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

Version 1.1.3 (June 8, 2016)[[1]](#footnote-1)

**1. Introduction.**

Both the Python and MATLAB API for the NWB format are implemented using a domain-independent specification language. The specification language allows specifying a schema that defines a format. The API software automatically provides a write-API based on the specification and also a validator that is used validate that data files are consistent with the format. The API currently uses HDF5 for storing the data, but other storage methods are possible.

The specification language is written using a Python dictionary in a JSON-like syntax, which can easily be converted to JSON. Python is used rather than pure JSON because python allows inserting comments and also provides more readable ways to include long strings.

* 1. *Schema Id (or ‘namespace’)*

The top-level of a format specification has the following form:

{"fs": {

"ns1": <specification for ns1 namespace>,

"ns2": <specification for ns2 namespace>,

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

The top level variable must be “fs”. (This stands for “format specification”). The value of variable “fs” is a dictionary with each key the “namespace” or “schema-id” of a format specification that is associated with that namespace. 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 and it is associated with the core format. Other namespaces (schema-ids) are associated with extensions to the core format. Information indicating where to obtain the specifications (usually names of files containing the specifications) and the default namespace identifier are passed into the API software when the software is initialized.

*1.2 Top level components*

The specification associated with each schema-id is a Python dictionary with three keys: info, schema, and doc. e.g.:

{ "info": <info specification>,

"schema": <schema specification>,

"doc": <auxiliary documentation>,

}

“info” and “schema” are required. “doc” is optional. <info specification> has the following form:

{

"name": "Name of specification",

"version": "<version number>",

"date": "<date last modified or released>",

"author": "<author name>",

"contact": "<author email or other contact info>",

"description": “description of the specification”

},

The schema specification section defines the groups, datasets and relationship that make up the format. This is the main part of the format specification. It is described in the following sections.

The <auxiliary documentation> section is for text that is added to documentation about the format that is generated from the format specification, using the “make\_docs.py” tool. This section is not described further here, but the structure and operation can be deduced by examining this part of the NWB format specification (e.g. file “nwb\_core.py”) and the generated documentation for the NWB format.

*1.3 Schema specification*

The schema specification consist of a Python dictionary where each key has the following form:

[ *absolute\_path* ] *identifier* [*quantity-flag*]

absolute\_path is optional. If present, it starts with a slash, and specifies the absolute location within an HDF5 file of the group or dataset. For the root group, the absolute path is empty and the identifier is “/”.

identifier is required. Identifiers that start with “<” and end with “>” or “>/”, e.g. have surrounding angle brackets, indicate that 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 is fixed and is the same as the identifier (but without a trailing slash). If the last character of the identifier is a slash “/” (after any angle brackets), then the identifier is associated with a group, otherwise a dataset.

*quantity\_flag* is optional. It is used to indicate the quantity. If present, it is a single character, one of: “?”, “!”, “^”, “+” or “\*”. These mean:

! – Required (this is the default)

? – Optional

^ – Recommended

+ - One or more instances of variable-named identifier required

\* - Zero or more instances of variable-named identifier allowed

Some example identifiers and their meaning are given below:

Unspecified location (no leading slash):

foo – dataset, name is “foo”

foo/ – group, name is “foo”

<foo> – dataset, variable name

<foo>/ – group, variable name

Specified location (has leading slash). Meaning same as above, but location specified.

/some/path/foo – dataset, name is “foo”, located at /some/path/

/some/path/foo/ – group, name is “foo”, located at /some/path/

/some/path/<foo> – dataset, variable name, located at /some/path/

/some/path/<foo>/ – group, variable name, located at specified path

With quantities:

foo? – dataset, name is “foo”. Is optional

foo/^ – group, name is “foo”. Is recommended

<foo>\* – dataset, variable name, zero or more allowed

<foo>/+ – group, variable name, at least one required

When an absolute path is specified (or if the identifier is for the root group) the identifier is “anchored” to the specified location. If there is no absolute path, then the group or dataset associated with the identifier can be incorporated into other groups using the “include” or “merge” directives that are described below.

