrix* or *ndarray* or *Dataset*) – the data to use for this CSRMatrix or CSR data array. If passing CSR data array, *indices*, *indptr*, and *shape* must also be provided
- **indices** (*ndarray* or *Dataset*) – CSR index array
- **indptr** (*ndarray* or *Dataset*) – CSR index pointer array
- **shape** (*list* or *tuple* or *ndarray*) – the shape of the matrix
- **name** (*str*) – the name to use for this when storing

shape

to_spmat ()

data_type = 'CSRMatrix'

namespace = 'hdmf-common'

hdmf.common.table module

```
class hdmf.common.table.Index(name, data, target)
```

Bases: *hdmf.container.Data*

Parameters

- **name** (*str*) – the name of this VectorData
- **data** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *DataIO*) – a dataset where the first dimension is a concatenation of multiple vectors
- **target** (*Data*) – the target dataset that this index applies to

data_type = 'Index'

namespace = 'hdmf-common'

target

the target dataset that this index applies to

```
class hdmf.common.table.VectorData(name, description, data=[])
```

Bases: *hdmf.container.Data*

A n-dimensional dataset representing a column of a DynamicTable. If used without an accompanying VectorIndex, first dimension is along the rows of the DynamicTable and each step along the first dimension is a cell of the larger table. VectorData can also be used to represent a ragged array if paired with a VectorIndex. This allows for storing arrays of varying length in a single cell of the DynamicTable by indexing into this VectorData. The first vector is at VectorData[0:VectorIndex(0)+1]. The second vector is at VectorData[VectorIndex(0)+1:VectorIndex(1)+1], and so on.

Parameters

- **name** (*str*) – the name of this VectorData
- **description** (*str*) – a description for this column
- **data** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *DataIO*) – a dataset where the first dimension is a concatenation of multiple vectors

description

a description for this column

add_row (*val*)

Parameters *val* (*None*) – the value to add to this column

data_type = 'VectorData'

namespace = 'hdmf-common'

class `hdmf.common.table.VectorIndex` (*name, data, target*)

Bases: `hdmf.common.table.Index`

When paired with a VectorData, this allows for storing arrays of varying length in a single cell of the DynamicTable by indexing into this VectorData. The first vector is at `VectorData[0:VectorIndex(0)+1]`. The second vector is at `VectorData[VectorIndex(0)+1:VectorIndex(1)+1]`, and so on.

Parameters

- **name** (*str*) – the name of this VectorIndex
- **data** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *DataIO*) – a 1D dataset containing indexes that apply to VectorData object
- **target** (*VectorData*) – the target dataset that this index applies to

add_vector (*arg*)**add_row** (*arg*)**__getitem__** (*arg*)

data_type = 'VectorIndex'

namespace = 'hdmf-common'

class `hdmf.common.table.ElementIdentifiers` (*name, data=[]*)

Bases: `hdmf.container.Data`

Parameters

- **name** (*str*) – the name of this ElementIdentifiers
- **data** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *DataIO*) – a 1D dataset containing identifiers

data_type = 'ElementIdentifiers'

namespace = 'hdmf-common'

class `hdmf.common.table.DynamicTable` (*name, description, id=None, columns=None, column_names=None*)

Bases: `hdmf.container.Container`

A column-based table. Columns are defined by the argument *columns*. This argument must be a list/tuple of *VectorData* and *VectorIndex* objects or a list/tuple of dicts containing the keys *name* and *description* that provide the name and description of each column in the table. Additionally, the keys *index* and *table* for specifying additional structure to the table columns. Setting the key *index* to *True* can be used to indicate that the *VectorData* column will store a ragged array (i.e. will be accompanied with a *VectorIndex*). Setting the key *table* to *True* can be used to indicate that the column will store regions to another *DynamicTable*.

Columns in *DynamicTable* subclasses can be statically defined by specifying the class attribute `__columns__`, rather than specifying them at runtime at the instance level. This is useful for defining a table structure that will get reused. The requirements for `__columns__` are the same as the requirements described above for specifying table columns with the *columns* argument to the *DynamicTable* constructor.

Parameters

- **name** (*str*) – the name of this table
- **description** (*str*) – a description of what is in this table
- **id** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *ElementIdentifiers*) – the identifiers for this table
- **columns** (*tuple* or *list*) – the columns in this table
- **colnames** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator*) – the names of the columns in this table

description

a description of what is in this table

id

the identifiers for this table

colnames

the names of the columns in this table

columns

the columns in this table

add_row (*data=None, id=None, enforce_unique_id=False*)

Add a row to the table. If *id* is not provided, it will auto-increment.

Parameters

- **data** (*dict*) – the data to put in this row
- **id** (*int*) – the ID for the row
- **enforce_unique_id** (*bool*) – enforce that the id in the table must be unique

add_column (*name, description, data=[], table=False, index=False*)

Add a column to this table. If data is provided, it must contain the same number of rows as the current state of the table.

Parameters

- **name** (*str*) – the name of this *VectorData*
- **description** (*str*) – a description for this column

- **data** (`ndarray` or `list` or `tuple` or `Dataset` or `HDMFDataset` or `AbstractDataChunkIterator` or `DataIO`) – a dataset where the first dimension is a concatenation of multiple vectors
- **table** (`bool` or `DynamicTable`) – whether or not this is a table region or the table the region applies to
- **index** (`bool` or `VectorIndex` or `ndarray` or `list` or `tuple` or `Dataset` or `HDMFDataset` or `AbstractDataChunkIterator`) – whether or not this column should be indexed

create_region (*name, region, description*)

Parameters

- **name** (`str`) – the name of the `DynamicTableRegion` object
- **region** (`slice` or `list` or `tuple`) – the indices of the table
- **description** (`str`) – a brief description of what the region is

__getitem__ (*key*)

get (*key, default=None*)

to_dataframe (*exclude=None*)

Produce a pandas `DataFrame` containing this table's data.

Parameters **exclude** (`set`) – Set of columns to exclude from the dataframe

classmethod from_dataframe (*df, name, index_column=None, table_description="", columns=None*)

Construct an instance of `DynamicTable` (or a subclass) from a pandas `DataFrame`.

The columns of the resulting table are defined by the columns of the dataframe and the index by the dataframe's index (make sure it has a name!) or by a column whose name is supplied to the `index_column` parameter. We recommend that you supply `columns` - a list/tuple of dictionaries containing the name and description of the column- to help others understand the contents of your table. See [DynamicTable](#) for more details on `columns`.

Parameters

- **df** (`DataFrame`) – source `DataFrame`
- **name** (`str`) – the name of this table
- **index_column** (`str`) – if provided, this column will become the table's index
- **table_description** (`str`) – a description of what is in the resulting table
- **columns** (`list` or `tuple`) – a list/tuple of dictionaries specifying columns in the table

copy ()

Return a copy of this `DynamicTable`. This is useful for linking.

data_type = `'DynamicTable'`

namespace = `'hdmf-common'`

class `hdmf.common.table.DynamicTableRegion` (*name, data, description, table=None*)

Bases: `hdmf.common.table.VectorData`

`DynamicTableRegion` provides a link from one table to an index or region of another. The `table` attribute is another `DynamicTable`, indicating which table is referenced. The data is `int(s)` indicating the row(s) (0-indexed) of the target array. `DynamicTableRegion`'s can be used to associate multiple rows with the same meta-data without

data duplication. They can also be used to create hierarchical relationships between multiple 'DynamicTable's. 'DynamicTableRegion' objects may be paired with a VectorIndex object to create ragged references, so a single cell of a DynamicTable can reference many rows of another DynamicTable.

Parameters

- **name** (*str*) – the name of this VectorData
- **data** (*ndarray* or *list* or *tuple* or *Dataset* or *HDMFDataset* or *AbstractDataChunkIterator* or *DataIO*) – a dataset where the first dimension is a concatenation of multiple vectors
- **description** (*str*) – a description of what this region represents
- **table** (*DynamicTable*) – the DynamicTable this region applies to

data_type = 'DynamicTableRegion'

namespace = 'hdmf-common'

table

__getitem__ (*key*)

to_dataframe (***kwargs*)

Convert the whole DynamicTableRegion to a pandas dataframe.

Keyword arguments are passed through to the to_dataframe method of DynamicTable that is being referenced (i.e., self.table). This allows specification of the 'exclude' parameter and any other parameters of DynamicTable.to_dataframe.

shape

Define the shape, i.e., (num_rows, num_columns) of the selected table region :return: Shape tuple with two integers indicating the number of rows and number of columns

12.1.3 Module contents

This package will contain functions, classes, and objects for reading and writing data in according to the HDMF-common specification

`hdmf.common.register_class` (*data_type*, *namespace='hdmf-common'*, *container_cls=None*)

Register an Container class to use for reading and writing a data_type from a specification If `container_cls` is not specified, returns a decorator for registering an Container subclass as the class for `data_type` in `namespace`.

Parameters

- **data_type** (*str*) – the data_type to get the spec for
- **namespace** (*str*) – the name of the namespace
- **container_cls** (*type*) – the class to map to the specified data_type

`hdmf.common.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

`hdmf.common.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*

`hdmf.common.available_namespaces()`

`hdmf.common.get_type_map(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*

`hdmf.common.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*

`hdmf.common.get_class(data_type, namespace)`

Get the class object of the Container subclass corresponding to a given neurdata_type.

Parameters

- **data_type** (*str*) – the data_type to get the Container class for
- **namespace** (*str*) – the namespace the data_type is defined in

`hdmf.common.validate(io, namespace='hdmf-common')`

Validate an file against a namespace

Parameters

- **io** (*HDMFIO*) – the HDMFIO object to read from
- **namespace** (*str*) – the namespace to validate against

Returns errors in the file

Return type *list*

12.2 hdmf.container module

class `hdmf.container.AbstractContainer` (*name*)

Bases: `object`

Parameters `name` (`str`) – the name of this container

name

The name of this Container

get_ancestor (*data_type=None*)

Traverse parent hierarchy and return first instance of the specified `data_type`

Parameters `data_type` (`str`) – the `data_type` to search for

fields

object_id

modified

set_modified (*modified=True*)

Parameters `modified` (`bool`) – whether or not this Container has been modified

children

add_child (*child=None*)

Parameters `child` (`Container`) – the child Container for this Container

classmethod `type_hierarchy` ()

container_source

The source of this Container

parent

The parent Container of this Container

class `hdmf.container.Container` (*name*)

Bases: `hdmf.container.AbstractContainer`

Parameters `name` (`str`) – the name of this container

data_type = `'Container'`

namespace = `'hdmf-common'`

class `hdmf.container.Data` (*name, data*)

Bases: `hdmf.container.AbstractContainer`

A class for representing dataset containers

Parameters

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

data

shape

Get the shape of the data represented by this container :return: Shape tuple :rtype: tuple of ints

set_dataio (*dataio*)

Apply `DataIO` object to the data held by this `Data` object

Parameters **dataio** (*DataIO*) – the DataIO to apply to the data held by this Data

`__getitem__` (*args*)

`append` (*arg*)

`extend` (*arg*)

`data_type` = 'Data'

`namespace` = 'hdmf-common'

class `hdmf.container.DataRegion` (*name, data*)

Bases: `hdmf.container.Data`

Parameters

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

data

The target data that this region applies to

region

The region that indexes into data e.g. slice or list of indices

12.3 hdmf.build package

12.3.1 Submodules

hdmf.build.builders module

class `hdmf.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

path

Get the path of this Builder

written

The source of this Builder

name

The name of this Builder

source

The source of this Builder

parent

The parent Builder of this Builder

class `hdmf.build.builders.BaseBuilder` (*name, attributes={}, parent=None, source=None*)

Bases: `hdmf.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

location

The location of this Builder in its source

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 hdmf.build.builders.GroupBuilder (name, groups={}, datasets={}, attributes={}, links={}, parent=None, source=None)
```

Bases: *hdmf.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

set_builder (*builder*)

Add an existing builder to this this GroupBuilder

Parameters **builder** (*Builder*) – the Builder to add to this GroupBuilder

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 *HDMFDataset* or *AbstractDataChunkIterator* 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 (*target, name=None*)

Create a soft link and add it to this group

Parameters

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

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.

__getitem__ (*key*)

Like dict.__getitem__, but looks in groups, datasets, attributes, and links sub-dictionaries.

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 hdmf.build.builders.DatasetBuilder (name, data=None, dtype=None, attributes={}, maxshape=None, chunks=False, parent=None, source=None)
```

Bases: *hdmf.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 *HDMFDataset* or *AbstractDataChunkIterator* or *str* or *int* or *float* or *DataIO* or *DatasetBuilder* or *RegionBuilder* or *Iterable* or *datetime*) – 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 `hdmf.build.builders.LinkBuilder` (*builder, name=None, parent=None, source=None*)

Bases: `hdmf.build.builders.Builder`

Parameters

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

builder

The target builder object

class `hdmf.build.builders.ReferenceBuilder` (*builder*)

Bases: `dict`

Parameters **builder** (*DatasetBuilder* or *GroupBuilder*) – the Dataset this region applies to

builder

The target builder object

class `hdmf.build.builders.RegionBuilder` (*region, builder*)

Bases: `hdmf.build.builders.ReferenceBuilder`

Parameters

- **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

region

The target builder object

hdmf.build.manager module

class `hdmf.build.manager.Proxy` (*manager, source, location, namespace, data_type*)

Bases: `object`

A temporary object to represent a Container. This gets used when resolving the true location of a Container's parent. Proxy objects allow simple bookkeeping of all potential parents a Container may have. This object is used by providing all the necessary information for describing the object. This object gets passed around and candidates are accumulated. Upon calling `resolve`, all saved candidates are matched against the information (provided to the constructor). The candidate that has an exact match is returned.

source

The source of the object e.g. file source

location

The location of the object. This can be thought of as a unique path

namespace

The namespace from which the data_type of this Proxy came from

data_type

The data_type of Container that should match this Proxy

matches (*object*)

Parameters **object** (*BaseBuilder* or *Container*) – the container or builder to get a proxy for

add_candidate (*container*)

Parameters **container** (*Container*) – the Container to add as a candidate match

resolve ()**class** `hdmf.build.manager.BuildManager` (*type_map*)

Bases: `object`

A class for managing builds of AbstractContainers

namespace_catalog**type_map****get_proxy** (*object*, *source=None*)**Parameters**

- **object** (*BaseBuilder* or *AbstractContainer*) – the container or builder to get a proxy for
- **source** (`str`) – the source of container being built i.e. file path

build (*container*, *source=None*, *spec_ext=None*)

