nsdf Documentation

Release 0.1

Subhasis Ray

November 04, 2014

Contents

Contents:

CHAPTER 1

NSDF Package

1.1 `nsdf` Package

1.1.1 NSDF

NSDF (Neuroscience Simulation Data Format) is a file format built on top of HDF5.

Although the design and development started with the aim of storing data generated from simulations in computational neuroscience, this format is generic enough that any time series data should fit in. Thus the actual application can be much broader than simulations in neuroscience.

There are three top level groups in an NSDF file: *model*, *data* and *map* for storing information about the model (computational model of a neuron, the human brain, model of the world, etc.), data collected from model components and the mapping between the data and its source (model component) respectively.

Data

NSDF stores the data collected from a model under the */data* group. Data is organized into four subgroups:

uniform: Time series data that has been sampled with a uniform sampling interval. This is the case for simulations that use fixed time step for integration. In this case the sampling interval *dt* and the start time is enough to determine the sampling time of any data point unambiguously.

nonuniform: Time series data that has been sampled with irregular sampling interval. In this case sampling time for each data point must be explicitly stored. This is the case for simulations using variable time step inetgration methods, like *cvode*.

There can be several scenarios under nonuniform sampling.

1. **simultaneous sampling: Although the sampling intervals are** different, all data sources are sampled simultaneously. Thus the length of data from each source is same and data points with the same index have the same sampling time. In this case, data from an entire population of data sources can be put together in a homogeneous 2D dataset and the share the sampling times.

   This is the case when the CVODE solver is global for a simulation.

   This is represented by the dialect nsdf.dialect.NUREGULAR.

2. **independent sampling: In a more general case, the data should** be sampled more densely when the variable is changing fast and sparsely when the rate of change is slow. If the data sources are independent of each other, they may be sampled at different time points. Thus data recorded for a fixed duration will of different length for different sources.

In this case data from a population of sources can be stored as:

(a) individual 1D datasets under a common group.

nsdf.dialect.ONED represents this case

(b) 2D vlen(variable length) dataset which represents a ragged array with each row having a different number of columns.

This is represented by nsdf.dialect.VLEN

(c) 2D dataset with NaN padding such that all rows have same number of columns.

This is represented by nsdf.dialect.NANPADDED.

event: Data points represent event times. Spike time data is the most common example in neuroscience. Similar to nonunform data, event times data from different sources are of different length. Thus they, too, can be stored in three different ways.

1. individual 1D datasets under a common group.

   nsdf.dialect.ONED represents this case

2. 2D vlen(variable length) dataset which represents a ragged array with each row having a different number of columns.

   This is represented by nsdf.dialect.VLEN.

3. 2D dataset with NaN padding such that all rows have same number of columns.

   This is represented by nsdf.dialect.NANPADDED.

static: In addition to time series or temporal data, components in a model can have static data associated with them. These are stored under the *static* group.

Model

The *model* group stores information about the model of the system from which data has been collected. This is relatively free form since different fields use different ways of describing the model. Within computational neuroscience there are multiple standards for model description. NSDF specification provides some containers for storing model description and one hierarchical structure that can be immediately useful for data analysis and visualization tools in neuroscience.

filecontents: This group can have a hierarchical structure underneath mapping the directory tree used for the model. Complex models are often organized in a directory tree and this maps nicely to the hieararchical structure in HDF5, where Groups can represent directories and Datasets can represent files. The contents of the files can be stored as strings or binary objects in the Datasets.

filerefs: This group can store datasets containing the paths of external model files. The disadvantage being that the model has to be distributed separately from the data.

links: This group can store links to external model definitions in other HDF5 files.

modeltree: This group stores a tree structure representing hierarchical models. NSDF provides mechanisms for efficient linking between the model components represented by the nodes of this tree with the data.

Each node in this tree is a Group with a *uid* attribute storing the unique identifier of the model component. There can be other attributes added by the user. One special attribute is *ontology* meant for storing the ontological term for this component.

Map

Mapping between data and its source is stored under the */map* group. Like */data*, it has the subgroups *uniform*, *nonuniform*, *event* and *static* which store the mapping between the datasets under corresponding subgroups of */data* and the model components.

