**Specification language used for Neurodata Without Borders (NWB) format**

Version 1.0 (August 5, 2015)

**1. Introduction.**

Both the Python and MATLAB API for the NWB format are implemented using a domain-independent specification language. The specification language allows defining the structures (groups and datasets) that make up a format. The API software automatically provides a write-API based on the specification. The API currently uses HDF5 for storing the data, but other storage methods are possible.

*1.1 Namespaces*

The specification language is written using a Python dictionary in a JSON-like syntax, which can easily be converted to JSON. The top level of a specification has the following format:

{ "ns1": <specification for ns1 namespace>,

"ns2": <specification for ns2 namespace>,

"ns3": <specification for ns3 namespace>, ... }

Each format is described by one or more specifications, each of which is associated with a unique namespace identifier. The namespace identifiers (ns1, ns2, ns3 in the above) are the keys in the dictionary. The specifications for each namespace identifier are the values. The namespace identifier can be any valid Python identifier (the identifiers are *not* restricted to start with ‘ns’). One of the namespaces is designated as the “default” namespace. The default namespace is the namespace that is used by default if no namespace is specified when making an API call that references a group or dataset defined in the specification. The dictionary of definitions (and/or where to load them from) and default namespace identifier are passed into the API software when calling the API to create a new file.

The specification associated with each namespace is a Python dictionary with two components: structures and locations. e.g.:

{ "structures": <specification of structures>,

"locations": <specification of locations> }

The structures specification defines the groups, datasets and relationship that make up the format. The locations specification, indicates where groups and datasets defined in the structures specification are stored in the file. These are described in more detail below.

*1.2 Structures specification*

The structures specification consist of a Python dictionary where each key is an identifier either followed by a “/” (to indicated a group) or not followed by a slash (to indicate a dataset). Identifiers include surrounding angle brackets to indicate the name of the group or dataset is variable (that is, specified through an API call when creating the group or dataset). If the identifier does not have surrounding angle brackets, then the name of the group or dataset is fixed and is the same as the key. The four possibilities (group or dataset, variable name or fixed name) are illustrated below:

foo – dataset, name is “foo”

foo/ – group, name is “foo”

<foo> – dataset, variable name

<foo>/ – group, variable name

**2. Specification of groups**

*2.1 Overall form*

The specification of a group (i.e. value associated with a key that has a trailing slash) is a Python dictionary with the following form:

{

"description": "<description of group>",

"\_description": "<description of group in case there is a dataset named description>",

"attributes": <attributes specification>,

"parent\_attributes": <parent attributes>,

"merge": [ <list of groups to merge> ],

"include": [ <dictionary of structures to include> ],

"dataset\_id[qty\_flag]": { <dataset specification> },

"group\_id/[qty\_flag]": { <group specification> }

}

None of the keys value pairs are required. The first six keys above (“description”, “\_description”, “attributes”, “parent\_attributes”, “merge”, “include”) are described in section “Group specification keys”. The last two ("dataset\_id", and "group\_id/") are used to specify a group or dataset inside the group. The specification for them is the same as the specification for top-level groups (described in this section) and for top-level datasets (describe later). There can be any number of groups or datasets specified inside a group.

*2.2 Quantity specifier*

The key for groups or datasets that are defined inside a group can be have a final character (denoted by “[qty\_flag]” above), that indicates whether or not the group or dataset is required and for a those with a variable name, whether or not multiple instances of the group or dataset are allowed. The characters, and meaning of them, match those used in regular expressions. They are given below:

! – Group or dataset required (this is the default)

? – Group or dataset not required

+ - One or more instance of variable-named group or dataset required

\* - Zero or more instances of variable-named groups or datasets allowed

*2.3 Group specification keys*

The following sections describes the first six keys in the illustrated group specification above (“description”, “\_description”, “attributes”, “parent\_attributes”, “merge”, “include”).

*2.4 description*

The value of the group specification “description” key is a string describing the group.

*2.5 \_description*

The key “\_description” (has an underscore in front) is used in place of “description” in case the key “description” is used to specify a dataset in the group named “description”.