Build the GroupBuilder/DatasetBuilder for the given AbstractContainer

Parameters

- **container** (*AbstractContainer*) – the container to convert to a Builder
- **source** (`str`) – the source of container being built i.e. file path
- **spec_ext** (*BaseStorageSpec*) – a spec that further refines the base specification

prebuilt (*container*, *builder*)

Save the Builder for a given AbstractContainer for future use

Parameters

- **container** (*AbstractContainer*) – the AbstractContainer to save as prebuilt
- **builder** (*DatasetBuilder* or *GroupBuilder*) – the Builder representation of the given container

construct (*builder*)

Construct the AbstractContainer represented by the given builder

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

get_builder_ns (*builder*)

Get the namespace of a builder

Parameters **builder** (*DatasetBuilder* or *GroupBuilder* or *LinkBuilder*) – the builder to get the sub-specification for

get_builder_dt (*builder*)

Get the data_type of a builder

Parameters **builder** (*DatasetBuilder* or *GroupBuilder* or *LinkBuilder*) – the builder to get the data_type for

class `hdmf.build.manager.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 `hdmf.build.manager.TypeMap` (*namespaces=None, mapper_cls=None*)

Bases: *object*

A class to maintain the map between ObjectMappers and AbstractContainer classes

Parameters

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

namespace_catalog

copy_mappers (*type_map*)

merge (*type_map, ns_catalog=False*)

load_namespaces (*namespace_path, resolve=True, reader=None*)

Load namespaces from a namespace file. This method will call `load_namespaces` on the `Namespace-Catalog` 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
- **reader** (*SpecReader*) – the class to user for reading specifications

Returns the namespaces loaded from the given file

Return type *dict*

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 `AbstractContainer` class for

Returns the class for the given namespace and `data_type`

Return type *type*

get_builder_dt (*builder*)

Get the `data_type` of a builder

Parameters **builder** (*DatasetBuilder* or *GroupBuilder* or *LinkBuilder*) – the builder to get the `data_type` for

get_builder_ns (*builder*)

Get the namespace of a builder

Parameters **builder** (*DatasetBuilder* or *GroupBuilder* or *LinkBuilder*) – the builder to get the sub-specification for

get_cls (*builder*)

Get the class object for the given Builder

Parameters **builder** (*Builder*) – the Builder object to get the corresponding `AbstractContainer` 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_ns_dt (*obj*)

get_container_cls_dt (*cls*)

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` (*AbstractContainer* 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 map 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 AbstractContainer 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, builder=None, spec_ext=None*)

Build the GroupBuilder/DatasetBuilder for the given AbstractContainer

Parameters

- **container** (*AbstractContainer*) – 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
- **builder** (*BaseBuilder*) – the Builder to build on
- **spec_ext** (*BaseStorageSpec*) – a spec extension

construct (*builder, build_manager=None, parent=None*)

Construct the AbstractContainer represented by the given builder

Parameters

- **builder** (*DatasetBuilder* or *GroupBuilder*) – the builder to construct the AbstractContainer from
- **build_manager** (*BuildManager*) – the BuildManager for constructing
- **parent** (*Proxy* or *Container*) – the parent Container/Proxy for the Container being built

get_builder_name (*container*)

Get the name a Builder should be given

Parameters `container` (*AbstractContainer*) – the container to convert to a Builder

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

Return type `str`

exception `hdmf.build.manager.TypeDoesNotExistError`
Bases: `Exception`

`hdmf.build.map` module

`hdmf.build.objectmapper` module

class `hdmf.build.objectmapper.ObjectMapper` (*spec*)
Bases: `object`

A class for mapping between Spec objects and AbstractContainer attributes

Create a map from AbstractContainer attributes to specifications

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

classmethod `no_convert` (*obj_type*)
Specify an object type that ObjectMappers should not convert.

classmethod `convert_dtype` (*spec*, *value*)
Convert values to the specified dtype. For example, if a literal int is passed in to a field that is specified as a unsigned integer, this function will convert the Python int to a numpy unsigned int.

Returns The function returns a tuple consisting of 1) the value, and 2) the data type. The value is returned as the function may convert the input value to comply with the dtype specified in the schema.

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 AbstractContainer 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 AbstractContainer 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 AbstractContainer and Builder objects. The decorated method should accept as its first argument the AbstractContainer 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

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*, *manager*)

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** (*AbstractContainer*) – the container to get the attribute value from
- **manager** (*BuildManager*) – the *BuildManager* used for managing this build

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, builder=None, spec_ext=None*)

Convert a AbstractContainer to a Builder representation

Parameters

- **container** (*AbstractContainer*) – the container to convert to a Builder
- **manager** (*BuildManager*) – the BuildManager to use for managing this build
- **parent** (*GroupBuilder*) – the parent of the resulting Builder
- **source** (*str*) – the source of container being built i.e. file path
- **builder** (*BaseBuilder*) – the Builder to build on
- **spec_ext** (*BaseStorageSpec*) – a spec extension

Returns the Builder representing the given AbstractContainer

Return type *Builder*

construct (*builder, manager, parent=None*)

Construct an AbstractContainer from the given Builder

Parameters

- **builder** (*DatasetBuilder* or *GroupBuilder*) – the builder to construct the AbstractContainer from
- **manager** (*BuildManager*) – the BuildManager for this build
- **parent** (*Proxy* or *AbstractContainer*) – the parent AbstractContainer/Proxy for the AbstractContainer being built

get_builder_name (*container*)

Get the name of a Builder that represents a AbstractContainer

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

```
constructor_args = {'name': <function ObjectMapper.get_container_name>}
```

```
obj_attrs = {}
```

hdmf.build.warnings module

exception `hdmf.build.warnings.OrphanContainerWarning`

Bases: `UserWarning`

Raised when a container does not have a parent.

Only the top level container (e.g. file) should be without a parent

exception `hdmf.build.warnings.MissingRequiredWarning`

Bases: `UserWarning`

Raised when a required field is missing.

12.3.2 Module contents

12.4 hdmf.spec package

12.4.1 Submodules

hdmf.spec.catalog module

class `hdmf.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 calculation 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 the 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 originally 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*)

For a given type get the type inheritance hierarchy for that type.

E.g., if we have a type MyContainer that inherits from BaseContainer then the result will be a tuple with the strings ('MyContainer', 'BaseContainer')

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

Returns Tuple of strings with the names of the types the given *data_type* inherits from.

Return type *tuple*

get_full_hierarchy ()

Get the complete hierarchy of all types. The function attempts to sort types by name using standard Python sorted.

Returns Hierarchically nested OrderedDict with the hierarchy of all the types

Return type *OrderedDict*

get_subtypes (*data_type*, *recursive=True*)

For a given data type recursively find all the subtypes that inherit from it.

E.g., assume we have the following inheritance hierarchy:

```
-BaseContainer--+->AContainer--->ADContainer
                  |
                  +-->BContainer
```

In this case, the subtypes of BaseContainer would be (AContainer, ADContainer, BContainer), the subtypes of AContainer would be (ADContainer), and the subtypes of BContainer would be empty ().

Parameters

- **data_type** (*str* or *type*) – the *data_type* to get the subtypes for
- **recursive** (*bool*) – recursively get all subtypes. Set to False to only get the direct subtypes

Returns Tuple of strings with the names of all types of the given *data_type*.

Return type *tuple*

hdmf.spec.namespace module

class `hdmf.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`

UNVERSIONED = `None`

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 `SpecNamespace.UNVERSIONED` if the version is missing or empty

date

Date last modified or released.

Returns `datetime` object, string, or `None`

name

String with short name or `None`

doc

schema

get_source_files()

Get the list of names of the source files included the schema of the namespace

get_source_description (*sourcefile*)

Get the description of a source file as described in the namespace. The result is a dict which contains the 'source' and optionally 'title', 'doc' and 'data_types' imported from the source file

Parameters `sourcefile` (`str`) – Name of the source file

Returns Dict with the source file documentation

Return type `dict`

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 `hdmf.spec.namespace.SpecReader` (`source`)

Bases: `object`

Parameters `source` (`str`) – the source from which this reader reads from

`source`

`read_spec` ()

`read_namespace` ()

class `hdmf.spec.namespace.YAMLSpecReader` (`indir='.'`)

Bases: `hdmf.spec.namespace.SpecReader`

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

`read_namespace` (`namespace_path`)

`read_spec` (`spec_path`)

class `hdmf.spec.namespace.NamespaceCatalog` (`group_spec_cls=<class 'hdmf.spec.spec.GroupSpec'>`,
`dataset_spec_cls=<class 'hdmf.spec.spec.DatasetSpec'>`,
`spec_namespace_cls=<class 'hdmf.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

merge (*ns_catalog*)

namespaces

The namespaces in this NamespaceCatalog

Returns a tuple of the available 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 use for reading specifications

Returns a dictionary describing the dependencies of loaded namespaces

Return type *dict*

hdmf.spec.spec module

class `hdmf.spec.spec.DtypeHelper`

Bases: *object*

primary_dtype_synonyms = {'ascii': ['ascii', 'bytes'], 'bool': ['bool'], 'double':

recommended_primary_dtypes = ['float', 'double', 'short', 'int', 'long', 'utf', 'ascii

valid_primary_dtypes = {'ascii', 'bool', 'bytes', 'datetime', 'double', 'float', 'float

static simplify_cpd_type (*cpd_type*)

Transform a list of DtypeSpecs into a list of strings. Use for simple representation of compound type and validation.

Parameters **cpd_type** (*list*) – The list of DtypeSpecs to simplify

class `hdmf.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 `hdmf.spec.spec.Spec` (*doc*, *name=None*, *required=True*, *parent=None*)

Bases: *hdmf.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 `hdmf.spec.spec.RefSpec` (*target_type*, *reftype*)

Bases: `hdmf.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 `hdmf.spec.spec.AttributeSpec` (*name*, *doc*, *dtype*, *shape=None*, *dims=None*, *required=True*, *parent=None*, *value=None*, *default_value=None*)

Bases: `hdmf.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* or *RefSpec*) – 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** (*BaseStorageSpec*) – 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

classmethod build_const_args (*spec_dict*)

Build constructor arguments for this Spec class from a dictionary

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

Bases: *hdmf.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, allowed only if quantity is not ‘+’ or ‘*’
- **default_name** (*str*) – The default name of this base storage container, used only if name is None
- **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 data type this specification represents
- **data_type_inc** (*str* or *BaseStorageSpec*) – the data 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 `id_key` ()
Get the key used to store data ID on an instance
Override this method to use a different name for ‘object_id’

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` or `RefSpec`) – 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** (`BaseStorageSpec`) – 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 `hdmf.spec.spec.DtypeSpec` (*name, doc, dtype*)

Bases: `hdmf.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 `hdmf.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: `hdmf.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 Dtype-Specs 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 data type this specification represents
- **data_type_inc** (*str* or *DatasetSpec*) – the data 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

default_value

The default value of the dataset or None if not specified

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 `hdmf.spec.spec.LinkSpec` (*doc*, *target_type*, *quantity=1*, *name=None*)

Bases: `hdmf.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 hdmf.spec.spec.GroupSpec (doc, name=None, default_name=None, groups=[],  
                                datasets=[], attributes=[], links=[], linkable=True, quan-  
                                tity=1, data_type_def=None, data_type_inc=None)
```

Bases: *hdmf.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 data type this specification represents
- **data_type_inc** (*str* or *GroupSpec*) – the data 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

is_overridden_type (*spec*)

Returns True if *spec* represents a spec that was overridden by the subtype

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 specified in this GroupSpec

datasets

The datasets specified in this GroupSpec

links

The links specified 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 data type this specification represents
- **data_type_inc** (*str* or GroupSpec) – the data 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 data type this specification represents
- **data_type_inc** (*str* or *DatasetSpec*) – the data 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

hdmf.spec.write module

class hdmf.spec.write.SpecWriter

Bases: `object`

write_spec (spec_file_dict, path)

write_namespace (namespace, path)

class hdmf.spec.write.YAMLSpecWriter (outdir='.')

Bases: `hdmf.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)

Write the given namespace key-value pairs as YAML to the given path.

Parameters

- **namespace** – SpecNamespace holding the key-value pairs that define the namespace
- **path** – File path to write the namespace to as YAML under the key ‘namespaces’

reorder_yaml (path)

Open a YAML file, load it as python data, sort the data alphabetically, and write it back out to the same path.

sort_keys (obj)

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

Bases: `object`

A class for building namespace and spec files

Parameters

- **doc (str)** – Description about what the namespace represents
- **name (str)** – Name of the namespace
- **full_name (str)** – Extended full name of the 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
- **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
- **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, doc=None, title=None*)

Add a source file to the namespace

Parameters

- **source** (*str*) – the path to write the spec to
- **doc** (*str*) – additional documentation for the source file
- **title** (*str*) – optional heading to be used for the source

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

name

class `hdmf.spec.write.SpecFileBuilder`

Bases: `dict`

add_spec (*spec*)

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

`hdmf.spec.write.export_spec` (*ns_builder, new_data_types, output_dir*)

Create YAML specification files for a new namespace and extensions with the given data type specs.

Parameters

- – **NamespaceBuilder** instance used to build the namespace and extension (*ns_builder*) –

- - **Iterable of specs that represent new data types** (*new_data_types*) – to be added

12.4.2 Module contents

12.5 hdmf.backends package

12.5.1 Subpackages

hdmf.backends.hdf5 package

Submodules

hdmf.backends.hdf5.h5_utils module

class `hdmf.backends.hdf5.h5_utils.H5Dataset` (*dataset, io*)

Bases: `hdmf.query.HDMFDataset`

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (*HDF5IO*) – the IO object that was used to read the underlying dataset

io

regionref

ref

shape

class `hdmf.backends.hdf5.h5_utils.DatasetOfReferences` (*dataset, io*)

Bases: `hdmf.backends.hdf5.h5_utils.H5Dataset, hdmf.query.ReferenceResolver`

An extension of the base ReferenceResolver class to add more abstract methods for subclasses that will read HDF5 references

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (*HDF5IO*) – the IO object that was used to read the underlying dataset

get_object (*h5obj*)

A class that maps an HDF5 object to a Builder or Container

invert ()

Return an object that defers reference resolution but in the opposite direction.

class `hdmf.backends.hdf5.h5_utils.BuilderResolverMixin`

Bases: `hdmf.query.BuilderResolver`

A mixin for adding to HDF5 reference-resolving types the `get_object` method that returns Builders

get_object (*h5obj*)

A class that maps an HDF5 object to a Builder

class `hdmf.backends.hdf5.h5_utils.ContainerResolverMixin`

Bases: `hdmf.query.ContainerResolver`

A mixin for adding to HDF5 reference-resolving types the `get_object` method that returns Containers

get_object (*h5obj*)

A class that maps an HDF5 object to a Container

class `hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset` (*dataset, io, types*)

Bases: `hdmf.backends.hdf5.h5_utils.DatasetOfReferences`

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset
- **types** (list or tuple) – the IO object that was used to read the underlying dataset

types

dtype

__getitem__ (*arg*)

resolve (*manager*)

class `hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset` (*dataset, io*)

Bases: `hdmf.backends.hdf5.h5_utils.DatasetOfReferences`

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset

__getitem__ (*arg*)

dtype

class `hdmf.backends.hdf5.h5_utils.AbstractH5RegionDataset` (*dataset, io*)

Bases: `hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset`

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset

__getitem__ (*arg*)

dtype

class `hdmf.backends.hdf5.h5_utils.ContainerH5TableDataset` (*dataset, io, types*)

Bases: `hdmf.backends.hdf5.h5_utils.ContainerResolverMixin`, `hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset`

A reference-resolving dataset for resolving references inside tables (i.e. compound dtypes) that returns resolved references as Containers

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset
- **types** (list or tuple) – the IO object that was used to read the underlying dataset

classmethod `get_inverse_class()`

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

class `hdmf.backends.hdf5.h5_utils.BuilderH5TableDataset(dataset, io, types)`

Bases: `hdmf.backends.hdf5.h5_utils.BuilderResolverMixin`, `hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset`

A reference-resolving dataset for resolving references inside tables (i.e. compound dtypes) that returns resolved references as Builders

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset
- **types** (list or tuple) – the IO object that was used to read the underlying dataset

classmethod `get_inverse_class()`

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

class `hdmf.backends.hdf5.h5_utils.ContainerH5ReferenceDataset(dataset, io)`

Bases: `hdmf.backends.hdf5.h5_utils.ContainerResolverMixin`, `hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset`

A reference-resolving dataset for resolving object references that returns resolved references as Containers

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset

classmethod `get_inverse_class()`

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

class `hdmf.backends.hdf5.h5_utils.BuilderH5ReferenceDataset(dataset, io)`

Bases: `hdmf.backends.hdf5.h5_utils.BuilderResolverMixin`, `hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset`

A reference-resolving dataset for resolving object references that returns resolved references as Builders

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (HDF5IO) – the IO object that was used to read the underlying dataset

classmethod `get_inverse_class()`

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