For all datasets except event and nonuniform data stored in 1D datasets, the uid of a population of sources are stored as a dataset under the corresponding group and linked to the datasets recorded from these as a HDF5 DimensionScale (DS). Multiple variables recorded from the same source population share this dimension scale. The rows in the population DS have one to one correspondence with the rows in the datasets.

For 1D datasets, a 2 column dataset is created for each population for each recorded variable. The first column is *source* and stores the uid of the source component and the second column is *data* which stores the reference to the 1D dataset collected from this source.

Note on namespace

The nsdf package is organized into *constants Module*, *nsdfdata Module*, *model Module*, *nsdfwriter Module* and *util Module* submodules. However all their contents are directly accessible under the *nsdf* namespace. Thus, instead of *nsdf.nsdfwriter.NSDFWriter* you should use *nsdf.NSDFWriter*.

CHAPTER 2

Constants

2.1 constants Module

class nsdf.constants.**dialect**
Bases: object

Enumeration of different dialects of NSDF.

The following constants are defined:

VLEN: nonuniform and event data stored in vlen 2d datasets.

ONED: nonuniform and event data stored in 1d datasets.

NANPADDED: nonuniform and event data stored in regular 2d datasets with NaN padding.

NUREGULAR: nonuniform datasets have shared sampling times. Thus nonuniform data goes into regular 2D datasets. In this case the events are stored in 1D datasets.

CHAPTER 3

Data containers

3.1 nsdfdata Module

Classes for NSDF data.

class nsdf.nsdfdata.**EventData**(**args*, ***kwargs*)
Bases: nsdf.nsdfdata.NSDFData

Stores event times recorded from data sources.

class nsdf.nsdfdata.**NSDFData**(*name*, *unit=None*, *field=None*, *dtype=<Mock object at 0x7f78cb076790>*)
Bases: object

Base class for NSDF Data.

name
str

name of the dataset.

unit
str

unit of the recorded quantity.

field
str

the recorded field/parameter of the source object. If unspecified it defaults to *name*.

dtype
numpy.dtype

type of the recorded data. Default: numpy.float64

get_all_data()
Return the data for all the sources as a list.

get_data(*source*)
Return the data for specified source

get_source_data_dict()
Return a dictionary storing the mapping from source->data.

get_sources()
Return the source ids as a list

put_data (*source*, *data*)
Set the data array for source.

Parameters

- **source** (*str*) – uid of the data source.
- **data** (*a scalar or sequence of elements of dtype*) – the data for this source.

Returns None

update_source_data_dict (*src_data*)
Insert a bunch of source, data pairs.

Parameters **src_data** (*dict-like*) – an object that is a dict or a

Returns None

Examples

```
>>> data_obj = nsdf.UniformData('current', unit='pA')
>>> ika, ikdr = [0.1, 0.3, 0.5], [0.3, 0.14]
>>> data_obj.update_source_data_dict([('KA', ika), ('KDR', ikdr)])
```

class nsdf.nsdfdata.**NonuniformData** (*name*, *unit=None*, *field=None*, *tunit=None*, *dtype=<Mock object at 0x7f78cb0768d0>*, *ttype=<Mock object at 0x7f78cb076950>*)
Bases: nsdf.nsdfdata.TimeSeriesData

Stores nonuniformly sampled data.

ttype
np.dtype data type of time points. Default np.float64

put_data (*source*, *data*)
Set the data array for source.

Parameters

- **source** (*str*) – uid of the data source.
- **data** (*a 2-tuple*) – the data and sampling times for this source.

Returns None

class nsdf.nsdfdata.**NonuniformRegularData** (**args*, ***kwargs*)
Bases: nsdf.nsdfdata.TimeSeriesData

Stores nonuniformly sampled data where all sources are sampled at the same time points.

get_times ()
Returns the sampling times for the entire dataset.

put_data (*source*, *data*)
Set the data array for source.

Parameters

- **source** (*str*) – uid of the data source.
- **data** (*a scalar or sequence of elements of dtype*) – the data for this source.