1.4: Extensions:

As mentioned, extensions to the core format are specified using schema\_ids that are different from the schema\_id used for the core format. The way that extensions are implemented is very simple: The schema specified in extensions are simply “merged” into the schema specified in the core format based on having the same absolute path (if given) and the same identifier. For example, if the core format schema includes key “<foo>/” (specifying a group with a variable name “foo”) and an extension also includes a key “<foo>/”, the value associated with both of these (which must be a dictionary) are combined to form the specification of the core format and the extension. While it’s possible to define multiple extensions in the same file (as illustrated in section 1.1) normally, the specification associated with each schema\_id will be in a separate file as illustrated below:

File containing specification for core format:

{"fs": {

"core": <specification for core format>

}

File containing specification for extension 1:

{"fs": {

"ex1": <specification for extension ex1>

}

File containing specification for extension 2:

{"fs": {

"ex2": <specification for extension ex2>

}

**2. Specification of groups**

*2.1 Overall form*

The specification of a group (i.e. value of a schema specification identifier 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>,

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

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

"link": <link specification>,

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

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

}

None of the key-value pairs are required. The first six keys above (“description”, “\_description”, “attributes”, “merge”, “include”, ‘link”) are described in the next 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. The quantity-flag which can be specified for the groups and datasets has the same possible values and meaning as described in section 1.3.

*2.2 Group specification keys*

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

*2.3 description*

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

*2.4 \_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”.

*2.5 attributes*

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

{

"attribute\_name\_1[qty\_flag]": <specification for attribute\_name\_1>,

"attribute\_name\_2[qty-flag]": <specification for attribute\_name\_2>,

... }

The keys are the attribute names, optionally followed by a “qty\_flag.” The ‘qty\_flag’ (stands for ‘quantity flag’ is similar to that for groups and data sets. It specifies if the attribute is required (“!”) – the default, optional (“?”) or recommended (“^”). The value of each key is the specification for that attribute. Each attribute specification has the following form:

{

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

"dimensions": <dimensions list>,

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

"value": <value to store>,

"const": <True or False>,

"autogen": <autogen specification>,

}

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

uint – unsigned integer

number – indicates either a floating point or an integer

text – a text string

For all of the above types except number, a default size (in bits) can be specified by appending the size to the type, e.g., int32. If “!” is appended to the default size, e.g. “float64!”, then the default size is also the required minimum size,

If the attribute stores an array, the <dimensions list> specifies the list of dimensions. The format for this is the same as the <dimensions list> for data sets which is described in section 3. If no <dimension list> is given, the attribute stores a scalar value.

The description is a text string describing the attribute. The value is the value to store in the attribute. If a value is specified and “const”:True is specified,, then the value is treated as a constant and cannot be changed by the API. The autogen specification is described in Section 4.

*2.6 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 (identifiers described in section 1.3) to merge. (Each element of the list must have a trailing slash since they all must be groups).

*2.7 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 used to specify properties and values that are merged into the included structure and also options for the include. 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). Options for the include are specified by key “\_options”. Currently, there is only one option: “subclasses” which has value True to indicate that “subclasses” of the included group should also be included. Subclasses of a group are groups that inherit from a base group using the “merge” directive (described in the next section). Some examples of the include directive are shown below:

"include": { # include with subclasses

"<TimeSeries>/\*":{"\_options": {"subclasses": True}},

}}

"include": {"<TimeSeries>/\*": {}} # include without subclasses

*2.8 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": ...,  }  } |

*2.9 link*

The link specification is used to indicate that the group must be hdf5 link to another group. (Hard or Soft links can be used, but Soft links are recommended). The link specification is a Python dictionary. It has the following form:

{

"target\_type": "*<type\_of\_target>*",

"allow\_subclasses": <True or False">,

}

“target\_type” specifies the key for a group in the top level structure of a namespace. It is used to indicate that the link must be to an instance of that structure. “allow supclasses” is set to True to indicate the link can be to subclasses of the target structure. Subclasses are structures that include the target using a “merge” specification. Neither of the keys are required. The default value for “allow\_subclasses” is “False”. If target type is not specified, then the link can be to any group.

**3. Specification of datasets**

*3.1 Overall form*

The specification of a dataset (i.e. value associated with an identifier described in section 1.3 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": <dimensions list>, # required if not scalar

"attributes": <attributes specification>,

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

"link": <link specification>,

"autogen": <autogen specification>,

"dim1": *<dimension specification>*,

"dim2": *<dimension specification>*,

...

}

Either the data\_type or link property must be present All others are optional. If the dataset is specified and is an array (not scalar) than the dimensions property is required. The autogen specification is described in Section 4. Others are described below.

*3.2 description*

A string describing the dataset.

*3.3 data\_type*

A string indicating the type of data stored. This is the same as the data type for attributes, described in section 2.5.

*3.4 dimensions*

If present, <dimension\_list> is either a list of named dimensions, e.g.: [“dim1”, “dim2”, ...], or a list of lists of named dimensions, e.g.: [[“dim1”], [“dim1”, “dim2”]]. The first form is used if there is only one possibility for the number of dimensions. The second form is used if there are multiple possible number of dimensions. 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 *Attributes*

Dataset attributes are specified in the same was as group attributes, described in Section 2.5.