```
class hdmf.backends.hdf5.h5_utils.ContainerH5RegionDataset (dataset, io)  
  Bases: hdmf.backends.hdf5.h5_utils.ContainerResolverMixin, hdmf.backends.hdf5.h5_utils.AbstractH5RegionDataset
```

A reference-resolving dataset for resolving region references that returns resolved references as Containers

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (*HDF5IO*) – the IO object that was used to read the underlying dataset

```
classmethod get_inverse_class ()
```

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

```
class hdmf.backends.hdf5.h5_utils.BuilderH5RegionDataset (dataset, io)  
  Bases: hdmf.backends.hdf5.h5_utils.BuilderResolverMixin, hdmf.backends.hdf5.h5_utils.AbstractH5RegionDataset
```

A reference-resolving dataset for resolving region references that returns resolved references as Builders

Parameters

- **dataset** (Dataset or *Array*) – the HDF5 file lazily evaluate
- **io** (*HDF5IO*) – the IO object that was used to read the underlying dataset

```
classmethod get_inverse_class ()
```

Return the class that represents the ReferenceResolver that resolves references to the opposite type.

BuilderResolver.get_inverse_class should return a class that subclasses ContainerResolver.

ContainerResolver.get_inverse_class should return a class that subclasses BuilderResolver.

```
class hdmf.backends.hdf5.h5_utils.H5SpecWriter (group)  
  Bases: hdmf.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_spec (spec, path)
```

```
write_namespace (namespace, path)
```

```
class hdmf.backends.hdf5.h5_utils.H5SpecReader (group)  
  Bases: hdmf.spec.namespace.SpecReader
```

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

```
read_spec (spec_path)
```

```
read_namespace (ns_path)
```

```
class hdmf.backends.hdf5.h5_utils.H5RegionSlicer (dataset, region)  
  Bases: hdmf.region.RegionSlicer
```

Parameters

- **dataset** (Dataset or *H5Dataset*) – the HDF5 dataset to slice
- **region** (RegionReference) – the region reference to use to slice

`__getitem__` (*idx*)

Must be implemented by subclasses

```
class hdmf.backends.hdf5.h5_utils.H5DataIO (data=None, maxshape=None, chunks=None,
                                           compression=None, compression_opts=None,
                                           fillvalue=None, shuffle=None,
                                           fletcher32=None, link_data=False, al-
                                           low_plugin_filters=False)
```

Bases: `hdmf.data_utils.DataIO`

Wrap data arrays for write via HDF5IO to customize I/O behavior, such as compression and chunking for data arrays.

Parameters

- **data** (`ndarray` or `list` or `tuple` or `Dataset` or `Iterable`) – the data to be written. NOTE: If an `h5py.Dataset` is used, all other settings but `link_data` will be ignored as the dataset will either be linked to or copied as is in `H5DataIO`.
- **maxshape** (`tuple`) – Dataset will be resizable up to this shape (Tuple). Automatically enables chunking. Use `None` for the axes you want to be unlimited.
- **chunks** (`bool` or `tuple`) – Chunk shape or `True` to enable auto-chunking
- **compression** (`str` or `bool` or `int`) – Compression strategy. If a `bool` is given, then `gzip` compression will be used by default. <http://docs.h5py.org/en/latest/high/dataset.html#dataset-compression>
- **compression_opts** (`int` or `tuple`) – Parameter for compression filter
- **fillvalue** (`None`) – Value to be returned when reading uninitialized parts of the dataset
- **shuffle** (`bool`) – Enable shuffle I/O filter. <http://docs.h5py.org/en/latest/high/dataset.html#dataset-shuffle>
- **fletcher32** (`bool`) – Enable fletcher32 checksum. <http://docs.h5py.org/en/latest/high/dataset.html#dataset-fletcher32>
- **link_data** (`bool`) – If data is an `h5py.Dataset` should it be linked to or copied. NOTE: This parameter is only allowed if data is an `h5py.Dataset`
- **allow_plugin_filters** (`bool`) – Enable passing dynamically loaded filters as compression parameter

`get_io_params` ()

Returns a dict with the I/O parameters specified in this `DataIO`.

static filter_available (*filter*, *allow_plugin_filters*)

Check if a given I/O filter is available

Parameters

- **filter** (`String`, `int`) – String with the name of the filter, e.g., `gzip`, `szip` etc. `int` with the registered filter ID, e.g. 307
- **allow_plugin_filters** – `bool` indicating whether the given filter can be dynamically loaded

Returns `bool` indicating whether the given filter is available

`link_data`

`io_settings`

valid

bool indicating if the data object is valid

hdmf.backends.hdf5.h5tools module

class `hdmf.backends.hdf5.h5tools.HDF5IO` (*path*, *mode*, *manager=None*, *comm=None*,
file=None)

Bases: `hdmf.backends.io.HDMFIO`

Open an HDF5 file for IO

For *mode*, see `h5py.File` <<http://docs.h5py.org/en/latest/high/file.html#opening-creating-files>>_.

Parameters

- **path** (*str*) – the path to the HDF5 file
- **mode** (*str*) – the mode to open the HDF5 file with, one of (“w”, “r”, “r+”, “a”, “w-“, “x”)
- **manager** (*TypeMap* or *BuildManager*) – the *BuildManager* or a *TypeMap* to construct a *BuildManager* to use for I/O
- **comm** (*Intracomm*) – the MPI communicator to use for parallel I/O
- **file** (*File*) – a pre-existing *h5py.File* object

comm

classmethod `load_namespaces` (*namespace_catalog*, *path*, *namespaces=None*)

Load cached namespaces from a file.

Parameters

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

Returns dict with the loaded namespaces

Return type dict

classmethod `copy_file` (*source_filename*, *dest_filename*, *expand_external=True*, *expand_refs=False*, *expand_soft=False*)

Convenience function to copy an HDF5 file while allowing external links to be resolved.

NOTE: The source file will be opened in ‘r’ mode and the destination file will be opened in ‘w’ mode using *h5py*. To avoid possible collisions, care should be taken that, e.g., the source file is not opened already when calling this function.

Parameters

- **source_filename** (*str*) – the path to the HDF5 file to copy
- **dest_filename** (*str*) – the name of the destination file
- **expand_external** (*bool*) – expand external links into new objects
- **expand_refs** (*bool*) – copy objects which are pointed to by reference
- **expand_soft** (*bool*) – expand soft links into new objects

write (*container*, *cache_spec=True*, *link_data=True*, *exhaust_dci=True*)

Parameters

- **container** (*Container*) – the Container object to write
- **cache_spec** (*bool*) – cache specification to file
- **link_data** (*bool*) – If not specified otherwise link (True) or copy (False) HDF5 Datasets
- **exhaust_dci** (*bool*) – exhaust DataChunkIterators one at a time. If False, exhaust them concurrently

read ()

Returns the Container object that was read in

Return type *Container*

read_builder ()

Returns a GroupBuilder representing the data object

Return type *GroupBuilder*

get_builder (*h5obj*)

Get the builder for the corresponding h5py Group or Dataset

raises ValueError When no builder has been constructed yet for the given h5py object

Parameters *h5obj* (*Dataset* or *Group*) – the HDF5 object to the corresponding Builder object for

get_container (*h5obj*)

Get the container for the corresponding h5py Group or Dataset

raises ValueError When no builder has been constructed yet for the given h5py object

Parameters *h5obj* (*Dataset* or *Group*) – the HDF5 object to the corresponding Container/Data object for

open ()

Open this HDMFIO object for writing of the builder

close ()

Close this HDMFIO object to further reading/writing

write_builder (*builder*, *link_data=True*, *exhaust_dci=True*)

Parameters

- **builder** (*GroupBuilder*) – the GroupBuilder object representing the HDF5 file
- **link_data** (*bool*) – If not specified otherwise link (True) or copy (False) HDF5 Datasets
- **exhaust_dci** (*bool*) – exhaust DataChunkIterators one at a time. If False, exhaust them concurrently

classmethod **get_type** (*data*)

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* or *Dataset*, indexed by attribute name

write_group (*parent, builder, exhaust_dci=True*)

Parameters

- **parent** (*Group*) – the parent HDF5 object
- **builder** (*GroupBuilder*) – the *GroupBuilder* to write
- **exhaust_dci** (*bool*) – exhaust *DataChunkIterators* one at a time. If *False*, exhaust them concurrently

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*

write_dataset (*parent, builder, link_data=True, exhaust_dci=True*)

Write a dataset to HDF5

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

Parameters

- **parent** (*Group*) – the parent HDF5 object
- **builder** (*DatasetBuilder*) – the *DatasetBuilder* to write
- **link_data** (*bool*) – If not specified otherwise link (*True*) or copy (*False*) HDF5 Datasets
- **exhaust_dci** (*bool*) – exhaust *DataChunkIterators* one at a time. If *False*, exhaust them concurrently

Returns the *Dataset* that was created

Return type *Dataset*

mode

Return the HDF5 file mode. One of (“w”, “r”, “r+”, “a”, “w-“, “x”).

Module contents

12.5.2 Submodules

hdmf.backends.io module

class `hdmf.backends.io.HDMFIO` (*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 HDMFIO 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, exhaust_dci=True*)

Parameters

- **container** (*Container*) – the Container object to write
- **exhaust_dci** (*bool*) – exhaust DataChunkIterators one at a time. If False, exhaust them concurrently

read_builder()

Read data and return the GroupBuilder representing

Returns a GroupBuilder representing the read data

Return type *GroupBuilder*

write_builder (*builder, exhaust_dci=True*)

Write a GroupBuilder representing an Container object

Parameters

- **builder** (*GroupBuilder*) – the GroupBuilder object representing the Container
- **exhaust_dci** (*bool*) – exhaust DataChunkIterators one at a time. If False, exhaust them concurrently

open()

Open this HDMFIO object for writing of the builder

close()

Close this HDMFIO object to further reading/writing

exception `hdmf.backends.io.UnsupportedOperation`

Bases: `ValueError`

hdmf.backends.warnings module

exception `hdmf.backends.warnings.BrokenLinkWarning`

Bases: `UserWarning`

Raised when a group has a key with a None value.

12.5.3 Module contents

12.6 hdmf.data_utils module

class `hdmf.data_utils.AbstractDataChunkIterator`

Bases: `object`

Abstract iterator class used to iterate over DataChunks.

Derived classes must ensure that all abstract methods and abstract properties are implemented, in particular, `dtype`, `maxshape`, `__iter__`, `__next__`, `recommended_chunk_shape`, and `recommended_data_shape`.

`__iter__` ()

Return the iterator object

`__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 `DataChunk.selection` of the current chunk

Returns `DataChunk` object with the data and selection of the current chunk

Return type `DataChunk`

recommended_chunk_shape ()

Recommend the chunk shape for the data array.

Returns NumPy-style shape tuple describing the recommended shape for the chunks of the target array or `None`. This may or may not be the same as the shape of the chunks returned in the iteration process.

recommended_data_shape ()

Recommend the initial shape for the data array.

This is useful in particular to avoid repeated resized of the target array when reading from this data iterator. This should typically be either the final size of the array or the known minimal shape of the array.

Returns NumPy-style shape tuple indicating the recommended initial shape for the target array. This may or may not be the final full shape of the array, i.e., the array is allowed to grow. This should not be `None`.

dtype

Define the data type of the array

Returns NumPy style dtype or otherwise compliant dtype string

maxshape

Property describing the maximum shape of the data array that is being iterated over

Returns NumPy-style shape tuple indicating the maximum dimensions up to which the dataset may be resized. Axes with `None` are unlimited.

class `hdmf.data_utils.DataChunkIterator` (*data=None, maxshape=None, dtype=None, buffer_size=1, iter_axis=0*)

Bases: `hdmf.data_utils.AbstractDataChunkIterator`

Custom iterator class used to iterate over chunks of data.

This default implementation of `AbstractDataChunkIterator` accepts any iterable and assumes that we iterate over a single dimension of the data array (default: the first dimension). `DataChunkIterator` 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.

Initialize the DataChunkIterator. If ‘data’ is an iterator and ‘dtype’ is not specified, then next is called on the iterator in order to determine the dtype of the data.

Parameters

- **data** (*None*) – The data object used for iteration
- **maxshape** (*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
- **iter_axis** (*int*) – The dimension to iterate over

classmethod from_iterable (*data=None, maxshape=None, dtype=None, buffer_size=1, iter_axis=0*)

Parameters

- **data** (*None*) – The data object used for iteration
- **maxshape** (*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
- **iter_axis** (*int*) – The dimension to iterate over

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 DataChunk.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 an initial size for the dataset

maxshape

Get a shape tuple describing the maximum shape of the array described by this DataChunkIterator. If an iterator is provided and no data has been read yet, then the first chunk will be read (i.e., next will be called on the iterator) in order to determine the maxshape.

Returns Shape tuple. None is used for dimensions where the maximum shape is not known or unlimited.

dtype

Get the value data type

Returns np.dtype object describing the datatype

class hdmf.data_utils.DataChunk (*data=None, selection=None*)

Bases: object

Class used to describe a data chunk. Used in DataChunkIterator.

Parameters

- **data** (ndarray) – Numpy array with the data value(s) of the chunk
- **selection** (None) – Numpy index tuple describing the location of the chunk

astype (*dtype*)

Get a new DataChunk with the self.data converted to the given type

dtype

Data type of the values in the chunk

Returns np.dtype of the values in the DataChunk

hdmf.data_utils.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 hdmf.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 validation 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_DIMS_ERROR': 'Unequal number of axes'}`
 Dict where the Keys are the type of errors that may have occurred during shape comparison and the values are strings with default error messages for the type.

`class hdmf.data_utils.DataIO (data=None)`

Bases: `object`

Base class for wrapping data arrays for I/O. Derived classes of `DataIO` are typically used to pass dataset-specific I/O parameters to the particular HDMFIO backend.

Parameters `data` (`ndarray` or `list` or `tuple` or `Dataset` or `HDMFDataset` or `AbstractDataChunkIterator`) – the data to be written

`get_io_params ()`

Returns a dict with the I/O parameters specified in this `DataIO`.

`data`

Get the wrapped data object

`__getitem__ (item)`

Delegate slicing to the data object

`valid`

bool indicating if the data object is valid

`exception hdmf.data_utils.InvalidDataIOError`

Bases: `Exception`

12.7 hdmf.utils module

`class hdmf.utils.AllowPositional`

Bases: `enum.Enum`

An enumeration.

`ALLOWED = 1`

`ERROR = 3`

`WARNING = 2`

`hdmf.utils.docval_macro` (*macro*)

Class decorator to add the class to a list of types associated with the key `macro` in the `__macros` dict

`hdmf.utils.get_docval` (*func*, **args*)

Get a copy of docval arguments for a function. If args are supplied, return only docval arguments with value for 'name' key equal to the args

`hdmf.utils.fmt_docval_args` (*func*, *kwargs*)

Separate positional and keyword arguments

Useful for methods that wrap other methods

`hdmf.utils.call_docval_func` (*func*, *kwargs*)

`hdmf.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 dimensions of an input array add the optional 'shape' 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_shape** – Enforce the dimensions of input arrays (Default=True)
- **validator** – dict objects specifying the method parameters
- **options** – additional options for documenting and validating method parameters

`hdmf.utils.getargs` (**argnames*, *argdict*)

Convenience function to retrieve arguments from a dictionary in batch

`hdmf.utils.popargs (*argnames, argdict)`

Convenience function to retrieve and remove arguments from a dictionary in batch

class `hdmf.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)

`hdmf.utils.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 determine 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. Dimensions for which the size is unknown will be set to None.

`hdmf.utils.pystr` (*s*)

Convert a string of characters to Python str object

class `hdmf.utils.LabelledDict` (*label, key_attr='name'*)

Bases: `dict`

A dict wrapper class with a label and which allows retrieval of values based on an attribute of the values

For example, if the key attribute is set as 'name' in `__init__`, then all objects added to the `LabelledDict` must have a 'name' attribute and a particular object in the `LabelledDict` can be accessed using the syntax [`object_name`] if the `object.name == 'object_name'`. In this way, `LabelledDict` acts like a set where values can be retrieved using square brackets around the value of the key attribute. An 'add' method makes clear the association between the key attribute of the `LabelledDict` and the values of the `LabelledDict`.

`LabelledDict` also supports retrieval of values with the syntax `my_dict['attr == val']`, which returns a set of objects in the `LabelledDict` which have an attribute 'attr' with a string value 'val'. If no objects match that condition, a `KeyError` is raised. Note that if 'attr' equals the key attribute, then the single matching value is returned, not a set.

Usage: `LabelledDict(label='my_objects', def_key_name = 'name') my_dict[obj.name] = obj my_dict.add(obj)`
simpler syntax

Example