Returns None

Raises

- **ValueError if length of data does not match that of –**
- **sampling times. –**

set_times(*times*, *tunit=None*)
Set the sampling times of all the data points.

class nsdf.nsdfdata.**StaticData**(**args*, ***kwargs*)
Bases: nsdf.nsdfdata.NSDFData

Stores static data recorded from data sources.

class nsdf.nsdfdata.**UniformData**(**args*, ***kwargs*)
Bases: nsdf.nsdfdata.TimeSeriesData

Stores uniformly sampled data.

dt
float

the sampling interval of the data.

tunit
float

unit of time.

set_dt(*value*, *unit*)
Set the timestep used for data recording.

CHAPTER 4

Data structure for model hierarchy

4.1 `model` Module

Classes for representing the model.

class `nsdf.model.`**`ModelComponent`**(*name*, *uid=None*, *parent=None*, *attrs=None*, *hdfgroup=None*)

Bases: `object`

Tree node for model tree.

`parent`
ModelComponent

parent of this component.

`children`
dict

dict of child components - name is key and ModelComponent is value.

`attrs`
dict

attributes of the component. These become HDF5 attributes when it is written to file.

`hdfgroup`
hdf5 Group

the group that this component corresponds to in NSDF file.

`add_child`(*child*)
Add a child component under this model component.

Parameters **child** (*ModelComponent*) – child component to add to this component

Returns None

Raises `TypeError` –

`add_children`(*children*)
Add a list of children to current component.

Parameters **children** (*list*) – list of children to be added.

Returns None

Raises `TypeError` –

check_uid(*uid_dict*)
Check that uid are indeed unique.

Parameters **uid_dict** (*dict*) – an empty dict for storing the uids

---

Note: If any uid is not set, this function as a side effect creates the uids in the form parentuid/name - similar to unix file paths.

---

get_id_path_dict()
Return a dictionary mapping the unique id of the model components to their path in modeltree.

See also:

update_id_path_dict

get_node(*path*)
Get node at *path* relative to this node.

Parameters **path** (*str*) – path obtained by concatenating component names with / as separator.

Returns ModelComponent at the specified path

Raises **KeyError if there is no element at the specified path.** –

path
Path of this component

print_tree(*indent=''*)
Recursively print subtree rooted at this component.

Parameters **indent** (*str*) – indentation.

Returns None

update_id_path_dict()
Update the id->path mapping.

This must be called before using get_id_path_dict whenever the model tree is been modified

See also:

get_id_path_dict

visit(*function*, **args*, ***kwargs*)
Visit the subtree starting with *node* recursively, applying function *fn* to each node.

Parameters

- **node** (*ModelComponent*) – node to start with.
- **fn** (node, *args, **kwargs) – a function to apply on each node.

Returns None

nsdf.model.**common_prefix**(*paths*, *sep='/'*)
Find the common prefix of paths.

Note: does not check for malformed paths right now.

CHAPTER 5

NSDF file writer

5.1 nsdfwriter Module

Writer for NSDF file format.

class nsdf.nsdfwriter.**NSDFWriter**(*filename*, *dialect='ONED'*, *mode='a'*, ***h5args*)

Bases: object

Writer for NSDF files.

An NSDF file has three main groups: */model*, */data* and */map*.

mode
str

File open mode. Defaults to append ('a'). Can be 'w' or 'w+' also.

dialect
nsdf.dialect member

ONED for storing nonuniformly sampled and event data in 1D arrays.

VLEN for storing such data in 2D VLEN datasets.

NANPADDED for storing such data in 2D homogeneous datasets with NaN padding.

model
h5.Group

/model group

data
h5.Group

/data group

mapping
h5.Group

/map group

time_dim
h5.Group

/map/time group contains the sampling time points as dimension scales of data. It is mainly used for nonuniformly sampled data.

modeltree
(h5.Group): '/model/modeltree group can be used for storing the model in a hierarchical manner. Each subgroup under *modeltree* is a model component and can contain other subgroups representing subcomponents. Each group stores the unique identifier of the model component it represents in the string attribute *uid*.

add_event_1d(*source_ds*, *data_object*, *source_name_dict=None*, *fixed=False*)
Add event time data when data from each source is in a separate 1D dataset.

For a population of sources called {population}, a group */map/event/{population}* must be first created (using add_event_ds). This is passed as *source_ds* argument.