*2.6 attributes*

The value of the group specification “attributes” key is a Python dictionary of the following form:

{

"attribute\_name\_1": <specification for attribute\_name\_1>,

"attribute\_name\_2": <specification for attribute\_name\_2>,

... }

The keys are the attribute names. Any valid Python identifier can be used. The values are the specification for the attributes. Each attribute specification has the following form:

{

"data\_type": <float, int, number, or text>,

"description': '<description of attribute>',

"value": <value to store>

}

Only data\_type is required. The value for data\_type is a string specifying the data\_type of the attribute. Allowable values are:

float– indicates a floating point number

int – indicates an integer

number – indicates either a floating point or an integer

text – a text string

Currently, only scalar data types can be specified for attributes. However, the API allows array types to be stored.

The description is a text string describing the attribute. The value is the value to store in the attribute. Regardless of whether or not a value is specified in the specification language file, a new value can be provided through the API in the call to make\_group or to “set\_attr”.

*2.7 parent\_attributes*

The parent attributes specification allows specifying attributes that are added to the parent group (that is, the group containing the created group) when the specified group is created. It has the same format as the group “attributes” specification described above. For parent\_attributes that have data\_type “text”, the value can be prefixed by a plus character (“+”), to indicate that the value should be appended to any value of the parent group attribute, but separated by a comma. If the data type is not type text, or if there is not a prefixed plus character, then the parent group attribute is overwritten by the value specified in the child group parent\_attribute.

*2.8 merge*

The merge specification is used to merge the specification of other groups into the current group. It consists of a Python list of the groups to merge. (Each element of the list must have a trailing slash since they all must be groups).

*2.9 include*

The include specification is used to include the specification of a group or dataset inside the current group. The format is a Python dictionary, in which each key is the key associated with a group or dataset to include and the values are a dictionary of properties and values that are merged into the properties of the included structure. The key that designate the group or dataset to include may have a final character that specifies a quantity (same as described in section 2.2).

*2.10 merge vs. include*

The merge operation implements a type of subclassing because properties of the merged in groups (the superclasses) are included, but overridden by properties in the group specifying the merge if there are conflicts. The include specification implements a type of reuse. The merge and include operations are illustrated by the following diagram:

|  |  |
| --- | --- |
| Merge – (for subclassing)  "A/": {  "x": ...,  "y": ...,  }    "B/": {  "merge": ["A/",],  "m": ...,  "n": ...,  }    Result:  "B/": {  "x": ...,  "y": ...,  "m": ...,  "n": ...,  } | Include – for reuse  "A/": {  "x": ...,  "y": ...,  }    "B/": {  "include": {"A/": {}},  "m": ...,  "n": ...,  }    Result:  "B/": {  "m": ...,  "n": ...,  "A/": {  "x": ...,  "y": ...,  }  } |

**3. Specification of datasets**

*3.1 Overall form*

The specification of a dataset (i.e. value associated with a structure specification key described in section 1.2 that does not have a trailing slash) is a Python dictionary with the following form:

{

"description": "*<description>*",

"data\_type": ("int", "float", "number", or "text"), # required

"dimensions": ["dim1", "dim2", ...], # required if not scalar

"unit": "<*unit*>",

"scale": <*scale factor*>,

"references": "<*reference target specification*>",

"semantic-type": '*<type of data, e.g. timestamp>*'

"dim1": *<dimension specification>*,

"dim2": *<dimension specification>*,

...

}

Only the data\_type property is always required. If the dataset is an array (not scalar) than the dimensions property is required. If dimensions are listed, the definition of the dimensions may be specified by a value associated with the name of the dimension (e.g. ‘dim1’ and ‘dim2’ in the above form. The properties and dimension definitions are described below.

*3.2 description*

A string describing the dataset.

*3.3 data\_type*

A string indicating the type of data stored. Valid types are: int, float, number or text.

*3.4 dimensions*

A list of dimension names if the value stored in the dataset is an array (i.e. not a scalar). Each dimension name is an identifier (giving a dimension name) or a integer (specifying the size of the dimension). Dimensions names are used both for specifying properties of dimensions (as described below) and for specifying relationships between datasets.

*3.5 unit*

A string specifying the unit associated with numeric data stored in the dataset. For example: “Volt”.

*3.6 scale*

A scale factor to convert stored values to the units of measure. Any value specified should be numeric and and is copied to the 'conversion' attribute of the created dataset.

