exdir Documentation

Svenn-Arne Dragly, Milad H. Mobarhan, Mikkel E. Lepperød

Feb 23, 2021

CONTENTS

The Experimental Directory Structure (Exdir) is a proposed, open file format specification for experimental pipelines. Exdir uses the same abstractions as HDF5 and is compatible with the HDF5 Abstract Data Model, but stores data and metadata in directories instead of in a single file. Exdir uses file system directories to represent the hierarchy, with metadata stored in human-readable YAML files, datasets stored in binary NumPy files, and raw data stored directly in subdirectories. Furthermore, storing data in multiple files makes it easier to track for version control systems. Exdir is not a file format in itself, but a specification for organizing files in a directory structure. With the publication of Exdir, we invite the scientific community to join the development to create an open specification that will serve as many needs as possible and as a foundation for open access to and exchange of data.

Exdir is described in detail in our reasearch paper:

Experimental Directory Structure (Exdir): An Alternative to HDF5 Without Introducing a New File Format.

CHAPTER

ONE

INSTALLATION

1.1 Pre-configured installation (recommended)

It's strongly recommended that you use Anaconda to install exdir along with its compiled dependencies.

With Anaconda or Miniconda:

```
conda install -c cinpla exdir
```

CHAPTER

TWO

FILE OBJECTS

class exdir.core.**File**(*directory*, *mode=None*, *allow_remove=False*, *name_validation=None*, *plugins=None*)

Bases: exdir.core.group.Group

Exdir file object. A File is a special type of *Group*. See *Group* for documentation of inherited functions.

To create a File, call the File constructor with the name of the File you wish to create:

```
>>> import exdir
>>> import numpy as np
>>> f = exdir.File("mytestfile.exdir")
```

The `File` object `f` now points to the root folder in the exdir file structure. You can add groups and datasets to it as follows:

```
>>> my_group = f.require_group("my_group")
>>> a = np.arange(100)
>>> dset = f.require_dataset("my_data", data=a)
```

The data is immediately written to disk.

Parameters

- **directory** – Name of the directory to be opened or created as an Exdir File.
- **mode** (*str, optional*) – A file mode string that defines the read/write behavior. See open() for information about the different modes.
- **allow_remove** (*bool*) – Set to True if you want mode 'w' to remove existing trees if they exist. This False by default to avoid removing entire directory trees by mistake.
- **name_validation** (*str, function, optional*) – Set the validation mode for names. Can be a function that takes a name and returns True if the name is valid or one of the following built-in validation modes:
  - 'strict': only allow numbers, lowercase letters, underscore (_) and dash (-)
  - 'simple': allow numbers, lowercase letters, uppercase letters, underscore (_) and dash (-), check if any file exists with same name in any case.
  - 'thorough': verify if name is safe on all platforms, check if any file exists with same name in any case.
  - 'none': allows any filename

  The default is 'thorough'.
- **plugins** (*list, optional*) – A list of instantiated plugins or modules with a plugins() function that returns a list of plugins.

close()
Closes the File object. Sets the OpenMode to FILE_CLOSED which denies access to any attribute or child

create_group(*name*)
Create a group with the given name or absolute path.

See *Group* for more details.

Note: Creating groups with absolute paths is only allowed on File objects and not on Group objects in general.

---

require_group(*name*)
Open an existing subgroup or create one if it does not exist.

See *Group* for more details.

Note: Creating groups with absolute paths is only allowed on File objects and not on Group objects in general.

CHAPTER

THREE

GROUPS

class exdir.core.**Group**(*root_directory*, *parent_path*, *object_name*, *file*)

Bases: exdir.core.exdir_object.Object

Container of other groups and datasets.

create_dataset(*name*, *shape=None*, *dtype=None*, *data=None*, *fillvalue=None*)

Create a dataset. This will create a folder on the filesystem with the given name, an exdir.yaml file that identifies the folder as an Exdir Dataset, and a data.npy file that contains the data.

Parameters

- **name** (*str*) – Name of the dataset to be created.
- **shape** (*tuple, semi-optional*) – Shape of the dataset to be created. Must be set together with dtype. Cannot be set together with *data*, but must be set if *data* is not set.
- **dtype** (*numpy.dtype*) – Data type of the dataset to be created. Must be set together with *shape*. Cannot be set together with *data*, but must be set if *data* is not set.
- **data** (*scalar, list, numpy.array or plugin-supported type, semi-optional*) – Data to be inserted in the created dataset. Cannot be set together with *dtype* or *shape*, but must be set if *dtype* and *shape* are not set.
- **fillvalue** (*scalar*) – Used to create a dataset with the given *shape* and *type* with the initial value of *fillvalue*.