When adding the data, the uid of the sources and the names for the corresponding datasets must be specified in *source_name_dict* and this function will create one dataset for each source under */data/event/{population}/{name}* where {name} is the name of the data_object, preferably the field name.

Parameters

- **source_ds** (*HDF5 Dataset*) – the dataset */map/event/{populationname}{variablename}* created for this population of sources (created by add_event_ds_1d). The name of this group reflects that of the group under */data/event* which stores the datasets.
- **data_object** (*nsdf.EventData*) – NSDFData object storing the data for all sources in *source_ds*.
- **source_name_dict** (*dict*) – mapping from source id to dataset name. If None (default) it tries to use the uids in the source_ds. If the uids do not fit the hdf5 naming convention, the index of the entries in source_ds will be used.
- **fixed** (*bool*) – if True, the data cannot grow. Default: False

Returns dict mapping source ids to datasets.

add_event_ds(*name*, *idlist*)
Create a group under */map/event* with name *name* to store mapping between the datasources and event data.

Parameters

- **name** (*str*) – name with which the datasource list should be stored. This will represent a population of data sources.
- **idlist** (*list*) – unique ids of the data sources.

Returns The HDF5 Group */map/event/{name}*.

add_event_ds_1d(*popname*, *varname*, *idlist*)
Create a group under */map/event* with name *name* to store mapping between the datasources and event data.

Parameters

- **popname** (*str*) – name of the group under which the datasource list should be stored. This will represent a population of data sources.
- **varname** (*str*) – name of the dataset mapping source uid to data. This should be same as the name of the recorded variable.

Returns The HDF5 Dataset */map/event/{popname}/{varname}*.

add_event_nan(*source_ds*, *data_object*, *fixed=False*)
Add event data when data from all sources in a population is stored in a 2D array with NaN padding.

Parameters

- **source_ds** (*HDF5 Dataset*) – the dataset under */map/event* created for this population of sources (created by add_nonunifrom_ds).
- **data_object** (*nsdf.EventData*) – NSDFData object storing the data for all sources in *source_ds*.
- **fixed** (*bool*) – if True, this is a one-time write and the data cannot grow. Default: False

Returns HDF5 Dataset containing the data.

add_event_vlen (*source_ds*, *data_object*, *fixed=False*)
Add event data when data from all sources in a population is stored in a 2D ragged array.

When adding the data, the uid of the sources and the names for the corresponding datasets must be specified and this function will create the dataset */data/event/{population}/{name}* where {name} is name of the data_object, preferably the name of the field being recorded.

Parameters

- **source_ds** (*HDF5 Dataset*) – the dataset under */map/event* created for this population of sources (created by add_nonunifrom_ds).
- **data_object** (*nsdf.EventData*) – NSDFData object storing the data for all sources in *source_ds*.
- **fixed** (*bool*) – if True, this is a one-time write and the data cannot grow. Default: False

Returns HDF5 Dataset containing the data.

Notes

Concatenating old data with new data and reassigning is a poor choice for saving data incrementally. HDF5 does not seem to support appending data to VLEN datasets.

h5py does not support vlen datasets with float64 elements. Change dtype to np.float64 once that is developed.

add_model_filecontents (*filenames*, *ascii=True*, *recursive=True*)
Add the files and directories listed in *filenames* to `/model/filecontents`.

This function is for storing the contents of model files in the NSDF file. In case of external formats like NeuroML, NineML, SBML and NEURON/GENESIS scripts, this function is useful. Each directory is stored as a group and each file is stored as a dataset.

Parameters

- **filenames** (*sequence*) – the paths of files and/or directories which contain model information.
- **ascii** (*bool*) – whether the files are in ascii.
- **recursive** (*bool*) – whether to recursively store subdirectories.

add_modeltree (*root*, *target='/'*)
Add an entire model tree. This will cause the modeltree rooted at *root* to be written to the NSDF file.

Parameters

- **root** (*ModelComponent*) – root of the source tree.
- **target** (*str*) – target node path in NSDF file with respect to '/model/modeltree'. *root* and its children are added under this group.

add_nonuniform_1d(*source_ds*, *data_object*, *source_name_dict=None*, *fixed=False*)
Add nonuniform data when data from each source is in a separate 1D dataset.