```
# MyTestClass is a class with attributes 'prop1' and 'prop2'. MyTestClass.__init__ sets those attributes. ld =
LabelledDict(label='all_objects', key_attr='prop1') obj1 = MyTestClass('a', 'b') obj2 = MyTestClass('d', 'b')
ld[obj1.prop1] = obj1 # obj1 is added to the LabelledDict with the key obj1.prop1. Any other key is not allowed.
ld.add(obj2) # Simpler 'add' syntax enforces the required relationship ld['a'] # Returns obj1 ld['prop1 == a'] #
Also returns obj1 ld['prop2 == b'] # Returns set([obj1, obj2]) - the set of all values v in ld where v.prop2 == 'b'
```

Parameters

- **label** (*str*) – the label on this dictionary
- **key_attr** (*str*) – the attribute name to use as the key

label

Return the label of this LabelledDict

key_attr

Return the attribute used as the key for values in this LabelledDict

__getitem__ (*args*)

Get a value from the LabelledDict with the given key.

Supports syntax `my_dict['attr == val']`, which returns a set of objects in the LabelledDict which have an attribute 'attr' with a string value 'val'. If no objects match that condition, a `KeyError` is raised.

Note that if 'attr' equals the key attribute, then the single matching value is returned, not a set.

add (*value*)

Add a value to the dict with the key `value.key_attr`.

Raises `ValueError` if value does not have attribute `key_attr`.

12.8 hdmf.validate package

12.8.1 Submodules

hdmf.validate.errors module

```
class hdmf.validate.errors.Error (name, reason, location=None)
```

Bases: `object`

Parameters

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

name

reason

location

```
class hdmf.validate.errors.DTypeError (name, expected, received, location=None)
```

Bases: `hdmf.validate.errors.Error`

Parameters

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

- **expected** (*dtype* or *type* or *str* or *list*) – the expected dtype
- **received** (*dtype* or *type* or *str* or *list*) – the received dtype
- **location** (*str*) – the location of the error

class `hdmf.validate.errors.MissingError` (*name*, *location=None*)

Bases: `hdmf.validate.errors.Error`

Parameters

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

class `hdmf.validate.errors.MissingDataType` (*name*, *data_type*, *location=None*)

Bases: `hdmf.validate.errors.Error`

Parameters

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

data_type

class `hdmf.validate.errors.ExpectedArrayError` (*name*, *expected*, *received*, *location=None*)

Bases: `hdmf.validate.errors.Error`

Parameters

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

class `hdmf.validate.errors.ShapeError` (*name*, *expected*, *received*, *location=None*)

Bases: `hdmf.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
- **location** (*str*) – the location of the error

class `hdmf.validate.errors.IllegalLinkError` (*name*, *location=None*)

Bases: `hdmf.validate.errors.Error`

A validation error for indicating that a link was used where an actual object (i.e. a dataset or a group) must be used

Parameters

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

class `hdmf.validate.errors.IncorrectDataType` (*name, expected, received, location=None*)
Bases: `hdmf.validate.errors.Error`

A validation error for indicating that the incorrect `data_type` (not `dtype`) was used.

Parameters

- **name** (*str*) – the name of the component that is erroneous
- **expected** (*str*) – the expected `data_type`
- **received** (*str*) – the received `data_type`
- **location** (*str*) – the location of the error

hdmf.validate.validator module

`hdmf.validate.validator.check_type` (*expected, received*)
expected should come from the spec *received* should come from the data

`hdmf.validate.validator.get_iso8601_regex` ()

`hdmf.validate.validator.get_type` (*data*)

`hdmf.validate.validator.check_shape` (*expected, received*)

class `hdmf.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

namespace

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

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 `hdmf.validate.validator.Validator` (*spec, validator_map*)

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
- **validator_map** (*ValidatorMap*) – the ValidatorMap to use during validation

spec

vmap

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 `hdmf.validate.validator.AttributeValidator` (*spec, validator_map*)

Bases: `hdmf.validate.validator.Validator`

A class for validating values against AttributeSpecs

Parameters

- **spec** (*AttributeSpec*) – the specification to use to validate
- **validator_map** (*ValidatorMap*) – the ValidatorMap to use during validation

validate (*value*)

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

Returns a list of Errors

Return type `list`

class `hdmf.validate.validator.BaseStorageValidator` (*spec, validator_map*)

Bases: `hdmf.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
- **validator_map** (*ValidatorMap*) – the ValidatorMap to use during validation

validate (*builder*)

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

Returns a list of Errors

Return type `list`

class `hdmf.validate.validator.DatasetValidator` (*spec, validator_map*)

Bases: `hdmf.validate.validator.BaseStorageValidator`

A class for validating DatasetBuilders against DatasetSpecs

Parameters

- **spec** (*DatasetSpec*) – the specification to use to validate
- **validator_map** (*ValidatorMap*) – the ValidatorMap to use during validation

validate (*builder*)**Parameters** **builder** (*DatasetBuilder*) – the builder to validate**Returns** a list of Errors**Return type** *list***class** `hdmf.validate.validator.GroupValidator` (*spec, validator_map*)Bases: `hdmf.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*

12.8.2 Module contents

12.9 hdmf.testing package

12.9.1 Submodules

`hdmf.testing.testcase` module

class `hdmf.testing.testcase.TestCase` (*methodName='runTest'*)Bases: `unittest.case.TestCase`Extension of `unittest`'s `TestCase` to add useful functions for unit testing in HDMF.Create an instance of the class that will use the named test method when executed. Raises a `ValueError` if the instance does not have a method with the specified name.**assertRaisesWith** (*exc_type, exc_msg, *args, **kwargs*)Asserts the given invocation raises the expected exception. This is similar to `unittest`'s `assertRaises` and `assertRaisesRegex`, but checks for an exact match.**assertWarnsWith** (*warn_type, exc_msg, *args, **kwargs*)Asserts the given invocation raises the expected warning. This is similar to `unittest`'s `assertWarns` and `assertWarnsRegex`, but checks for an exact match.**assertContainerEqual** (*container1, container2, ignore_name=False, ignore_hdmf_attrs=False*)

Asserts that the two containers have equal contents.

`ignore_name` - whether to ignore testing equality of name `ignore_hdmf_attrs` - whether to ignore testing equality of HDMF container attributes, such as `container_source` and `object_id`


```
class hdmf.testing.testcase.H5RoundTripMixin
```

```
Bases: object
```

Mixin class for methods to run a roundtrip test writing a container to and reading the container from an HDF5 file. The `setUp`, `test_roundtrip`, and `tearDown` methods will be run by unittest.

The abstract method `setUpContainer` needs to be implemented by classes that include this mixin.

Example:

```
class TestMyContainerRoundTrip(H5RoundTripMixin, TestCase):
    def setUpContainer(self):
        # return the Container to read/write
```

NOTE: This class is a mix-in and not a subclass of `TestCase` so that unittest does not discover it, try to run it, and skip it.

```
setUp ()
```

```
tearDown ()
```

```
manager
```

```
The build manager
```

```
setUpContainer ()
```

```
Should return the Container to read/write
```

```
test_roundtrip ()
```

```
Test whether the test Container read from file has the same contents the original Container and validate the file
```

```
roundtripContainer (cache_spec=False)
```

```
Write just the Container to an HDF5 file, read the container from the file, and return it
```

```
validate ()
```

```
Validate the created file
```

12.9.2 Module contents

12.10 hdmf package

12.10.1 Subpackages

12.10.2 Submodules

hdmf.array module