Returns

Return type The newly created Dataset.

Raises **FileExistsError** – If an object with the same *name* already exists.

See also:

require_dataset

create_group(*name*)

Create a group. This will create a folder on the filesystem with the given name and an exdir.yaml file that identifies the folder as a group. A group can contain multiple groups and datasets.

Parameters **name** (*str*) – Name of the subgroup. Must follow the naming convention of the parent Exdir File.

Raises **FileExistsError** – If an object with the same *name* already exists.

Returns

Return type The newly created Group.

See also:

require_group

get (*key*)
Get an object in the group. :Parameters: **key** (*str*) – The key of the desired object

Returns

Return type Value or None if object does not exist.

items ()

Returns A view of the keys and objects in the group.

Return type ItemsView

keys ()

Returns A view of the names of the objects in the group.

Return type KeysView

require_dataset (*name*, *shape=None*, *dtype=None*, *exact=False*, *data=None*, *fillvalue=None*)
Open an existing dataset or create it if it does not exist.

Parameters

- **name** (*str*) – Name of the dataset. Must follow naming convention of parent Exdir File.
- **shape** (*np.array, semi-optional*) – Shape of the dataset. Must be set together with dtype. Cannot be set together with data, but must be set if data is not set. Will be used to verify that an existing dataset has the same shape or to create a new dataset of the given shape. See also *exact*.
- **dtype** (*np.dtype, semi-optional*) – NumPy datatype of the dataset. Must be set together with shape. Cannot be set together with data, but must be set if data is not set. Will be used to verify that an existing dataset has the same or a convertible dtype or to create a new dataset with the given dtype. See also *exact*.
- **exact** (*bool, optional*) – Only used if the dataset already exists. If *exact* is *False*, the shape must match the existing dataset and the data type must be convertible between the existing and requested data type. If *exact* is *True*, the *shape* and *dtype* must match exactly. The default is False. See also *shape*, *dtype* and *data*.
- **data** (*list, np.array, semi-optional*) – The data that will be used to create the dataset if it does not already exist. The shape and dtype of *data* will be compared to the existing dataset if it already exists. See *shape*, *dtype* and *exact*.
- **fillvalue** (*scalar*) – Used to create a dataset with the given *shape* and *type* with the initial value of *fillvalue*.

require_group (*name*)
Open an existing subgroup or create one if it does not exist. This might create a new subfolder on the file system.

Parameters **name** (*str*) – Name of the subgroup. Must follow the naming convention of the parent Exdir File.

Returns

Return type The existing or created group.

See also:

create_group

values()

Returns A view of the objects in the group.

Return type ValuesView

CHAPTER

FOUR

DATASETS

This is data set class. It has class *exdir.core.Dataset*:

class exdir.core.**Dataset**(*root_directory*, *parent_path*, *object_name*, *file*)

Bases: exdir.core.exdir_object.Object

Dataset class

> **Warning:** This class modifies the view and it is possible to overwrite an existing dataset, which is different from the behavior in h5py.

property data

Property that gives access the entire dataset. Equivalent to calling `dataset[:]`.

Returns The entire dataset.

Return type numpy.memmap

property dtype

The NumPy data type of the dataset. Equivalent to calling `dataset[:].dtype`.

Returns The NumPy data type of the dataset.

Return type numpy.dtype

set_data(*data*)

> **Warning:** Deprecated convenience function. Use `dataset.data = data` instead.

property shape

The shape of the dataset. Equivalent to calling `dataset[:].shape`.

Returns The shape of the dataset.

Return type tuple

property size

The size of the dataset. Equivalent to calling `dataset[:].size`.

Returns The size of the dataset.

Return type np.int64

property value

Convenience alias for the `data` property.

Warning: This property is only provided as a convenience to make the API interoperable with h5py. We recommend to use `data` instead of `value`.

CHAPTER

FIVE

RAW

class exdir.core.**Raw**(*root_directory*, *parent_path*, *object_name*, *file*)

Bases: exdir.core.exdir_object.Object

Raw objects are simple folders with any content.

Raw objects currently have no features apart from showing their path.

CHAPTER

SIX

ATTRIBUTES

class exdir.core.**Attribute**(*parent*, *mode*, *file*, *path=None*)

Bases: object

The attribute object is a dictionary-like object that is used to access the attributes stored in the `attributes.yaml` file for a given Exdir Object.

The Attribute object should not be created, but retrieved by accessing the `.attrs` property of any Exdir Object, such as a Dataset, Group or File.

property filename

Returns

Return type The filename of the `attributes.yaml` file.

items()

Returns

Return type a new view of the Attribute's items.

keys()

Returns

Return type a new view of the Attribute's keys.

to_dict()

Convert the Attribute into a standard Python dictionary.

update(*value*)

Update the Attribute with the key/value pairs from `value`, overwriting existing keys.