For a population of sources called {population}, a group */map/nonuniform/{population}* must be first created (using add_nonuniform_ds). This is passed as *source_ds* argument.

When adding the data, the uid of the sources and the names for the corresponding datasets must be specified and this function will create one dataset for each source under */data/nonuniform/{population}/{name}* where {name} is the name of the data_object, preferably the name of the field being recorded.

This function can be used when different sources in a population are sampled at different time points for a field value. Such case may arise when each member of the population is simulated using a variable timestep method like CVODE and this timestep is not global.

Parameters

- **source_ds** (*HDF5 dataset*) – the dataset */map/nonuniform/{population}/{variable}* created for this population of sources (created by add_nonunifrom_ds_1d).
- **data_object** (*nsdf.NonuniformData*) – NSDFData object storing the data for all sources in *source_ds*.
- **source_name_dict** (*dict*) – mapping from source id to dataset name. If None (default), the uids of the sources will be used as dataset names. If the uids are not compatible with HDF5 names (contain '.' or '/'), then the index of the source in source_ds will be used.
- **fixed** (*bool*) – if True, the data cannot grow. Default: False

Returns dict mapping source ids to the tuple (dataset, time).

Raises **AssertionError when dialect is not ONED.** –

add_nonuniform_ds(*popname*, *idlist*)
Add the sources listed in idlist under /map/nonuniform/{popname}.

Parameters

- **popname** (*str*) – name with which the datasource list should be stored. This will represent a population of data sources.
- **idlist** (*list of str*) – list of unique identifiers of the data sources. This becomes irrelevant if homogeneous=False.

Returns An HDF5 Dataset storing the source ids when dialect is VLEN or NANPADDED. This is converted into a dimension scale when actual data is added.

Raises **AssertionError if idlist is empty or dialect is ONED.** –

add_nonuniform_ds_1d(*popname*, *varname*, *idlist*)
Add the sources listed in idlist under /map/nonuniform/{popname}/{varname}.

In case of 1D datasets, for each variable we store the mapping from source id to dataset ref in a two column compund dataset with dtype=[('source', VLENSTR), ('data', REFTYPE)]

Parameters

- **popname** (*str*) – name with which the datasource list should be stored. This will represent a population of data sources.
- **varname** (*str*) – name of the variable beind recorded. The same name should be passed when actual data is being added.
- **idlist** (*list of str*) – list of unique identifiers of the data sources.

Returns An HDF5 Dataset storing the source ids in *source* column.

Raises **AssertionError if idlist is empty or if dialect is not ONED.** –

add_nonuniform_nan(*source_ds*, *data_object*, *fixed=False*)

Add nonuniform data when data from all sources in a population is stored in a 2D array with NaN padding.

Parameters

- **source_ds** (*HDF5 Dataset*) – the dataset under */map/event* created for this population of sources (created by add_nonunifrom_ds).
- **data_object** (*nsdf.EventData*) – NSDFData object storing the data for all sources in *source_ds*.
- **fixed** (*bool*) – if True, this is a one-time write and the data cannot grow. Default: False

Returns HDF5 Dataset containing the data.

Notes

Concatenating old data with new data and reassigning is a poor choice for saving data incrementally. HDF5 does not seem to support appending data to VLEN datasets.

h5py does not support vlen datasets with float64 elements. Change dtype to np.float64 once that is developed.

add_nonuniform_regular(*source_ds*, *data_object*, *fixed=False*)

Append nonuniformly sampled *variable* values from *sources* to *data*. In this case sampling times of all the sources are same and the data is stored in a 2D dataset.

Parameters

- **source_ds** – the dataset storing the source ids under map. This is attached to the stored data as a dimension scale called *source* on the row dimension.
- **fixed** (*bool*) – if True, the data cannot grow. Default: False

Returns HDF5 dataset storing the data

Raises

- **KeyError if the sources in 'data_object' do not match** –
- **those in 'source_ds'.** –
- **ValueError if the data arrays are not all equal in length.** –
- **ValueError if dt is not specified or <= 0 when inserting** –
- **data for the first time.** –

add_nonuniform_vlen(*source_ds*, *data_object*, *fixed=False*)

Add nonuniform data when data from all sources in a population is stored in a 2D ragged array.