```
class hdmf.array.Array (data)
```

```
Bases: object
```

```
data
```

```
get_data ()
```

```
__getitem__ (arg)
```

```
class hdmf.array.AbstractSortedArray (data)
```

```
Bases: hdmf.array.Array
```

An abstract class for representing sorted array

find_point (*val*)

get_data ()

class `hdmf.array.SortedArray` (*array*)

Bases: `hdmf.array.AbstractSortedArray`

A class for wrapping sorted arrays. This class overrides `<`, `>`, `<=`, `>=`, `==`, and `!=` to leverage the sorted content for efficiency.

find_point (*val*)

class `hdmf.array.LinSpace` (*start*, *stop*, *step*)

Bases: `hdmf.array.SortedArray`

find_point (*val*)

hdmf.monitor module

exception `hdmf.monitor.NotYetExhausted`

Bases: `Exception`

class `hdmf.monitor.DataChunkProcessor` (*data*)

Bases: `hdmf.data_utils.AbstractDataChunkIterator`

Initialize the `DataChunkIterator`

Parameters *data* (`DataChunkIterator`) – the `DataChunkIterator` to analyze

recommended_chunk_shape ()

Recommend the chunk shape for the data array.

Returns NumPy-style shape tuple describing the recommended shape for the chunks of the target array or None. This may or may not be the same as the shape of the chunks returned in the iteration process.

recommended_data_shape ()

Recommend the initial shape for the data array.

This is useful in particular to avoid repeated resized of the target array when reading from this data iterator. This should typically be either the final size of the array or the known minimal shape of the array.

Returns NumPy-style shape tuple indicating the recommended initial shape for the target array. This may or may not be the final full shape of the array, i.e., the array is allowed to grow. This should not be None.

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 `NotYetExhausted` exception

class `hdmf.monitor.NumSampleCounter` (***kwargs*)

Bases: `hdmf.monitor.DataChunkProcessor`

process_data_chunk (*data_chunk*)

Parameters **data_chunk** (*DataChunk*) – a chunk to process

compute_final_result ()

hdmf.query module

class `hdmf.query.Query` (*obj, op, arg*)

Bases: `object`

evaluate (*expand=True*)

Parameters **expand** (`bool`) – None

class `hdmf.query.HDMFDataset` (*dataset*)

Bases: `object`

Parameters **dataset** (`ndarray` or `list` or `tuple` or `Dataset` or `Array`) – the HDF5 file lazily evaluate

__getitem__ (*key*)

dataset

dtype

next ()

class `hdmf.query.ReferenceResolver`

Bases: `object`

A base class for classes that resolve references

classmethod **get_inverse_class** ()

Return the class the represents the ReferenceResolver that resolves refernces to the opposite type.

`BuilderResolver.get_inverse_class` should return a class that subclasses `ContainerResolver`.

`ContainerResolver.get_inverse_class` should return a class that subclasses `BuilderResolver`.

invert ()

Return an object that defers reference resolution but in the opposite direction.

class `hdmf.query.BuilderResolver`

Bases: `hdmf.query.ReferenceResolver`

A reference resolver that resolves references to Builders

Subclasses should implement the `invert` method and the `get_inverse_class` classmethod

`BuilderResolver.get_inverse_class` should return a class that subclasses `ContainerResolver`.

class `hdmf.query.ContainerResolver`

Bases: `hdmf.query.ReferenceResolver`

A reference resolver that resolves references to Containers

Subclasses should implement the `invert` method and the `get_inverse_class` classmethod

`ContainerResolver.get_inverse_class` should return a class that subclasses `BuilderResolver`.

hdmf.region module

class `hdmf.region.RegionSlicer` (*target, slice*)

Bases: `hdmf.container.DataRegion`

A abstract base class to control getting using a region

Subclasses must implement `__getitem__` and `__len__`

Parameters

- **target** (*None*) – the target to slice
- **slice** (*None*) – the region to slice

data

The target data. Same as `self.target`

region

The selected region. Same as `self.slice`

target

The target data

slice

The selected slice

`__getitem__`

Must be implemented by subclasses

`__len__`

Must be implemented by subclasses

class `hdmf.region.ListSlicer` (*dataset, region*)

Bases: `hdmf.region.RegionSlicer`

Implementation of `RegionSlicer` for slicing Lists and Data

Parameters

- **dataset** (*list* or *tuple* or *Data*) – the dataset to slice
- **region** (*list* or *tuple* or *slice*) – the region reference to use to slice

`__getitem__` (*idx*)

Get data values from selected data

12.10.3 Module contents

`hdmf.get_region_slicer` (*dataset, region*)

Parameters

- **dataset** (*None*) – the HDF5 dataset to slice
- **region** (*None*) – the region reference to use to slice

modindex

13.1 Continuous Integration

HDMF is tested against Ubuntu, macOS, and Windows operating systems. The project has both unit and integration tests.

- [CircleCI](#) runs all HDMF tests on Ubuntu
- [Azure Pipelines](#) runs all HDMF tests on Windows and macOS

Each time a PR is published or updated, the project is built, packaged, and tested on all supported 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 4 kinds of requirements specification in HDMF.

The first one is the [requirements-min.txt](#) file, which lists the package dependencies and their minimum versions for installing HDMF. These dependencies are read by [setup.py](#) into the *install_requires* key, with the adjustment that the '==' listed in [requirements-min.txt](#) are replaced with '>=' to reflect that they are minimum versions.

The second one is [requirements.txt](#) which contain a list of pinned (concrete) dependencies to reproduce an entire development environment to use HDMF.

The third one is `requirements-dev.txt` which contain a list of pinned (concrete) dependencies to reproduce an entire development environment to use HDMF, run HDMF tests, check code style, compute coverage, and create test environments.

The final one is `requirements-doc.txt` which contain a list of dependencies to generate the documentation for HDMF. Both this file and `requirements.txt` are used by `ReadTheDocs` to initialize the local environment for Sphinx to run.

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

HDMF 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 HDMF's case) to generate a release name. After all the tests pass, CircleCI creates both a wheel (*.whl) and source distribution (*.tar.gz) for 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.

It is important to note that GitHub automatically generates source code archives in .zip and .tar.gz formats and attaches those files to all releases as an asset. These files currently do not contain the submodules within HDMF and thus do not serve as a complete installation. For a complete source code archive, use the source distribution generated by CircleCI, typically named `hdmf-{version}.tar.gz`.

How to Make a Roundtrip Test

The HDMF test suite has tools for easily doing round-trip tests of container classes. These tools exist in the `hdmf.testing` module. Round-trip tests exist for the container classes in the `hdmf.common` module. We recommend you write any additional round-trip tests in the `tests/unit/common` subdirectory of the Git repository.

For executing your new tests, we recommend using the `test.py` script in the top of the Git repository. Roundtrip tests will get executed as part of the full test suite, which can be executed with the following command:

```
$ python test.py
```

The roundtrip test will generate a new HDMF file with the name `test_<CLASS_NAME>.h5` where `CLASS_NAME` is the class name of the container class you are roundtripping. The test will write an HDMF file with an instance of the container to disk, read this instance back in, and compare it to the instance that was used for writing to disk. Once the test is complete, the HDMF file will be deleted. You can keep the HDMF file around after the test completes by setting the environment variable `CLEAN_HDMF` to `0`, `false`, `False`, or `FALSE`. Setting `CLEAN_HDMF` to any value not listed here will cause the roundtrip HDMF file to be deleted once the test has completed.

Before writing tests, we also suggest you familiarize yourself with the *software architecture* of HDMF.

14.1 H5RoundTripMixin

To write a roundtrip test, you will need to subclass the `H5RoundTripMixin` class and the `TestCase` class, in that order, and override some of the instance methods of the `H5RoundTripMixin` class to test the process of going from in-memory Python object to data stored on disk and back.

14.1.1 setUpContainer

To configure the test for a particular container class, you need to override the `setUpContainer` method. This method should take no arguments, and return an instance of the container class you are testing.

Here is an example using a `CSRMatrix`:

```
from hdmf.common import CSRMatrix
from hdmf.testing import TestCase, H5RoundTripMixin
import numpy as np

class TestCSRMatrixRoundTrip(H5RoundTripMixin, TestCase):

    def setUpContainer(self):
        data = np.array([1, 2, 3, 4, 5, 6])
        indices = np.array([0, 2, 2, 0, 1, 2])
        indptr = np.array([0, 2, 3, 6])
        return CSRMatrix(data, indices, indptr, (3, 3))
```

How to Make a Release

A core developer should use the following steps to create a release *X.Y.Z* of **hdmf**.

Note: Since the hdmf 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.

15.1 Prerequisites

- All CI tests are passing on [CircleCI](#) and [Azure Pipelines](#).
- You have a [GPG signing key](#).
- Dependency versions in `requirements.txt`, `requirements-dev.txt`, `requirements-doc.txt`, `requirements-min.txt` are up-to-date.
- Legal information and copyright dates are up-to-date in `Legal.txt`, `license.txt`, `README.rst`, `docs/source/conf.py`, and any other files.
- Package information in `setup.py` is up-to-date.
- `README.rst` information is up-to-date.
- The `hdmf-common-schema` submodule is up-to-date. The version number should be checked manually in case syncing the git submodule does not work as expected.
- Documentation reflects any new features and changes in HDMF functionality.
- Documentation builds locally.
- Documentation builds on ReadTheDocs on the 'latest' build.
- Release notes have been prepared.

- An appropriate new version number has been selected.

15.2 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.

15.3 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.

15.4 PyPI: Step-by-step

1. Make sure that all CI tests are passing on [CircleCI](#) and [Azure Pipelines](#).
2. List all tags sorted by version

```
$ git tag -l | sort -V
```

3. Choose the next release version number

```
$ release=X.Y.Z
```

Warning: To ensure the packages are uploaded on [PyPI](#), tags must match this regular expression:
`^[0-9]+(\.[0-9]+)*(\.post[0-9]+)?$.`

4. Download latest sources

```
$ cd /tmp && git clone --recurse-submodules git@github.com:hdmf-dev/hdmf && \
↳ cd hdmf
```

5. Tag the release

```
$ git tag --sign -m "hdmf ${release}" ${release} origin/dev
```

Warning: This step requires a [GPG signing key](#).

6. Publish the release tag

```
$ git push origin ${release}
```

Important: This will trigger builds on each CI services and automatically upload the wheels and source distribution on [PyPI](#).

7. Check the status of the builds on [CircleCI](#) and [Azure Pipelines](#).8. Once the builds are completed, check that the distributions are available on [PyPI](#) and that a new [GitHub release](#) was created.

9. Create a clean testing environment to test the installation

```
$ mkvirtualenv hdmf-${release}-install-test && \
  pip install hdmf && \
  python -c "import hdmf; print(hdmf.__version__)"
```

Note: If the `mkvirtualenv` command is not available, this means you do not have [virtualenvwrapper](#) installed, in that case, you could either install it or directly use [virtualenv](#) or [venv](#).

10. Cleanup

```
$ deactivate && \
  rm -rf dist/* && \
  rmvirtualenv hdmf-${release}-install-test
```

15.5 Conda: Step-by-step

Warning: Publishing on conda requires you to have corresponding package version uploaded on [PyPI](#). So you have to do the PyPI and Github release before you do the conda release.

In order to release a new version on conda-forge, follow the steps below:

1. Choose the next release version number (that matches with the pypi version that you already published)

```
$ release=X.Y.Z
```

2. Fork hdmf-feedstock

First step is to fork [hdmf-feedstock](#) repository. This is the recommended [best practice](#) by conda.

3. Clone forked feedstock

Fill the `YOURGITHUBUSER` part.

```
$ cd /tmp && git clone https://github.com/YOURGITHUBUSER/hdmf-feedstock.git
```

4. Download corresponding source for the release version

```
$ cd /tmp && \
  wget https://github.com/hdmf-dev/hdmf/releases/download/$release/hdmf-
  ↳$release.tar.gz
```

5. Create a new branch

```
$ cd hdmf-feedstock && \
  git checkout -b $release
```

6. Modify `meta.yaml`

Update the [version string](#) and [sha256](#).

We have to modify the sha and the version string in the `meta.yaml` file.

For linux flavors:

```
$ sed -i "2s/.*/{% set version = \"\$release\" %}/" recipe/meta.yaml
$ sha=$(openssl sha256 /tmp/hdmf-$release.tar.gz | awk '{print $2}')
$ sed -i "3s/.*/{% set sha256 = \"\$sha\" %}/" recipe/meta.yaml
```

For macOS:

```
$ sed -i -- "2s/.*/{% set version = \"\$release\" %}/" recipe/meta.yaml
$ sha=$(openssl sha256 /tmp/hdmf-$release.tar.gz | awk '{print $2}')
$ sed -i -- "3s/.*/{% set sha256 = \"\$sha\" %}/" recipe/meta.yaml
```

If `requirements-min.txt` was changed, the changes should be reflected in the `requirements/run` list.

7. Push the changes

```
$ git push origin $release
```

8. Create a Pull Request

Create a pull request against the [main repository](#). If the tests pass, merge the PR, and a new release will be published on Anaconda cloud.

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
- requirements-min.txt

16.1 requirements.txt

requirements.txt of the project can be created or updated and then captured using the following script:

```
mkvirtualenv hdmf-requirements

cd hdmf
pip install .
pip check # check for package conflicts
pip freeze > requirements.txt

deactivate
rmvirtualenv hdmf-requirements
```

16.2 requirements-(dev|doc).txt

Any of these requirements files can be updated using the following scripts:

```
cd hdmf

# Set the requirements file to update
```

(continues on next page)

(continued from previous page)

```
target_requirements=requirements-dev.txt

mkvirtualenv hdmf-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>

# Check for any conflicts in installed packages
pip check

# Update list of pinned requirements
pip freeze > $target_requirements

deactivate
rmvirtualenv hdmf-requirements
```

16.3 requirements-min.txt

Minimum requirements should be updated manually if a new feature or bug fix is added in a dependency that is required for proper running of HDMF. Minimum requirements should also be updated if a user requests that HDMF be installable with an older version of a dependency, all tests pass using the older version, and there is no valid reason for the minimum version to be as high as it is.

CHAPTER 17

Copyright

“hdmf” Copyright (c) 2017-2020, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

If you have questions about your rights to use or distribute this software, please contact Berkeley Lab’s Innovation & Partnerships Office at IPO@lbl.gov.

NOTICE. This Software was developed under funding from the U.S. Department of Energy and the U.S. Government consequently retains certain rights. As such, the U.S. Government has been granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, distribute copies to the public, prepare derivative works, and perform publicly and display publicly, and to permit other to do so.

“hdmf” Copyright (c) 2017-2020, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- (1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- (3) Neither the name of the University of California, Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code (“Enhancements”) to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

CHAPTER 19

Indices and tables

- `genindex`
- `modindex`
- `search`

h

- hdmf, 96
- hdmf.array, 93
- hdmf.backends, 82
- hdmf.backends.hdf5, 80
- hdmf.backends.hdf5.h5_utils, 73
- hdmf.backends.hdf5.h5tools, 78
- hdmf.backends.io, 81
- hdmf.backends.warnings, 81
- hdmf.build, 57
- hdmf.build.builders, 45
- hdmf.build.manager, 49
- hdmf.build.map, 54
- hdmf.build.objectmapper, 54
- hdmf.build.warnings, 56
- hdmf.common, 42
- hdmf.common.io, 38
- hdmf.common.io.table, 37
- hdmf.common.sparse, 38
- hdmf.common.table, 38
- hdmf.container, 44
- hdmf.data_utils, 82
- hdmf.monitor, 94
- hdmf.query, 95
- hdmf.region, 96
- hdmf.spec, 73
- hdmf.spec.catalog, 57
- hdmf.spec.namespace, 59
- hdmf.spec.spec, 62
- hdmf.spec.write, 71
- hdmf.testing, 93
- hdmf.testing.testcase, 92
- hdmf.utils, 85
- hdmf.validate, 92
- hdmf.validate.errors, 88
- hdmf.validate.validator, 90

Symbols

- __getitem__ (*hdmf.region.RegionSlicer attribute*), 96
 __getitem__ () (*hdmf.array.Array method*), 93
 __getitem__ () (*hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset method*), 74
 __getitem__ () (*hdmf.backends.hdf5.h5_utils.AbstractH5RegionDataset method*), 74
 __getitem__ () (*hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset method*), 74
 __getitem__ () (*hdmf.backends.hdf5.h5_utils.H5RegionSlicer method*), 76
 __getitem__ () (*hdmf.build.builders.GroupBuilder method*), 48
 __getitem__ () (*hdmf.common.table.DynamicTable method*), 41
 __getitem__ () (*hdmf.common.table.DynamicTableRegion method*), 42
 __getitem__ () (*hdmf.common.table.VectorIndex method*), 39
 __getitem__ () (*hdmf.container.Data method*), 45
 __getitem__ () (*hdmf.data_utils.DataIO method*), 85
 __getitem__ () (*hdmf.query.HDMFDataset method*), 95
 __getitem__ () (*hdmf.region.ListSlicer method*), 96
 __getitem__ () (*hdmf.utils.LabelledDict method*), 88
 __iter__ () (*hdmf.data_utils.AbstractDataChunkIterator method*), 82
 __len__ (*hdmf.region.RegionSlicer attribute*), 96
 __next__ () (*hdmf.data_utils.AbstractDataChunkIterator method*), 82
- A**
- AbstractContainer (*class in hdmf.container*), 44
 AbstractDataChunkIterator (*class in hdmf.data_utils*), 82
 AbstractH5ReferenceDataset (*class in hdmf.backends.hdf5.h5_utils*), 74
 AbstractH5RegionDataset (*class in hdmf.backends.hdf5.h5_utils*), 74
 AbstractH5TableDataset (*class in hdmf.backends.hdf5.h5_utils*), 74
 AbstractSortedArray (*class in hdmf.array*), 93
 add () (*hdmf.utils.LabelledDict method*), 88
 add_attribute () (*hdmf.spec.spec.BaseStorageSpec method*), 65
 add_candidate () (*hdmf.build.manager.Proxy method*), 50
 add_child () (*hdmf.container.AbstractContainer method*), 44
 add_column () (*hdmf.common.table.DynamicTable method*), 40
 add_dataset () (*hdmf.build.builders.GroupBuilder method*), 47
 add_dataset () (*hdmf.spec.spec.GroupSpec method*), 69
 add_group () (*hdmf.build.builders.GroupBuilder method*), 47
 add_group () (*hdmf.spec.spec.GroupSpec method*), 69
 add_link () (*hdmf.build.builders.GroupBuilder method*), 47
 add_link () (*hdmf.spec.spec.GroupSpec method*), 70
 add_namespace () (*hdmf.spec.namespace.NamespaceCatalog method*), 61
 add_row () (*hdmf.common.table.DynamicTable method*), 40
 add_row () (*hdmf.common.table.VectorData method*), 39
 add_row () (*hdmf.common.table.VectorIndex method*), 39
 add_source () (*hdmf.spec.write.NamespaceBuilder method*), 72
 add_spec () (*hdmf.spec.write.NamespaceBuilder method*), 71
 add_spec () (*hdmf.spec.write.SpecFileBuilder method*), 72
 add_vector () (*hdmf.common.table.VectorIndex method*), 39
 ALLOWED (*hdmf.utils.AllowPositional attribute*), 85
 AllowPositional (*class in hdmf.utils*), 85

- append() (*hdmf.container.Data method*), 45
 Array (*class in hdmf.array*), 93
 assertContainerEqual() (*hdmf.testing.testcase.TestCase method*), 92
 assertEqualsShape() (*in module hdmf.data_utils*), 84
 assertRaisesWith() (*hdmf.testing.testcase.TestCase method*), 92
 assertValidDtype() (*hdmf.spec.spec.DtypeSpec static method*), 66
 assertWarnsWith() (*hdmf.testing.testcase.TestCase method*), 92
 astype() (*hdmf.data_utils.DataChunk method*), 84
 attr_columns() (*hdmf.common.io.table.DynamicTableMap method*), 37
 attributes (*hdmf.build.builders.BaseBuilder attribute*), 46
 attributes (*hdmf.spec.spec.BaseStorageSpec attribute*), 65
 AttributeSpec (*class in hdmf.spec.spec*), 63
 AttributeValidator (*class in hdmf.validate.validator*), 91
 author (*hdmf.spec.namespace.SpecNamespace attribute*), 59
 auto_register() (*hdmf.spec.catalog.SpecCatalog method*), 57
 available_namespaces() (*in module hdmf.common*), 43
- ## B
- BaseBuilder (*class in hdmf.build.builders*), 45
 BaseStorageSpec (*class in hdmf.spec.spec*), 64
 BaseStorageValidator (*class in hdmf.validate.validator*), 91
 BrokenLinkWarning, 81
 build() (*hdmf.build.manager.BuildManager method*), 50
 build() (*hdmf.build.manager.TypeMap method*), 53
 build() (*hdmf.build.objectmapper.ObjectMapper method*), 56
 build_const_args() (*hdmf.spec.spec.AttributeSpec class method*), 64
 build_const_args() (*hdmf.spec.spec.BaseStorageSpec class method*), 66
 build_const_args() (*hdmf.spec.spec.ConstructableDict class method*), 62
 build_const_args() (*hdmf.spec.spec.DatasetSpec class method*), 67
 build_const_args() (*hdmf.spec.spec.DtypeSpec class method*), 66
 build_const_args() (*hdmf.spec.spec.GroupSpec class method*), 71
 build_const_args() (*hdmf.spec.spec.Spec class method*), 63
 build_namespace() (*hdmf.spec.namespace.SpecNamespace class method*), 60
 build_spec() (*hdmf.spec.spec.ConstructableDict class method*), 62
 Builder (*class in hdmf.build.builders*), 45
 builder (*hdmf.build.builders.LinkBuilder attribute*), 49
 builder (*hdmf.build.builders.ReferenceBuilder attribute*), 49
 BuilderH5ReferenceDataset (*class in hdmf.backends.hdf5.h5_utils*), 75
 BuilderH5RegionDataset (*class in hdmf.backends.hdf5.h5_utils*), 76
 BuilderH5TableDataset (*class in hdmf.backends.hdf5.h5_utils*), 75
 BuilderResolver (*class in hdmf.query*), 95
 BuilderResolverMixin (*class in hdmf.backends.hdf5.h5_utils*), 73
 BuildManager (*class in hdmf.build.manager*), 50
- ## C
- call_docval_func() (*in module hdmf.utils*), 86
 catalog (*hdmf.spec.namespace.SpecNamespace attribute*), 60
 check_shape() (*in module hdmf.validate.validator*), 90
 check_type() (*in module hdmf.validate.validator*), 90
 children (*hdmf.container.AbstractContainer attribute*), 44
 chunks (*hdmf.build.builders.DatasetBuilder attribute*), 48
 close() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
 close() (*hdmf.backends.io.HDMFIO method*), 81
 colnames (*hdmf.common.table.DynamicTable attribute*), 40
 columns (*hdmf.common.table.DynamicTable attribute*), 40
 comm (*hdmf.backends.hdf5.h5tools.HDF5IO attribute*), 78
 compute_final_result() (*hdmf.monitor.DataChunkProcessor method*), 94
 compute_final_result() (*hdmf.monitor.NumSampleCounter method*), 95

`construct ()` (*hdmf.build.manager.BuildManager method*), 50
`construct ()` (*hdmf.build.manager.TypeMap method*), 53
`construct ()` (*hdmf.build.objectmapper.ObjectMapper method*), 56
`ConstructableDict` (*class in hdmf.spec.spec*), 62
`constructor_arg ()` (*hdmf.build.objectmapper.ObjectMapper static method*), 54
`constructor_args` (*hdmf.build.objectmapper.ObjectMapper attribute*), 56
`constructor_args` (*hdmf.common.io.table.DynamicTableMap attribute*), 37
`contact` (*hdmf.spec.namespace.SpecNamespace attribute*), 59
`Container` (*class in hdmf.container*), 44
`container_source` (*hdmf.container.AbstractContainer attribute*), 44
`ContainerH5ReferenceDataset` (*class in hdmf.backends.hdf5.h5_utils*), 75
`ContainerH5RegionDataset` (*class in hdmf.backends.hdf5.h5_utils*), 75
`ContainerH5TableDataset` (*class in hdmf.backends.hdf5.h5_utils*), 74
`ContainerResolver` (*class in hdmf.query*), 95
`ContainerResolverMixin` (*class in hdmf.backends.hdf5.h5_utils*), 73
`convert_dt_name ()` (*hdmf.build.objectmapper.ObjectMapper class method*), 54
`convert_dtype ()` (*hdmf.build.objectmapper.ObjectMapper class method*), 54
`copy ()` (*hdmf.common.table.DynamicTable method*), 41
`copy_file ()` (*hdmf.backends.hdf5.h5tools.HDF5IO class method*), 78
`copy_mappers ()` (*hdmf.build.manager.TypeMap method*), 51
`create_region ()` (*hdmf.common.table.DynamicTable method*), 41
`CSRMatrix` (*class in hdmf.common.sparse*), 38

D

`Data` (*class in hdmf.container*), 44
`data` (*hdmf.array.Array attribute*), 93
`data` (*hdmf.build.builders.DatasetBuilder attribute*), 48
`data` (*hdmf.container.Data attribute*), 44
`data` (*hdmf.container.DataRegion attribute*), 45
`data` (*hdmf.data_utils.DataIO attribute*), 85
`data` (*hdmf.region.RegionSlicer attribute*), 96
`data_type` (*hdmf.build.manager.Proxy attribute*), 50
`data_type` (*hdmf.build.manager.TypeSource attribute*), 51
`data_type` (*hdmf.common.sparse.CSRMatrix attribute*), 38
`data_type` (*hdmf.common.table.DynamicTable attribute*), 41
`data_type` (*hdmf.common.table.DynamicTableRegion attribute*), 42
`data_type` (*hdmf.common.table.ElementIdentifiers attribute*), 39
`data_type` (*hdmf.common.table.Index attribute*), 38
`data_type` (*hdmf.common.table.VectorData attribute*), 39
`data_type` (*hdmf.common.table.VectorIndex attribute*), 39
`data_type` (*hdmf.container.Container attribute*), 44
`data_type` (*hdmf.container.Data attribute*), 45
`data_type` (*hdmf.validate.errors.MissingDataType attribute*), 89
`data_type_def` (*hdmf.spec.spec.BaseStorageSpec attribute*), 65
`data_type_inc` (*hdmf.spec.spec.BaseStorageSpec attribute*), 65
`data_type_inc` (*hdmf.spec.spec.LinkSpec attribute*), 67
`DataChunk` (*class in hdmf.data_utils*), 84
`DataChunkIterator` (*class in hdmf.data_utils*), 82
`DataChunkProcessor` (*class in hdmf.monitor*), 94
`DataIO` (*class in hdmf.data_utils*), 85
`DataRegion` (*class in hdmf.container*), 45
`dataset` (*hdmf.query.HDMFDataset attribute*), 95
`dataset_spec_cls` (*hdmf.spec.namespace.NamespaceCatalog attribute*), 61
`dataset_spec_cls ()` (*hdmf.spec.spec.GroupSpec class method*), 70
`DatasetBuilder` (*class in hdmf.build.builders*), 48
`DatasetOfReferences` (*class in hdmf.backends.hdf5.h5_utils*), 73
`datasets` (*hdmf.build.builders.GroupBuilder attribute*), 46
`datasets` (*hdmf.spec.spec.GroupSpec attribute*), 69
`DatasetSpec` (*class in hdmf.spec.spec*), 66
`DatasetValidator` (*class in hdmf.validate.validator*), 91
`date` (*hdmf.spec.namespace.SpecNamespace attribute*), 59
`deep_update ()` (*hdmf.build.builders.BaseBuilder method*), 46
`deep_update ()` (*hdmf.build.builders.DatasetBuilder method*), 49
`deep_update ()` (*hdmf.build.builders.GroupBuilder method*), 48
`def_key ()` (*hdmf.spec.spec.BaseStorageSpec class method*), 65
`default_name` (*hdmf.spec.spec.BaseStorageSpec attribute*), 64

- default_value (*hdmf.spec.spec.AttributeSpec attribute*), 63
- default_value (*hdmf.spec.spec.DatasetSpec attribute*), 67
- description (*hdmf.common.table.DynamicTable attribute*), 40
- description (*hdmf.common.table.VectorData attribute*), 39
- dims (*hdmf.spec.spec.AttributeSpec attribute*), 64
- dims (*hdmf.spec.spec.DatasetSpec attribute*), 67
- doc (*hdmf.spec.namespace.SpecNamespace attribute*), 59
- doc (*hdmf.spec.spec.DtypeSpec attribute*), 66