This function accepts either another Attribute object, a dictionary object or an iterable of key/value pairs

values()

Returns

Return type a new view of the Attribute's values.

CHAPTER

SEVEN

PLUGINS

The functionality of Exdir can be extended with plugins. These allow modifying the behavior of Exdir when enabled. For instance, dataset and attribute plugins can perform pre- and post-processing of data during reading and writing operations. Note that plugins do not change the underlying specifications of Exdir. Plugins are intended to perform verification of data consistency, and to provide convenient mapping from general in-memory objects to objects that can be stored in the Exdir format and back again. Some plugins are provided in the exdir.plugins module, while new plugins can be defined by Exdir users or package developers.

One of the built-in plugins provides experimental support for units using the *quantities* package:

```
>>> import exdir
>>> import exdir.plugins.quantities
>>> import quantities as pq
>>> f = exdir.File("test.exdir", plugins=[exdir.plugins.quantities])
>>> q = np.array([1,2,3])*pq.mV
>>> dset_q = f.create_dataset("quantities_array", data=q)
>>> dset_q[:]
array([ 1.,  2.,  3.]) * mV
```

As shown in the above example, a plugin is enabled when creating a File object by passing the plugin to the plugins argument.

To create a custom plugin, one of the handler classes in *exdir.plugin_interface* must be inherited. The abstract handler classes are named after the object type you want to create a handler for. In this example we have a simplified *Quantity* class, which only contains a magnitude and a corresponding unit:

```
>>> class Quantity:
>>>     def __init__(self, magnitude, unit):
>>>         self.magnitude = magnitude
>>>         self.unit = unit
```

Below, we create a plugin that enables us to directly use a *Quantity* object as a *Dataset* in Exdir. We do this by inheriting from *exdir.plugin_interface.Dataset* and overloading *prepare_write* and *prepare_read*:

```
>>> import exdir
>>> class DatasetQuantity(exdir.plugin_interface.Dataset):
>>>     def prepare_write(self, dataset_data):
>>>         magnitude = dataset_data.data.magnitude
>>>         unit = dataset_data.data.unit
>>>
>>>         dataset_data.data = magnitude
>>>         dataset_data.attrs = {"unit": unit}
>>>
>>>         return dataset_data
>>>
```

(continues on next page)

(continued from previous page)

```
>>>     def prepare_read(self, dataset_data):
>>>         unit = dataset_data.attrs["unit"]
>>>         magnitude = dataset_data.data
>>>
>>>         dataset_data.data = Quantity(magnitude, unit)
>>>
>>>         return dataset_data
```

The overloaded functions take *dataset_data* as an argument. This has the *data*, *attrs*, and *meta* properties. The property *attrs* is a dictionary with optional attributes, while *meta* is a dictionary with information about the plugin.

In *prepare_write*, the magnitude and unit of the data is translated to a value (numeric or *numpy.ndarray*) and an attribute (dictionary-like) that then can be written to file. *prepare_read* receives the data from the NumPy file and the attributes from the YAML file, and uses these to reconstruct a *Quantity* object.

We create a plugin that uses this handler as follows:

```
>>> my_plugin = exdir.plugin_interface.Plugin(
>>>     name="dataset_quantity",
>>>     dataset_plugins=[DatasetQuantity()]
>>> )
```

The plugin is enabled when opening a File by passing it to the plugins parameter:

```
>>> f = exdir.File("test.exdir", plugins=[my_plugin])
>>> dset = f.create_dataset("test", data=Quantity(1.5, "meter"))
```

CHAPTER

EIGHT

SPECIFICATION

exdir is not a file format in itself, but rather a specification for a directory structure with NumPy and YAML files.

```
example.exdir (File, folder)
│   attributes.yaml (-, file)
│   exdir.yaml (-, file)
│
├── dataset1 (Dataset, folder)
│   ├── data.npy (-, file)
│   ├── attributes.yaml (-, file)
│   └── exdir.yaml (-, file)
│
└── group1 (Group, folder)
    ├── attributes.yaml (-, file)
    ├── exdir.yaml (-, file)
    │
    └── dataset2 (Dataset, folder)
        ├── data.npy (-, file)
        ├── attributes.yaml (-, file)
        ├── exdir.yaml (-, file)
        │
        └── raw (Raw, folder)
            ├── image0001.tif (-, file)
            ├── image0002.tif (-, file)
            └── ...
```

The above structure shows the name of the object, the type of the object in exdir and the type of the object on the file system as follows:

` [name] ([exdir type], [file system type]) `

A dash (-) indicates that the object doesn't have a separate internal representation in the format, but is used indirectly. It is however explicitly stored in the file system.