When adding the data, the uid of the sources and the names for the corresponding datasets must be specified and this function will create the dataset */data/nonuniform/{population}/{name}* where {name} is the first argument, preferably the name of the field being recorded.

This function can be used when different sources in a population are sampled at different time points for a field value. Such case may arise when each member of the population is simulated using a variable timestep method like CVODE and this timestep is not global.

Parameters

- **source_ds** (*HDF5 dataset*) – the dataset under */map/nonuniform* created for this population of sources (created by add_nonunifrom_ds).
- **data_object** (*nsdf.NonuniformData*) – NSDFData object storing the data for all sources in *source_ds*.
- **fixed** (*bool*) – if True, this is a one-time write and the data cannot grow. Default: False

Returns tuple containing HDF5 Datasets for the data and sampling times.

TODO: Concatenating old data with new data and reassigning is a poor choice. waiting for response from h5py mailing list about appending data to rows of vlen datasets. If that is not possible, vlen dataset is a technically poor choice.

h5py does not support vlen datasets with float64 elements. Change dtype to np.float64 once that is developed.

add_static_data (*source_ds*, *data_object*, *fixed=True*)
Append static data *variable* values from *sources* to *data*.

Parameters **source_ds** (*HDF5 Dataset*) –

the dataset storing the source ids under map. This is attached to the stored data as a dimension scale called *source* on the row dimension.

data_object (nsdf.EventData): NSDFData object storing the data for all sources in *source_ds*.

fixed (bool): if True, the data cannot grow. Default: True

Returns HDF5 dataset storing the data

Raises

- **KeyError if the sources in 'source_data_dict' do not match** –
- **those in 'source_ds'.** –

add_static_ds (*popname*, *idlist*)
Add the sources listed in idlist under /map/static.

Parameters

- **popname** (*str*) – name with which the datasource list should be stored. This will represent a population of data sources.
- **idlist** (*list of str*) – list of unique identifiers of the data sources.

Returns An HDF5 Dataset storing the source ids. This is converted into a dimension scale when actual data is added.

add_uniform_data (*source_ds*, *data_object*, *tstart=0.0*, *fixed=False*)
Append uniformly sampled *variable* values from *sources* to *data*.

Parameters

- **source_ds** (*HDF5 Dataset*) – the dataset storing the source ids under map. This is attached to the stored data as a dimension scale called *source* on the row dimension.
- **data_object** (*nsdf.UniformData*) – Uniform dataset to be added to file.
- **tstart** (*double*) – (optional) start time of this dataset recording. Defaults to 0.
- **fixed** (*bool*) – if True, the data cannot grow. Default: False

Returns HDF5 dataset storing the data

Raises

- **KeyError if the sources in 'source_data_dict' do not match –**
- **those in 'source_ds'. –**
- **ValueError if dt is not specified or <= 0 when inserting –**
- **data for the first time. –**

add_uniform_ds (*name*, *idlist*)
Add the sources listed in idlist under /map/uniform.

Parameters

- **name** (*str*) – name with which the datasource list should be stored. This will represent a population of data sources.
- **idlist** (*list of str*) – list of unique identifiers of the data sources.

Returns An HDF5 Dataset storing the source ids. This is converted into a dimension scale when actual data is added.

contributor
List of contributors to the content of this file.

description
Description of the file. A text string.

license
License information about the file. This is text string.

method
(numerical) methods applied in generating the data.

rights
The rights of the file contents.

set_properties (*properties*)
Set the file attributes (environments).

Parameters **properties** (*dict*) – mapping property names to values. It must contain the following keyes:

title (str) creator (list of str) software (list of str) method (list of str) description (str) rights (str) tstart (datetime.datetime) tend (datetime.datetime) contributor (list of str)

Raises **KeyError if not all environment properties are specified in the dict. –**

software
Software (one or more) used to generate the data in the file.

tend
End time of the simulation/recording.

title
Title of the file

tstart
Start time of the simulation / data recording. A string representation of the timestamp in ISO format

nsdf.nsdfwriter.**add_model_component** (*component*, *parentgroup*)
Add a model component as a group under *parentgroup*.