- doc (*hdmf.spec.spec.Spec attribute*), 63
- docval () (*in module hdmf.utils*), 86
- docval_macro () (*in module hdmf.utils*), 85
- dtype (*hdmf.backends.hdf5.h5_utils.AbstractH5ReferenceDataset class method*), 74
- dtype (*hdmf.backends.hdf5.h5_utils.AbstractH5RegionDataset class method*), 74
- dtype (*hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset attribute*), 74
- dtype (*hdmf.build.builders.DatasetBuilder attribute*), 49
- dtype (*hdmf.data_utils.AbstractDataChunkIterator attribute*), 82
- dtype (*hdmf.data_utils.DataChunk attribute*), 84
- dtype (*hdmf.data_utils.DataChunkIterator attribute*), 83
- dtype (*hdmf.query.HDMFDataset attribute*), 95
- dtype (*hdmf.spec.spec.AttributeSpec attribute*), 63
- dtype (*hdmf.spec.spec.DatasetSpec attribute*), 67
- dtype (*hdmf.spec.spec.DtypeSpec attribute*), 66
- dtype_spec_cls () (*hdmf.spec.spec.DatasetSpec class method*), 67
- DtypeError (*class in hdmf.validate.errors*), 88
- DtypeHelper (*class in hdmf.spec.spec*), 62
- DtypeSpec (*class in hdmf.spec.spec*), 66
- DynamicTable (*class in hdmf.common.table*), 39
- DynamicTableMap (*class in hdmf.common.io.table*), 37
- DynamicTableRegion (*class in hdmf.common.table*), 41
- ## E
- ElementIdentifiers (*class in hdmf.common.table*), 39
- Error (*class in hdmf.validate.errors*), 88
- ERROR (*hdmf.utils.AllowPositional attribute*), 85
- evaluate () (*hdmf.query.Query method*), 95
- ExpectedArrayError (*class in hdmf.validate.errors*), 89
- export () (*hdmf.spec.write.NamespaceBuilder method*), 72
- export_spec () (*in module hdmf.spec.write*), 72
- extend () (*hdmf.container.Data method*), 45
- ExtenderMeta (*class in hdmf.utils*), 87
- ## F
- fields (*hdmf.container.AbstractContainer attribute*), 44
- filter_available () (*hdmf.backends.hdf5.h5_utils.H5DataIO static method*), 77
- find_point () (*hdmf.array.AbstractSortedArray method*), 94
- find_point () (*hdmf.array.LinSpace method*), 94
- find_point () (*hdmf.array.SortedArray method*), 94
- fmt_docval_args () (*in module hdmf.utils*), 86
- from_dataframe () (*hdmf.common.table.DynamicTable class method*), 41
- from_iterable () (*hdmf.data_utils.DataChunkIterator class method*), 83
- full_name (*hdmf.spec.namespace.SpecNamespace attribute*), 59
- ## G
- get () (*hdmf.build.builders.GroupBuilder method*), 48
- get () (*hdmf.common.table.DynamicTable method*), 41
- get_ancestor () (*hdmf.container.AbstractContainer method*), 44
- get_attr_names () (*hdmf.build.objectmapper.ObjectMapper class method*), 55
- get_attr_spec () (*hdmf.build.objectmapper.ObjectMapper method*), 55
- get_attr_value () (*hdmf.build.objectmapper.ObjectMapper method*), 55
- get_attr_value () (*hdmf.common.io.table.DynamicTableMap method*), 37
- get_attribute () (*hdmf.build.objectmapper.ObjectMapper method*), 55
- get_attribute () (*hdmf.spec.spec.BaseStorageSpec method*), 66
- get_builder () (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
- get_builder_dt () (*hdmf.build.manager.BuildManager method*), 51
- get_builder_dt () (*hdmf.build.manager.TypeMap method*), 52
- get_builder_loc () (*hdmf.validate.validator.Validator class method*), 91
- get_builder_name () (*hdmf.build.manager.BuildManager method*), 51
- get_builder_name () (*hdmf.build.manager.TypeMap method*), 53

`get_builder_name()` (*hdmf.build.objectmapper.ObjectMapper* method), 56
`get_builder_ns()` (*hdmf.build.manager.BuildManager* method), 51
`get_builder_ns()` (*hdmf.build.manager.TypeMap* method), 52
`get_carg_spec()` (*hdmf.build.objectmapper.ObjectMapper* method), 55
`get_class()` (in module *hdmf.common*), 43
`get_cls()` (*hdmf.build.manager.BuildManager* method), 50
`get_cls()` (*hdmf.build.manager.TypeMap* method), 52
`get_const_arg()` (*hdmf.build.objectmapper.ObjectMapper* method), 55
`get_container()` (*hdmf.backends.hdf5.h5tools.HDF5IO* method), 79
`get_container_classes()` (*hdmf.build.manager.TypeMap* method), 52
`get_container_cls()` (*hdmf.build.manager.TypeMap* method), 52
`get_container_cls_dt()` (*hdmf.build.manager.TypeMap* method), 52
`get_container_name()` (*hdmf.build.objectmapper.ObjectMapper* method), 54
`get_container_ns_dt()` (*hdmf.build.manager.TypeMap* method), 52
`get_data()` (*hdmf.array.AbstractSortedArray* method), 94
`get_data()` (*hdmf.array.Array* method), 93
`get_data_shape()` (in module *hdmf.utils*), 87
`get_data_type()` (*hdmf.spec.spec.GroupSpec* method), 69
`get_data_type_spec()` (*hdmf.spec.spec.BaseStorageSpec* class method), 65
`get_dataset()` (*hdmf.spec.spec.GroupSpec* method), 70
`get_docval()` (in module *hdmf.utils*), 86
`get_final_result()` (*hdmf.monitor.DataChunkProcessor* method), 94
`get_full_hierarchy()` (*hdmf.spec.catalog.SpecCatalog* method), 58
`get_group()` (*hdmf.spec.spec.GroupSpec* method), 69
`get_hierarchy()` (*hdmf.spec.catalog.SpecCatalog* method), 58
`get_hierarchy()` (*hdmf.spec.namespace.NamespaceCatalog* method), 61
`get_hierarchy()` (*hdmf.spec.namespace.SpecNamespace* method), 60
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.BuilderH5ReferenceDataset* class method), 75
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.BuilderH5RegionDataset* class method), 76
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.BuilderH5TableDataset* class method), 75
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.ContainerH5ReferenceDataset* class method), 75
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.ContainerH5RegionDataset* class method), 76
`get_inverse_class()` (*hdmf.backends.hdf5.h5_utils.ContainerH5TableDataset* class method), 74
`get_inverse_class()` (*hdmf.query.ReferenceResolver* class method), 95
`get_io_params()` (*hdmf.backends.hdf5.h5_utils.H5DataIO* method), 77
`get_io_params()` (*hdmf.data_utils.DataIO* method), 85
`get_iso8601_regex()` (in module *hdmf.validate.validator*), 90
`get_link()` (*hdmf.spec.spec.GroupSpec* method), 70
`get_manager()` (in module *hdmf.common*), 43
`get_map()` (*hdmf.build.manager.TypeMap* method), 53
`get_namespace()` (*hdmf.spec.namespace.NamespaceCatalog* method), 61
`get_namespace_sources()` (*hdmf.spec.namespace.NamespaceCatalog* method), 62
`get_namespace_spec()` (*hdmf.spec.spec.BaseStorageSpec* class method), 65
`get_object()` (*hdmf.backends.hdf5.h5_utils.BuilderResolverMixin* method), 73
`get_object()` (*hdmf.backends.hdf5.h5_utils.ContainerResolverMixin* method), 74
`get_object()` (*hdmf.backends.hdf5.h5_utils.DatasetOfReferences* method), 73
`get_proxy()` (*hdmf.build.manager.BuildManager* method), 50
`get_region_slicer()` (in module *hdmf*), 96
`get_registered_types()` (*hdmf.spec.catalog.SpecCatalog* method), 57
`get_registered_types()` (*hdmf.spec.namespace.NamespaceCatalog* method), 61

(hdmf.spec.namespace.SpecNamespace method), 60
 get_source_description() *(hdmf.spec.namespace.SpecNamespace method)*, 59
 get_source_files() *(hdmf.spec.namespace.SpecNamespace method)*, 59
 get_sources() *(hdmf.spec.namespace.NamespaceCatalog method)*, 61
 get_spec() *(hdmf.spec.catalog.SpecCatalog method)*, 57
 get_spec() *(hdmf.spec.namespace.NamespaceCatalog method)*, 61
 get_spec() *(hdmf.spec.namespace.SpecNamespace method)*, 60
 get_spec_loc() *(hdmf.validate.validator.Validator class method)*, 91
 get_spec_source_file() *(hdmf.spec.catalog.SpecCatalog method)*, 57
 get_subspec() *(hdmf.build.manager.BuildManager method)*, 51
 get_subspec() *(hdmf.build.manager.TypeMap method)*, 52
 get_subtypes() *(hdmf.spec.catalog.SpecCatalog method)*, 58
 get_type() *(hdmf.backends.hdf5.h5tools.HDF5IO class method)*, 79
 get_type() *(in module hdmf.validate.validator)*, 90
 get_type_map() *(in module hdmf.common)*, 43
 get_types() *(hdmf.spec.namespace.NamespaceCatalog method)*, 62
 get_validator() *(hdmf.validate.validator.ValidatorMap method)*, 90
 getargs() *(in module hdmf.utils)*, 86
 group_spec_cls *(hdmf.spec.namespace.NamespaceCatalog attribute)*, 61
 GroupBuilder *(class in hdmf.build.builders)*, 46
 groups *(hdmf.build.builders.GroupBuilder attribute)*, 46
 groups *(hdmf.spec.spec.GroupSpec attribute)*, 69
 GroupSpec *(class in hdmf.spec.spec)*, 67
 GroupValidator *(class in hdmf.validate.validator)*, 92

H

H5DataIO *(class in hdmf.backends.hdf5.h5_utils)*, 77
 H5Dataset *(class in hdmf.backends.hdf5.h5_utils)*, 73
 H5RegionSlicer *(class in hdmf.backends.hdf5.h5_utils)*, 76
 H5RoundTripMixin *(class in hdmf.testing.testcase)*, 92

H5SpecReader *(class in hdmf.backends.hdf5.h5_utils)*, 76
 H5SpecWriter *(class in hdmf.backends.hdf5.h5_utils)*, 76
 HDF5IO *(class in hdmf.backends.hdf5.h5tools)*, 78
 hdmf *(module)*, 96
 hdmf.array *(module)*, 93
 hdmf.backends *(module)*, 82
 hdmf.backends.hdf5 *(module)*, 80
 hdmf.backends.hdf5.h5_utils *(module)*, 73
 hdmf.backends.hdf5.h5tools *(module)*, 78
 hdmf.backends.io *(module)*, 81
 hdmf.backends.warnings *(module)*, 81
 hdmf.build *(module)*, 57
 hdmf.build.builders *(module)*, 45
 hdmf.build.manager *(module)*, 49
 hdmf.build.map *(module)*, 54
 hdmf.build.objectmapper *(module)*, 54
 hdmf.build.warnings *(module)*, 56
 hdmf.common *(module)*, 42
 hdmf.common.io *(module)*, 38
 hdmf.common.io.table *(module)*, 37
 hdmf.common.sparse *(module)*, 38
 hdmf.common.table *(module)*, 38
 hdmf.container *(module)*, 44
 hdmf.data_utils *(module)*, 82
 hdmf.monitor *(module)*, 94
 hdmf.query *(module)*, 95
 hdmf.region *(module)*, 96
 hdmf.spec *(module)*, 73
 hdmf.spec.catalog *(module)*, 57
 hdmf.spec.namespace *(module)*, 59
 hdmf.spec.spec *(module)*, 62
 hdmf.spec.write *(module)*, 71
 hdmf.testing *(module)*, 93
 hdmf.testing.testcase *(module)*, 92
 hdmf.utils *(module)*, 85
 hdmf.validate *(module)*, 92
 hdmf.validate.errors *(module)*, 88
 hdmf.validate.validator *(module)*, 90
 HDMFDataset *(class in hdmf.query)*, 95
 HDMFIO *(class in hdmf.backends.io)*, 81

I

id *(hdmf.common.table.DynamicTable attribute)*, 40
 id_key() *(hdmf.spec.spec.BaseStorageSpec class method)*, 65
 IllegalLinkError *(class in hdmf.validate.errors)*, 89
 inc_key() *(hdmf.spec.spec.BaseStorageSpec class method)*, 65
 include_namespace() *(hdmf.spec.write.NamespaceBuilder method)*, 72

- `include_type()` (*hdmf.spec.write.NamespaceBuilder method*), 72
- `IncorrectDataType` (*class in hdmf.validate.errors*), 89
- `Index` (*class in hdmf.common.table*), 38
- `InvalidDataIOError`, 85
- `invert()` (*hdmf.backends.hdf5.h5_utils.DatasetOfReferences method*), 73
- `invert()` (*hdmf.query.ReferenceResolver method*), 95
- `io` (*hdmf.backends.hdf5.h5_utils.H5Dataset attribute*), 73
- `io_settings` (*hdmf.backends.hdf5.h5_utils.H5DataIO attribute*), 77
- `is_empty()` (*hdmf.build.builders.GroupBuilder method*), 48
- `is_inherited_attribute()` (*hdmf.spec.spec.BaseStorageSpec method*), 64
- `is_inherited_dataset()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_inherited_group()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_inherited_link()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_inherited_spec()` (*hdmf.spec.spec.BaseStorageSpec method*), 64
- `is_inherited_spec()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_inherited_type()` (*hdmf.spec.spec.GroupSpec method*), 69
- `is_many()` (*hdmf.spec.spec.BaseStorageSpec method*), 65
- `is_many()` (*hdmf.spec.spec.LinkSpec method*), 67
- `is_overridden_attribute()` (*hdmf.spec.spec.BaseStorageSpec method*), 65
- `is_overridden_dataset()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_overridden_group()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_overridden_link()` (*hdmf.spec.spec.GroupSpec method*), 68
- `is_overridden_spec()` (*hdmf.spec.spec.BaseStorageSpec method*), 64
- `is_overridden_spec()` (*hdmf.spec.spec.GroupSpec method*), 69
- `is_overridden_type()` (*hdmf.spec.spec.GroupSpec method*), 69
- `is_ref()` (*hdmf.spec.spec.DtypeSpec static method*), 66
- `is_region()` (*hdmf.spec.spec.RefSpec method*), 63
- `items()` (*hdmf.build.builders.GroupBuilder method*), 48
- ## K
- `key_attr` (*hdmf.utils.LabelledDict attribute*), 88
- `keys()` (*hdmf.build.builders.GroupBuilder method*), 48
- ## L
- `label` (*hdmf.utils.LabelledDict attribute*), 88
- `LabelledDict` (*class in hdmf.utils*), 87
- `link_data` (*hdmf.backends.hdf5.h5_utils.H5DataIO attribute*), 77
- `link_spec_cls()` (*hdmf.spec.spec.GroupSpec class method*), 71
- `linkable` (*hdmf.spec.spec.BaseStorageSpec attribute*), 65
- `LinkBuilder` (*class in hdmf.build.builders*), 49
- `links` (*hdmf.build.builders.GroupBuilder attribute*), 46
- `links` (*hdmf.spec.spec.GroupSpec attribute*), 69
- `LinkSpec` (*class in hdmf.spec.spec*), 67
- `LinSpace` (*class in hdmf.array*), 94
- `ListSlicer` (*class in hdmf.region*), 96
- `load_namespaces()` (*hdmf.backends.hdf5.h5tools.HDF5IO class method*), 78
- `load_namespaces()` (*hdmf.build.manager.TypeMap method*), 51
- `load_namespaces()` (*hdmf.spec.namespace.NamespaceCatalog method*), 62
- `load_namespaces()` (*in module hdmf.common*), 43
- `location` (*hdmf.build.builders.BaseBuilder attribute*), 46
- `location` (*hdmf.build.manager.Proxy attribute*), 49
- `location` (*hdmf.validate.errors.Error attribute*), 88
