**Matlab API for Neurodata Without Borders (NWB) format**

**1. Overview.**

The Matlab API for the NWB format is a write API that can be used to create NWB files (it does not provide functionality for reading). It is implemented using a specification language and API that are domain-independent. The API provides a small set of generic functions for storing data in the file (that is, creating HDF5 groups and datasets). The specialization of the API to create NWB files is achieved by having the format defined using the specification language.

**2. Files.**

The Matlab API is implemented using five files:

1. File.m
2. Node.m
3. Group.m
4. Dataset.m
5. Utils.m
6. NWB\_file.m
7. nwb\_core.json

The first four classes, **File**, **Node**, **Group** and **Dataset** implement the API. They are domain and format independent in that it could be used to implement other formats both for neuroscience data and for other domains.

File **nwb\_core.json** contains a formal specification of the NWB format. It is written in a “specification language”. The specification language itself is also domain and format independent and is constructed using a JSON syntax.

Files **NWB\_file** class is used to setup the API and initialize a newly created NWB file. The **Utils** class contains utility functions that are useful when creating NWB files.

**3. Using the API.**

*3.1 Initialization.*

The initialization of a NWB file is performed using code like the following:

% Create the NWB file

fname='sample\_01.nwb';

start\_time='2015-07-27 8:34AM';

f = NWB\_file(fname, start\_time);

In the above code variable “fname” stores the name of the NWB file being created and start\_time is the starting time of the session. (Parameter start\_time is optional. If not specified, the current time is used).

The call to function nwb\_file returns a “NWB\_file” object. This is an object that controls the creation of an hdf5 file using the format specified by the specification language file. In the above call to NWB\_file, since only the file name is provided, the software uses the nwb\_core.json file for the format specification. NWB\_file has additional optional parameters that make it possible to specify other format specification files or multiple format specifications. This functionality allows extending the format by writing a specification for new format features and providing those in the call to NWB\_file. The extensions can be shared between labs and also stored in a central repository to allow forming a collection of commonly used extensions, some of which could be incorporated into the NWB “core” format. These optional parameters and writing extensions is described in section “Advanced Topics.”

3.2 Referencing groups and datasets

The NWB Matlab API works by allowing the user to sequentially create hdf5 groups and datasets that conform to the specification of the format. To reference a group or dataset in a call to the API, the name of the group or dataset as given in the file format specification is used. Some groups and datasets have a name, which is variable, that is, which is specified in the call to the API rather than in the format specification. (For example, group “electrode\_X” in the general/intracellular\_ephys group.) In the API calls, groups or datasets that have a variable name are referenced by enclosing the identifier associated with them in angle brackets, e.g. “<electrode\_X>”.

3.3 File and Group objects.

The call to function NWB\_file returns a “File” object (stored in variable “f” in the above example). Methods of this object are used to create groups and datasets in the hdf5 file that are, figuratively speaking, at the “top-level” of the file format specification, that is, not located inside groups that are defined in the specification language. Calls to the API functions which creates groups, return an “Group” object. This object has methods that are used to create groups and datasets within the associated hdf5 group. There is also a method (set\_attrs), which can be used set attributes associated with groups or datasets.

The methods of the File and Group objects and the set\_attrs function are described in the following sections.

3.4 File object methods.

The File object has the following methods (functions):

1. make\_group
2. make\_custom\_group
3. set\_dataset
4. set\_custom\_dataset
5. get\_node
6. close

*3.4.1 File make\_group*

Method “make\_group” of object File creates a group in the hdf5 file. It has the following signature:

g = f.make\_group(qid, name, path, attrs, link, abort)

“f” signifies an File object. The returned object, stored in variable ‘g’ in the above, is a “ Group” object—which is described in the section 3.5. In the make\_group function, only the first argument (qid) is always required. The second two arguments (name and path) are sometimes required. All arguments are described below:

*qid,* is the “qualified id” for the group. The qualified id is the name used to reference the group (with surrounding angle brackets if the name is variable) optionally prefixed with a “namespace.” The namespace provides a way to associate extensions to the format with an identifier, in a manner similar to how namespaces are used in XML. This is described in the Advanced section. For normal uses (without extensions) the qid will just be the group name as given in the format specification.

*name* is only used if the group name is variable (referenced using <angle brackets>). It contains the name to be used when creating the group.

*path* specifies the path to the parent group within the hdf5 file. It is only needed if the location of the group within the file is ambiguous. For many groups, the location is not ambiguous and for those groups, the location is automatically determined by the API, without requiring a specification by argument “path”.