*3.7 references*

The references property is used to indicate that the values stored in the dataset are referencing groups, datasets or parts of other datasets in the file. The value of the references property is a reference target specification. This has one of the following four forms:

1. <path\_to\_dataset>.dimension
2. <path\_to\_dataset>.dimension.component
3. <path\_to\_group>/<variable\_node\_id>
4. /

The first form (a) specifies a reference to a particular dimension of a dataset. In this case all values in the referencing dataset should be integers that are equal to one of the indices in the referenced dataset dimension.

The second form (b) specifies a reference to a particular component of a structured dimension. Structured dimensions are described in the section about dimension specifications. In this case each value in the referencing dataset should be equal to a value in the referenced component of the referenced dataset and the values of the component in the referenced dataset should all be unique. This case corresponds to foreign key references in relational databases with the referenced component being an column in the referenced table satisfying a uniqueness constraint.

The third form (c) allows referencing variable named groups or datasets. In this case all values of the referencing dataset should be names of groups or datasets that are created with the name specified in the call to the API. The value of the reference target specification should contain the name of the group or dataset in angle brackets (since the name is variable) and have a trailing slash if it is a group (since groups are designated by a slash after the name).

The forth form (d) is a single slash. This form is to indicate that the values in the referencing dataset much link to a group or dataset somewhere in the file, but there are no other constraints.

*3.8 semantic type*

The semantic type is provided to allow a string to be included which indicates the type of data stored in the dataset. It is not used in the specification language API software but could be useful for applications using the data.

*3.9 dimension specification*

Within a dataset specification, there are two types of dimension specifications. The first, described in section 3.4, provides a list of the names of all dimensions in the dataset. The second (described in this section) provides a way to describe the properties of each dimension. It is not necessary to include the specification for all dimensions. Only those dimensions that have structured components (which are described below) need to be specified. These dimension specifications have a key equal to the name of the dimension, and the value is the specification of the properties of the dimension. The following format is used:

{

"type": ("struct"),

# for dimension type struct:

"components": [

{ "alias": "var1",

"unit": "<unit>",

' "references": "<*reference target specification*>"},

{ 'alias': 'var2', ... }, ... ]

}

The type specifies the type of dimension. Currently there is only one type implemented, named “struct”. Type struct is a structure type which allows storing different types of data into a single array similar to columns in a spreadsheet or fields in a relational data base table. This is also similar to the “metaarray” described in the SciPy cookbook: http://wiki.scipy.org/Cookbook/MetaArray and also Pandas DataFrame: http://pandas.pydata.org/pandas-docs/dev/index.html).

The different components are specified using a list of dictionaries, with each dictionary specifying the properties of the corresponding component. The “alias” specifies the component name that can be referenced in a <*reference target specification>*

(reference type “b” in section 3.7). “unit” allows specifying the unit of measure for numeric values. “references” allows specifying that the values in the component reference another part of the file using any of the methods described in section 3.7.

**4. Relationships**

Relationships are specified in one of two ways:

1. By sharing a common dimension identifier. Two arrays that are in the same group which have a common dimension identifier are related to each through a direct mapping between the two dimensions. This is equivalent to each dimension being a foreign key to the other in a relational database.
2. Through references specifications in dataset specifications that are described in section 3.7.

**5. locations specification**

As described in section 1.1, the locations specification specifies where in the hdf5 the groups and datasets defined in the structures specification are stored. The structures specification is a Python dictionary with the following format:

{ *location-1*: [ list of groups/datasets ],

*location-2*: [ list of groups/datasets ], ...}

Each key in the dictionary (location-n in the above) is either an absolute path in the hdf5 file, or the name of a group defined in the structures section. The “list of groups/datasets” associated with each location are those groups and datasets that are defined in the structures section which can be stored at the location specified by the key.

The special identifier "\_\_custom" is used to indicate custom groups or datasets (that is, groups or datasets that are not defined in the specification language file, but are created by calling the API methods to make a custom group or dataset). Identifier "\_\_custom" is placed in the list corresponding to the location of where custom groups and datasets are created by default (that if, if the path is not specified in the API call).

In the list of groups and datasets, each group or dataset identifier may be followed by a quantity specification as allowed for groups and datasets defined inside groups as described in section 2.2. However, in the locations specification, the default value if no quantity is specified is the “?” character (not required).