- ## M
- `manager` (*hdmf.backends.io.HDMFIO attribute*), 81
- `manager` (*hdmf.testing.testcase.H5RoundTripMixin attribute*), 93
- `map_attr()` (*hdmf.build.objectmapper.ObjectMapper method*), 55
- `map_const_arg()` (*hdmf.build.objectmapper.ObjectMapper method*), 55
- `map_spec()` (*hdmf.build.objectmapper.ObjectMapper method*), 55
- `matches()` (*hdmf.build.manager.Proxy method*), 50
- `maxshape` (*hdmf.build.builders.DatasetBuilder attribute*), 48
- `maxshape` (*hdmf.data_utils.AbstractDataChunkIterator attribute*), 82
- `maxshape` (*hdmf.data_utils.DataChunkIterator attribute*), 83
- `merge()` (*hdmf.build.manager.TypeMap method*), 51

- merge () (*hdmf.spec.namespace.NamespaceCatalog method*), 61
- MissingDataType (*class in hdmf.validate.errors*), 89
- MissingError (*class in hdmf.validate.errors*), 89
- MissingRequiredWarning, 56
- mode (*hdmf.backends.hdf5.h5tools.HDF5IO attribute*), 80
- modified (*hdmf.container.AbstractContainer attribute*), 44
- ## N
- name (*hdmf.build.builders.Builder attribute*), 45
- name (*hdmf.container.AbstractContainer attribute*), 44
- name (*hdmf.spec.namespace.SpecNamespace attribute*), 59
- name (*hdmf.spec.spec.DtypeSpec attribute*), 66
- name (*hdmf.spec.spec.Spec attribute*), 63
- name (*hdmf.spec.write.NamespaceBuilder attribute*), 72
- name (*hdmf.validate.errors.Error attribute*), 88
- namespace (*hdmf.build.manager.Proxy attribute*), 50
- namespace (*hdmf.build.manager.TypeSource attribute*), 51
- namespace (*hdmf.common.sparse.CSRMatrix attribute*), 38
- namespace (*hdmf.common.table.DynamicTable attribute*), 41
- namespace (*hdmf.common.table.DynamicTableRegion attribute*), 42
- namespace (*hdmf.common.table.ElementIdentifiers attribute*), 39
- namespace (*hdmf.common.table.Index attribute*), 38
- namespace (*hdmf.common.table.VectorData attribute*), 39
- namespace (*hdmf.common.table.VectorIndex attribute*), 39
- namespace (*hdmf.container.Container attribute*), 44
- namespace (*hdmf.container.Data attribute*), 45
- namespace (*hdmf.validate.validator.ValidatorMap attribute*), 90
- namespace_catalog (*hdmf.build.manager.BuildManager attribute*), 50
- namespace_catalog (*hdmf.build.manager.TypeMap attribute*), 51
- NamespaceBuilder (*class in hdmf.spec.write*), 71
- NamespaceCatalog (*class in hdmf.spec.namespace*), 60
- namespaces (*hdmf.spec.namespace.NamespaceCatalog attribute*), 61
- next () (*hdmf.data_utils.DataChunkIterator method*), 83
- next () (*hdmf.query.HDMFDataset method*), 95
- no_convert () (*hdmf.build.objectmapper.ObjectMapper class method*), 54
- NotYetExhausted, 94
- NumSampleCounter (*class in hdmf.monitor*), 94
- ## O
- obj_attrs (*hdmf.build.objectmapper.ObjectMapper attribute*), 56
- obj_attrs (*hdmf.common.io.table.DynamicTableMap attribute*), 37
- object_attr () (*hdmf.build.objectmapper.ObjectMapper static method*), 54
- object_id (*hdmf.container.AbstractContainer attribute*), 44
- OBJECT_REF_TYPE (*hdmf.build.builders.DatasetBuilder attribute*), 48
- ObjectMapper (*class in hdmf.build.objectmapper*), 54
- open () (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
- open () (*hdmf.backends.io.HDMFIO method*), 81
- OrphanContainerWarning, 56
- ## P
- parent (*hdmf.build.builders.Builder attribute*), 45
- parent (*hdmf.container.AbstractContainer attribute*), 44
- parent (*hdmf.spec.spec.Spec attribute*), 63
- path (*hdmf.build.builders.Builder attribute*), 45
- popargs () (*in module hdmf.utils*), 86
- post_init () (*hdmf.utils.ExtenderMeta class method*), 87
- pre_init () (*hdmf.utils.ExtenderMeta class method*), 87
- prebuilt () (*hdmf.build.manager.BuildManager method*), 50
- primary_dtype_synonyms (*hdmf.spec.spec.DtypeHelper attribute*), 62
- process_data_chunk () (*hdmf.monitor.DataChunkProcessor method*), 94
- process_data_chunk () (*hdmf.monitor.NumSampleCounter method*), 94
- Proxy (*class in hdmf.build.manager*), 49
- pystr () (*in module hdmf.utils*), 87
- ## Q
- quantity (*hdmf.spec.spec.BaseStorageSpec attribute*), 65
- quantity (*hdmf.spec.spec.LinkSpec attribute*), 67
- Query (*class in hdmf.query*), 95
- ## R
- read () (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79

- read() (*hdmf.backends.io.HDMFIO method*), 81
 read_builder() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
 read_builder() (*hdmf.backends.io.HDMFIO method*), 81
 read_namespace() (*hdmf.backends.hdf5.h5_utils.H5SpecReader method*), 76
 read_namespace() (*hdmf.spec.namespace.SpecReader method*), 60
 read_namespace() (*hdmf.spec.namespace.YAMLSpecReader method*), 60
 read_spec() (*hdmf.backends.hdf5.h5_utils.H5SpecReader method*), 76
 read_spec() (*hdmf.spec.namespace.SpecReader method*), 60
 read_spec() (*hdmf.spec.namespace.YAMLSpecReader method*), 60
 reason (*hdmf.validate.errors.Error attribute*), 88
 recommended_chunk_shape() (*hdmf.data_utils.AbstractDataChunkIterator method*), 82
 recommended_chunk_shape() (*hdmf.data_utils.DataChunkIterator method*), 83
 recommended_chunk_shape() (*hdmf.monitor.DataChunkProcessor method*), 94
 recommended_data_shape() (*hdmf.data_utils.AbstractDataChunkIterator method*), 82
 recommended_data_shape() (*hdmf.data_utils.DataChunkIterator method*), 83
 recommended_data_shape() (*hdmf.monitor.DataChunkProcessor method*), 94
 recommended_primary_dtypes (*hdmf.spec.spec.DtypeHelper attribute*), 62
 ref (*hdmf.backends.hdf5.h5_utils.H5Dataset attribute*), 73
 ReferenceBuilder (*class in hdmf.build.builders*), 49
 ReferenceResolver (*class in hdmf.query*), 95
 RefSpec (*class in hdmf.spec.spec*), 63
 reftype (*hdmf.spec.spec.RefSpec attribute*), 63
 region (*hdmf.build.builders.RegionBuilder attribute*), 49
 region (*hdmf.container.DataRegion attribute*), 45
 region (*hdmf.region.RegionSlicer attribute*), 96
 REGION_REF_TYPE (*hdmf.build.builders.DatasetBuilder attribute*), 48
 RegionBuilder (*class in hdmf.build.builders*), 49
 regionref (*hdmf.backends.hdf5.h5_utils.H5Dataset attribute*), 73
 RegionSlicer (*class in hdmf.region*), 96
 register_class() (*in module hdmf.common*), 42
 register_container_type() (*hdmf.build.manager.TypeMap method*), 53
 register_reader_map() (*hdmf.build.manager.TypeMap method*), 53
 register_map() (*in module hdmf.common*), 42
 register_spec() (*hdmf.spec.catalog.SpecCatalog method*), 57
 reorder_yaml() (*hdmf.spec.write.YAMLSpecWriter method*), 71
 required (*hdmf.spec.spec.AttributeSpec attribute*), 64
 required (*hdmf.spec.spec.BaseStorageSpec attribute*), 64
 required (*hdmf.spec.spec.LinkSpec attribute*), 67
 resolve() (*hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset method*), 74
 resolve() (*hdmf.build.manager.Proxy method*), 50
 resolve_spec() (*hdmf.spec.spec.BaseStorageSpec method*), 64
 resolve_spec() (*hdmf.spec.spec.DatasetSpec method*), 67
 resolve_spec() (*hdmf.spec.spec.GroupSpec method*), 68
 resolved (*hdmf.spec.spec.BaseStorageSpec attribute*), 64
 roundtripContainer() (*hdmf.testing.testcase.H5RoundTripMixin method*), 93
- ## S
- schema (*hdmf.spec.namespace.SpecNamespace attribute*), 59
 set_attribute() (*hdmf.build.builders.BaseBuilder method*), 46
 set_attribute() (*hdmf.build.builders.GroupBuilder method*), 46
 set_attribute() (*hdmf.spec.spec.BaseStorageSpec method*), 66
 set_attributes() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
 set_builder() (*hdmf.build.builders.GroupBuilder method*), 47
 set_dataio() (*hdmf.container.Data method*), 44
 set_dataset() (*hdmf.build.builders.GroupBuilder method*), 47
 set_dataset() (*hdmf.spec.spec.GroupSpec method*), 70
 set_group() (*hdmf.build.builders.GroupBuilder method*), 47
 set_group() (*hdmf.spec.spec.GroupSpec method*), 69
 set_link() (*hdmf.build.builders.GroupBuilder method*), 48

- set_link() (*hdmf.spec.spec.GroupSpec* method), 70
 set_modified() (*hdmf.container.AbstractContainer* method), 44
 setUp() (*hdmf.testing.testcase.H5RoundTripMixin* method), 93
 setUpContainer() (*hdmf.testing.testcase.H5RoundTripMixin* method), 93
 shape (*hdmf.backends.hdf5.h5_utils.H5Dataset* attribute), 73
 shape (*hdmf.common.sparse.CSRMatrix* attribute), 38
 shape (*hdmf.common.table.DynamicTableRegion* attribute), 42
 shape (*hdmf.container.Data* attribute), 44
 shape (*hdmf.spec.spec.AttributeSpec* attribute), 64
 shape (*hdmf.spec.spec.DatasetSpec* attribute), 67
 SHAPE_ERROR (*hdmf.data_utils.ShapeValidatorResult* attribute), 85
 ShapeError (*class in hdmf.validate.errors*), 89
 ShapeValidatorResult (*class in hdmf.data_utils*), 84
 simplify_cpd_type() (*hdmf.spec.spec.DtypeHelper* static method), 62
 slice (*hdmf.region.RegionSlicer* attribute), 96
 sort_keys() (*hdmf.spec.write.YAMLSpecWriter* method), 71
 SortedArray (*class in hdmf.array*), 94
 source (*hdmf.backends.io.HDMFIO* attribute), 81
 source (*hdmf.build.builders.Builder* attribute), 45
 source (*hdmf.build.builders.GroupBuilder* attribute), 46
 source (*hdmf.build.manager.Proxy* attribute), 49
 source (*hdmf.spec.namespace.SpecReader* attribute), 60
 Spec (*class in hdmf.spec.spec*), 62
 spec (*hdmf.build.objectmapper.ObjectMapper* attribute), 54
 spec (*hdmf.validate.validator.Validator* attribute), 91
 spec_namespace_cls (*hdmf.spec.namespace.NamespaceCatalog* attribute), 61
 SpecCatalog (*class in hdmf.spec.catalog*), 57
 SpecFileBuilder (*class in hdmf.spec.write*), 72
 SpecNamespace (*class in hdmf.spec.namespace*), 59
 SpecReader (*class in hdmf.spec.namespace*), 60
 SpecWriter (*class in hdmf.spec.write*), 71
 stringify() (*hdmf.backends.hdf5.h5_utils.H5SpecWriter* static method), 76
- T**
 table (*hdmf.common.table.DynamicTableRegion* attribute), 42
 target (*hdmf.common.table.Index* attribute), 38
 target (*hdmf.region.RegionSlicer* attribute), 96
 target_type (*hdmf.spec.spec.LinkSpec* attribute), 67
 target_type (*hdmf.spec.spec.RefSpec* attribute), 63
 tearDown() (*hdmf.testing.testcase.H5RoundTripMixin* method), 93
 test_roundtrip() (*hdmf.testing.testcase.H5RoundTripMixin* method), 93
 TestCase (*class in hdmf.testing.testcase*), 92
 to_dataframe() (*hdmf.common.table.DynamicTable* method), 41
 to_dataframe() (*hdmf.common.table.DynamicTableRegion* method), 42
 to_spmat() (*hdmf.common.sparse.CSRMatrix* method), 38
 type_hierarchy() (*hdmf.container.AbstractContainer* class method), 44
 type_key() (*hdmf.spec.spec.BaseStorageSpec* class method), 65
 type_map (*hdmf.build.manager.BuildManager* attribute), 50
 TypeDoesNotExistError, 54
 TypeMap (*class in hdmf.build.manager*), 51
 types (*hdmf.backends.hdf5.h5_utils.AbstractH5TableDataset* attribute), 74
 types_key() (*hdmf.spec.namespace.SpecNamespace* class method), 59
 TypeSource (*class in hdmf.build.manager*), 51
- U**
 unmap() (*hdmf.build.objectmapper.ObjectMapper* method), 55
 UnsupportedOperation, 81
 UNVERSIONED (*hdmf.spec.namespace.SpecNamespace* attribute), 59
- V**
 valid (*hdmf.backends.hdf5.h5_utils.H5DataIO* attribute), 77
 valid (*hdmf.data_utils.DataIO* attribute), 85
 valid_primary_dtypes (*hdmf.spec.spec.DtypeHelper* attribute), 62
 valid_types() (*hdmf.validate.validator.ValidatorMap* method), 90
 validate() (*hdmf.testing.testcase.H5RoundTripMixin* method), 93
 validate() (*hdmf.validate.validator.AttributeValidator* method), 91
 validate() (*hdmf.validate.validator.BaseStorageValidator* method), 91
 validate() (*hdmf.validate.validator.DatasetValidator* method), 92
 validate() (*hdmf.validate.validator.GroupValidator* method), 92

validate() (*hdmf.validate.validator.Validator method*), 91
 validate() (*hdmf.validate.validator.ValidatorMap method*), 90
 validate() (*in module hdmf.common*), 43
 Validator (*class in hdmf.validate.validator*), 90
 ValidatorMap (*class in hdmf.validate.validator*), 90
 value (*hdmf.spec.spec.AttributeSpec attribute*), 63
 values() (*hdmf.build.builders.GroupBuilder method*), 48
 VectorData (*class in hdmf.common.table*), 38
 VectorIndex (*class in hdmf.common.table*), 39
 version (*hdmf.spec.namespace.SpecNamespace attribute*), 59
 vmap (*hdmf.validate.validator.Validator attribute*), 91

W

WARNING (*hdmf.utils.AllowPositional attribute*), 85
 write() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 78
 write() (*hdmf.backends.io.HDMFIO method*), 81
 write_builder() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 79
 write_builder() (*hdmf.backends.io.HDMFIO method*), 81
 write_dataset() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 80
 write_group() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 80
 write_link() (*hdmf.backends.hdf5.h5tools.HDF5IO method*), 80
 write_namespace() (*hdmf.backends.hdf5.h5_utils.H5SpecWriter method*), 76
 write_namespace() (*hdmf.spec.write.SpecWriter method*), 71
 write_namespace() (*hdmf.spec.write.YAMLSpecWriter method*), 71
 write_spec() (*hdmf.backends.hdf5.h5_utils.H5SpecWriter method*), 76
 write_spec() (*hdmf.spec.write.SpecWriter method*), 71
 write_spec() (*hdmf.spec.write.YAMLSpecWriter method*), 71
 written (*hdmf.build.builders.Builder attribute*), 45

Y

YAMLSpecReader (*class in hdmf.spec.namespace*), 60
 YAMLSpecWriter (*class in hdmf.spec.write*), 71