*attrs* is a matlab cell containing hdf5 attributes keys and values in alteration to assign to the created group. It is optional.

*link* is used to create a hdf5 link to an existing group. If present it contains either a previously created Group object, or a string of the form “link:/path/to/group” or a string of the form: “extlink:path/to/file, path/to/group”. The first two are used to make a hdf5 link to a group within the current file. The third method specifies a link to a group in an external hdf5 file.

*abort* is a logical variable, default is True. It controls the program behavior if the group being created already exists. If abort is True the program will abort. If False, the function will return the previously existing group (Group object).

*3.4.2 File make\_custom\_group*

Method “make\_custom\_group” of object File creates a custom group in the hdf5 file (that is, a group that is not part of the format specification). This method is provided because method “make\_group” can only be used to create groups that are specified in the file format. The function signature for “make\_custom\_group” is:

g = f.make\_custom\_group(qid, name, path, attrs)

Return type and parameters are the same as for File method “make\_group”. If parameter “path” is not specified or is a relative path, then the custom group will be created in the default custom location, which is inside group “/general”.

*3.4.3 File set\_dataset*

Method “set\_dataset” of object File is used to create and store data in an hdf5 dataset. It has the following signature:

d = f.set\_dataset(qid, value, name, path, attrs, dtype, compress)

The return value is an object of type Dataset. The arguments are described below:

*qid* - the “qualified id” for the dataset. The qualified id is the name used to reference the dataset (with surrounding angle brackets if the name is variable) optionally prefixed with a “namespace” as described in the qid parameter for method make\_group.

*value* - value to store in the dataset. To store numeric or string values in the dataset (what is normally done) the value can be a scalar, array, char, cellstr, or a cell with one uniform type. To have the created dataset be a link to another Dataset, the value is set to a Dataset object or a string matching the pattern: link:/path/to/dataset (to link to a dataset within the file) or “extlink:path/to/file, /path/to/dataset” to link to a dataset in an external file.

*name* - name of the dataset in case the name is unspecified (qid is in <angle brackets>).

*path* - specified path of where to create the dataset (path to parent group). Only needed if the location of where to create the dataset is ambiguous in the format specification.

*attrs* - a dictionary containing hdf5 attributes keys and values to assign to the created group. It is optional.

*dtype* – type of data. This is currently not considered in the Matlab API. A type can be set by casting the input to the desired type before calling set\_dataset.

*compress* - if True, compression is specified in the call to the library routine which creates the dataset (h5py.create\_dataset). The default value is False. It is recommended that this be set True when saving large datasets in order to reduce the size of the generated file.

*3.4.4 File set\_custom\_dataset*

Method “set\_custom\_dataset” of object File creates a custom dataset in the hdf5 file (that is, a dataset that is not part of the format specification). The function signature is:

d = f.set\_custom\_dataset(qid, value, name, path, attrs, dtype, compress)

Return type and parameters are the same as for method “set\_dataset”.

*3.4.5 File get\_node*

Method “get\_node” returns the Group or Dataset object located at the specified location (full path in the hdf5 file). It has the following signature:

n = f.get\_node(full\_path, abort)

Arguments are:

*full\_path* – absolute path to group or dataset.

*abort* – A logical value that specifies what to do if there is no node (group or dataset) at the specified path. Default is True, which causes the program to abort. A value of False, causes the function to return None.

*3.4.6 File close*

Method “close” of object File is used to close the created file. It must be called to complete the creation of the file. Function signature is:

f.close()

There are no arguments.

3.5 Group methods.

A Group object is returned by File methods “make\_group” and “make\_custom\_group”. The Group object has the following methods:

1. make\_group
2. make\_custom\_group
3. set\_dataset
4. set\_custom\_dataset
5. set\_attr

The names of the first four of these methods are the same as the name of methods of the File object. The difference between the File and Group object methods is that the File methods are used to create groups and datasets that are not inside groups that are defined as part of the format specification whereas the Group methods are used to create groups and datasets inside the current group, that is, inside the Group object used to call the methods. The Group methods are described below:

*3.5.1 Group make\_group*

Method “make\_group” of object Group creates a group inside the current group. It has the following signature:

g = pg.make\_group(qid, name, attrs, link, abort)

In the above line, “pg” signifies a parent group, (object of type Group). The returned object, stored in variable ‘g’ in the above, is also a “ Group” object. Parameters in the function have the same meaning as those in function File make\_group. There is no “path” parameter (which was in the File object make\_group) because the location of the created group is always known. (The created group will be located inside the parent group used to invoke the method).