This creates a group *component.name* under parent group if not already present. The *uid* of the component is stored in the *uid* attribute of the group. Key-value pairs in the *component.attrs* dict are stored as attributes of the group.

Parameters

- **component** (*ModelComponent*) – model component object to be written to NSDF file.
- **parentgroup** (*HDF Group*) – group under which this component's group should be created.

Returns HDF Group created for this model component.

Raises

- **KeyError if the parentgroup is None and no group** –
- **corresponding to the component's parent exists.** –

nsdf.nsdfwriter.**match_datasets**(*hdfds*, *pydata*)
Match entries in hdfds with those in pydata. Returns true if the two sets are equal. False otherwise.

nsdf.nsdfwriter.**write_ascii_file**(*group*, *name*, *fname*, ***compression_opts*)
Add a dataset *name* under *group* and store the contents of text file *fname* in it.

nsdf.nsdfwriter.**write_binary_file**(*group*, *name*, *fname*, ***compression_opts*)
Add a dataset *name* under *group* and store the contents of binary file *fname* in it.

nsdf.nsdfwriter.**write_dir_contents**(*root_group*, *root_dir*, *ascii*, ***compression_opts*)
Walk the directory tree rooted at *root_dir* and replicate it under *root_group* in HDF5 file.

This is a helper function for copying model directory structure and file contents into an hdf5 file. If ascii=True all files are considered ascii text else all files are taken as binary blob.

Parameters

- **root_group** (*h5py.Group*) – group under which the directory tree is to be created.
- **root_dir** (*str*) – path of the directory from which to start traversal.
- **ascii** (*bool*) – whether to treat each file as ascii text file.

CHAPTER 6

NSDF file reader

6.1 nsdfreader Module

Reader for NSDF format

class nsdf.nsdfreader.**NSDFReader**(*filename*)

Bases: object

Reader for NSDF files.

This class encapsulates an NSDF file and provides utility functions to read the data in an organized manner.

contributor

List of contributors to the content of this file.

description

Description of the file. A text string.

event_populations

Names of the populations for which event variables have been recorded.

get_event_data(*population*, *variable*)

Get event variable recorded from population.

In NSDF a variable is recorded from a population of sources and data is organized as *population/variable*. This function retrieve this dataset and creates EventData object containing (source, data) pairs.

Parameters

- **population** (*str*) – name of the population from which this data was recorded.
- **variable** (*str*) – name of the variable this data represents.

Returns: nsdf.EventData

Note: Data is converted to float64 for VLEN dialect.

get_event_vars(*population*)

Returns the names of event variables recorded for *population*.

Parameters **population** (*str*) – name of the population.

Returns list of str: names of the groups storing event variables.

get_nonuniform_data(*population*, *variable*)

Get nonuniform data *variable* under *population*.

In NSDF a variable is recorded from a population of sources and data is organized as *population/variable*. This function retrieve this dataset and creates NonuniformData object containing (source, data) pairs. In case all the sources share the same sampling times, it is the NonuniformRegularData, a subclass of NonuniformData and contains the sampling times as a separate array. Otherwise, *data* is tuple of variable values and sampling times.

Parameters

- **population** (*str*) – name of the population from which this data was recorded.
- **variable** (*str*) – name of the variable this data represents.

Returns nsdf.NonuniformRegularData if dialect of the file is NUREGULAR. nsdf.NonuniformData otherwise.

Note: Data is converted to float64 for VLEN dialect.

get_nonuniform_vars (*population*)
Returns the names of nonuniform variables recorded for *population*.

Parameters **population** (*str*) – name of the population.

Returns list of str: names of the groups storing nonuniform variables.

get_uniform_data (*population*, *variable*)
Returns a UniformData object contents for recorded *variable* from *population*.

Parameters

- **population** (*str*) – name of the population.
- **variable** (*str*) – name of the variable.

Returns

dataobject – data container filled with

source, data, dt and units.

Return type nsdf.UniformData

get_uniform_dataset (*population*, *varname*)
Returns the data sources and data contents for recorded variable *varname* from *population*.