*3.6 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.

3.9 *link*

The link specification is used to indicate that the dataset must be implemented using a hdf5 link. Either hard or soft links can be used, but soft links are recommended because they indicate the source and target of the link). The link specification is a Python dictionary. It has the following form:

{

"target\_type": "*<type\_of\_target>*",

}

“target\_type” specifies the identifier for a dataset in the top level structure of a namespace. It is used to indicate that the link must be to an instance of that structure. If target type is not specified, then the link can be to any dataset id.

*3.10 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. Autogen

The autogen specification is used to indicate data that are can be derived from the structure of the hdf5 file and automatically filled in by the API. An API may use the autogen specification to automatically generate the values when creating a file, and to ensure that correct values are stored when validating a file. The autogen specification has the following form:

{ ‘type’: <type of autogen, one of: ‘links’, ‘link\_path’, ‘names’, ‘values’, ‘length’, ‘create’, ‘missing’>

‘target’: <path\_to\_target>,

‘trim’: True or False, default ‘False’

‘qty’: <Either ‘!’ – exactly one, or “\*” – zero or more. Default “\*”.

‘tsig’: <Signature of target>

‘include\_empty’: True or False. Default False,

‘sort’: True or False. Default True

‘format’: <link\_path\_format>

}

The type is the type of autogen. They are described below. For all types, except ‘create’ and ‘missing’ the “target” is required. All others are optional. For all types except “create”, “<path\_to\_target>” is a path of identifiers that specifies one or more groups or datasets that are descendant of the group that most directly contains the autogen specification). To specify multiple members the target path would have one or more variable-named id’s. (enclosed in <>). For type “create”, “<path\_to\_target>” is a list of members inside the group containing the autogen. In addition, the target “<\*>” indicates any group or dataset. If include\_empty is true, then if there are no values found that would be used to fill the autogen, the value is set to an empty list. Otherwise, the container for the autogen values (attribute or dataset) is not created. The “tsig” value is a “target signature” which is used to specify properties that must be satisfied for matching target(s). It is used to filter the nodes found at the target path to only those for which the autogen should apply. The format of tsig is:

{ “type”: <’group’, ‘dataset’>,

“attrs”: { “key1”: <value1>, “key2”: <value2, ... },

# Possible future: “level”: <integer> }

One of “type” or “attrs” is required (both may be present). “type” specifies the type of the target node. If not included, either group or dataset match. “attrs” specifies the attribute keys and values that can be compared to attributes in the target to detect a match. “level” is the number of matches (total of with type and each attribute) in order to consider the target a match (to be processed by the autogen). If level is not specified, then all characteristics specified in the tsig much match. (Level is currently not implemented. But could be if there is a need for it).

‘links’ indicates that the value of autogen is a list of paths that link to the group or dataset specified. If “trim” is True then when the paths are stored, if they all share the same trailing component of the path, e.g. /foo/bar/baz, and /x/y/baz; both share final component “baz”), then the common final component is trimmed from the paths before using them to fill in the data. If “sort” is true, values must be sorted.

“link\_path” indicates that the value of the autogen is the path of a link made from the referenced group or dataset. For example, if there is a group “foo” that links to “bar”, and a dataset named “baz” at the same level, defined by:

“baz”: {“autogen”: {“link\_path”: “foo”}}

Then when the file is created by an API that implementing the autogen, the value of baz to be the path to bar.

The “format” option allows specifying a formatting string used for “link\_path”. It can include strings: “$s” to indicate the source of a link and “$t” to indicate the target. If present, the format is used to create the “link\_path” entries. Default format is: “$t” (include just the target path). Another common format is “$s is $t” which will generate strings like: “<source> is <target>. If the ‘qty’ for “link\_path” is “

“names” – specify that the names of groups and/or datasets referenced are included as an array. If “sort” is True, the values must be sorted.

“values” – specify that values stored in each target data set are to be listed as a set (no duplicates) --in sorted order (if sort is True). The values in each data set must be an array of strings.

“length” – specifies that the value stored is the length of the target which must be a dataset storing a 1-D array.

“create” – provides a way to automatically create members (groups or datasets). For the create autogen, target is a list of members within the group containing the autogen. If these members are required, and if they do not exist, they are automatically created.

“missing” – returns a sorted list of all members within the group which are specified as being required or recommended, but are missing. There is no target specified.

**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).

1. The version number given here is for the specification language and is independent of the version number for the NWB format. The date after the version number is the last modification date of this document. [↑](#footnote-ref-1)