*3.5.2 Group make\_custom\_group*

Method “make\_custom\_group” of object Group creates a custom group, usually within the parent group. The function signature for “make\_custom\_group” is:

g = pg.make\_custom\_group(qid, name, path, attrs)

Return type and parameters are the same as for Group method “make\_group”. If path is not specified or is a relative path, the group will be created inside the parent group. If path is an absolute path, the group will be created at the specified location which can be anywhere in the hdf5 file.

*3.5.3 Group set\_dataset*

Method “set\_dataset” of object Group is used to create a dataset within the parent group. It has the following signature:

d = pg.set\_dataset(qid, value, name, attrs, dtype, compress)

The return value is an object of type Dataset. The parameters have the same meaning as those in function File set\_dataset. There is no “path” parameter because the created dataset is always located inside the parent group used to invoke the method.

*3.5.4 Group set\_custom\_dataset*

Method set\_custom\_dataset of object Group is used to create a custom dataset, usually within the parent group. It has the following signature:

d = pg.set\_custom\_dataset(qid, value, name, path, attrs, dtype, compress)

The return value is an object of type Dataset. The parameters have the same meaning as those in function Group set\_dataset. If path is not specified or is a relative path, the group will be created inside the parent group. If path is an absolute path, the group will be created at the specified location.

3.5.5 set\_attr method

Both the Group and Dataset objects have a method “set\_attr” which is used to set the value of an hdf5 attribute of the group or dataset. It has the following signature:

n.set\_attr(aid, value, custom)

In the above, “n” is a Node object (which is a Group or Dataset).

Parameters are:

*aid* – attribute id (name of the attribute).

*value* – value to store in the attribute.

*custom* – a logical value (default “False”) which indicates whether or not the attribute is a custom attribute (that is, not part of the file format specification). Setting a value of “True” when setting a custom attribute prevents a warning message from being displayed for the attribute when closing the file.

3.6. nwb\_utils.py functions

File nwb\_utils.py provides a variety of utility functions for different purposes.

Below is a list of functions, which have been used to create some of the very original python scripts:

function content = load\_file(filename)

function add\_roi\_mask\_pixels(seg\_iface, image\_plane, name, desc, pixel\_list, weights, width, height, start\_time)

function add\_roi\_mask\_img(seg\_iface, image\_plane, name, desc, img, start\_time)

function add\_masks(seg\_iface, image\_plane, name, desc, pixel\_list, weights, img, start\_time)

function add\_reference\_image(seg\_iface, plane, name, img)

function add\_epoch\_ts(e, start\_time, stop\_time, name, ts)

function [start\_idx, cnt] = get\_ts\_overlaps(tsg, start\_time, stop\_time)

function [lower\_index,upper\_index] = myFindDrGar(x,LowerBound,UpperBound)

These functions are provided as a reference. They might be useful for some users to create scripts, but most of them are untested at this point in time.

The other functions are used internally in the API to address specific Matlab needs. They should not be necessary to generate conversion scripts.

Matlab needs some additional helper functions to handle special characters in the specification language. Certain characters like ‘<’ or ‘?’ cannot be used as fieldnames in structs and have to be converted to their utf equivalent. To make these fieldnames human readable they have to be converted back to Unicode. The following two functions handle these conversions:

function new\_str = convert\_uc\_to\_utf( old\_str )

function new\_str = convert\_utf\_to\_uc( old\_str )

Matlab handles argument input in a different way than Python. The following function accounts for that:

function [arg\_vals] = parse\_arguments(args, arg\_names, arg\_types, arg\_default)

There are a variety of functions, which are needed to create and handle different hdf5 objects:

function create\_softlink(fid, targetpath, path, name)

function create\_external\_link(fid, path, targetpath, targetobj)

function write\_att(fid, filename,path,attname,attvalue)

function exist = node\_exists(fid, path)

function gid = create\_group(fid, path)

function create\_string\_dataset(file\_obj,path, name, dataset, compression)

function create\_numeric\_dataset(file\_obj, path, name, dataset, compression)

**Advanced topics**

\* To be added \*

The use of angle brackets to indicate names that are variable is also used in the specification language. In the specification language, groups are specified by an identifier followed by a slash, i.e. “/”. An identifier that is not followed by a slash specifies datasets. Groups and datasets that have a name which is variable (that is, the name is specified in the call to the API rather than in the format specification) are specified by an identifier enclosed in angle brackets, i.e. <>. Examples are below using the identifier “foo”:

foo – dataset, name is “foo”

foo/ – group, name is “foo”

<foo> – dataset, name specified by API call

<foo>/ – group, name specified by API call