Parameters

- **population** (*str*) – name of the population.
- **varname** (*str*) – name of the variable.

Returns (sources, data): *sources* is an dataset containing the source identifiers and data is a 2D dataset whose i-th row is the data from the i-th entry in *sources*.

get_uniform_dt (*population*, *varname*)
Returns sampling interval and time-unit for the uniform dataset *varname* recorded from *population*.

Parameters

- **population** (*str*) – name of the population of sources.
- **varname** (*str*) – name of the recorded variable.

Returns (dt, unit) : *dt* is the sampling interval for this dataset and unit is a string representing the unit of time.

get_uniform_row (*srcid*, *field*)
Get the data for *field* variable recorded from source with unique id *srcid*.

Parameters

- **srcid** (*str*) – unique id of the source.
- **varname** (*str*) – name of the variable.

Returns (data, unit, times, timeunit)

get_uniform_ts (*population*, *varname*)
Returns an array of sampling times and time-unit for the uniform dataset *varname* recorded from *population*.

Parameters

- **population** (*str*) – name of the population of sources.
- **varname** (*str*) – name of the recorded variable.

Returns (times, unit) : times is an array of doubles containing the sampling time for each column of the dataset and unit is a string representing the unit of time.

get_uniform_vars (*population*)
Returns the names of uniform variables recorded for *population*.

Parameters **population** (*str*) – name of the population.

Returns list of str: names of the datasets storing uniform variables.

license
License information about the file. This is text string.

method
(numerical) methods applied in generating the data.

nonuniform_populations
Names of the populations for which variables have been recorded with nonuniform sampling.

rights
The rights of the file contents.

software
Software (one or more) used to generate the data in the file.

tend
End time of the simulation/recording.

title
Title of the file

tstart
Start time of the simulation / data recording. A string representation of the timestamp in ISO format

uniform_populations
Names of the populations for which variables have been recorded with uniform sampling.

CHAPTER 7

Utilities

7.1 `util` Module

Utility functions for nsdf.

`nsdf.util.`**`find`**(*a*, *predicate*, *chunk_size=1024*)

Find the indices of array elements that match the predicate.

Parameters

- **a** (*array_like*) – Input data, must be 1D.
- **predicate** (*function*) – A function which operates on sections of the given array, returning element-wise True or False for each data value.
- **chunk_size** (*integer*) – The length of the chunks to use when searching for matching indices. For high probability predicates, a smaller number will make this function quicker, similarly choose a larger number for low probabilities.

Returns **index_generator** – A generator of (indices, data value) tuples which make the predicate True.

Return type generator

See also:

`where()`, `nonzero()`

Notes

This function is best used for finding the first, or first few, data values which match the predicate.

Examples

```
>>> a = np.sin(np.linspace(0, np.pi, 200))
>>> result = find(a, lambda arr: arr > 0.9)
>>> next(result)
((71, ), 0.900479032457)
>>> np.where(a > 0.9)[0][0]
71
```

Author:

Phil Elson (https://github.com/pelson). Code taken from numpy issue tracker: https://github.com/numpy/numpy/issues/2269# on Wed Jul 30 11:20:23 IST 2014

nsdf.util.**node_finder**(*container_list*, *match_fn*)

Return a function that can be passed to h5py.Group.visititem to collect all nodes satisfying *match_fn* collect in *container_list*

nsdf.util.**printtree**(*root*, *vchar='|'*, *hchar='__'*, *vcount=1*, *depth=0*, *prefix=''*, *is_last=False*)

Pretty-print an HDF5 tree.

Parameters

- **root** (*h5py.Group*) – path of the root element of the HDF5 subtree to be printed.
- **vchar** (*str*) – the character printed to indicate vertical continuation of a parent child relationship.
- **hchar** (*str*) – the character printed just before the node name.
- **vcount** (*int*) – determines how many lines will be printed between two successive nodes.
- **depth** (*int*) – for internal use - should not be explicitly passed.
- **prefix** (*str*) – for internal use - should not be explicitly passed.
- **is_last** (*bool*) – for internal use - should not be explicitly passed.

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*

n

A

C

D

E

F

G

H

L

M

N

P

R

S

T

U

V

W