CHAPTER

NINE

INSTALL

With Anaconda or Miniconda:

```
conda install -c cinpla exdir
```

For more, see *Installation*.

CHAPTER

TEN

QUICK USAGE EXAMPLE

```
>>> import exdir
>>> import numpy as np
>>> f = exdir.File("mytestfile.exdir")
```

The *File object* points to the root folder in the exdir file structure. You can add groups and datasets to it.

```
>>> my_group = f.require_group("my_group")
>>> a = np.arange(100)
>>> dset = f.require_dataset("my_data", data=a)
```

These can later be accessed with square brackets:

```
>>> f["my_data"][10]
10
```

Groups can hold other groups or datasets:

```
>>> subgroup = my_group.require_group("subgroup")
>>> subdata = subgroup.require_dataset("subdata", data=a)
```

Datasets support array-style slicing:

```
>>> dset[0:100:10]
memmap([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

Attributes can be added to files, groups and datasets:

```
>>> f.attrs["description"] = "My first exdir file"
>>> my_group.attrs["meaning_of_life"] = 42
>>> dset.attrs["trial_number"] = 12
>>> f.attrs["description"]
'My first exdir file'
```

CHAPTER

ELEVEN

CORE CONCEPTS

An exdir object contains two types of objects: *datasets*, which are array-like collections of data, and *groups*, which are directories containing datasets and other groups.

An exdir directory is created by:

```
>>> import exdir
>>> import numpy as np
>>> f = exdir.File("myfile.exdir", "w")
```

The *File object* containes many useful methods including *exdir.core.Group.require_dataset()*:

```
>>> data = np.arange(100)
>>> dset = f.require_dataset("mydataset", data=data)
```

The created object is not an array but *an exdir dataset*. Like NumPy arrays, datasets have a shape:

```
>>> dset.shape
(100,)
```

Also array-style slicing is supported:

```
>>> dset[0]
0
>>> dset[10]
10
>>> dset[0:100:10]
memmap([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

For more, see *File Objects* and *Datasets*.

CHAPTER

TWELVE

GROUPS AND HIERARCHICAL ORGANIZATION

Every object in an exdir directory has a name, and they're arranged in a POSIX-style hierarchy with /-separators:

```
>>> dset.name
'/mydataset'
```

The "directory" in this system are called *groups*. The *File object* we created is itself a group, in this case the *root group*, named /

```
>>> f.name
'/'
```

Creating a subgroup is done by using *exdir.core.Group.require_group()* method:

```
>>> grp = f.require_group("subgroup")
```

All *exdir.core.Group* objects also have the `require_*` methods like File:

```
>>> dset2 = grp.require_dataset("another_dataset", data=data)
>>> dset2.name
'/subgroup/another_dataset'
```

You retrieve objects in the file using the item-retrieval syntax:

```
>>> dataset_three = f['subgroup/another_dataset']
```

Iterating over a group provides the names of its members:

```
>>> for name in f:
...     print(name)
mydataset
subgroup
```

Containership testing also uses names:

```
>>> "mydataset" in f
True
>>> "somethingelse" in f
False
```

You can even use full path names:

```
>>> "subgroup/another_dataset" in f
True
```

(continues on next page)

(continued from previous page)

```
>>> "subgroup/somethingelse" in f
False
```

There are also the familiar *exdir.core.Group.keys()*, *exdir.core.Group.values()*, *exdir.core.Group.items()* and `exdir.core.Group.iter()` methods, as well as *exdir.core.Group.get()*.

For more, see *Groups*.

CHAPTER

THIRTEEN

ATTRIBUTES

With exdir you can store metadata right next to the data it describes. All groups and datasets can have attributes which are descibed by `exdir.core.attributes()`.

Attributes are accessed through the `attrs` proxy object, which again implements the dictionary interface:

```
>>> dset.attrs['temperature'] = 99.5
>>> dset.attrs['temperature']
99.5
>>> 'temperature' in dset.attrs
True
```

For more, see *Attributes*.

CHAPTER

FOURTEEN

ACKNOWLEDGEMENTS

The development of Exdir owes a great deal to other standardization efforts in science in general and neuroscience in particular, among them the contributors to HDF5, NumPy, YAML, PyYAML, ruamel-yaml, SciPy, Klusta Kwik, NeuralEnsemble, and Neurodata Without Borders.

CHAPTER

FIFTEEN

REFERENCES

- genindex
- search

INDEX