HDMF-common Specification

Release v1.5.1

Jan 10, 2022

Table of contents

Part I

Introduction

CHAPTER 1

Overview of hdmf-common

hdmf-common defines common data structures to be used across applications.

1.1 DynamicTable

The DynamicTable type is used to store tabular data. The tables are created in a columnar fashion with each column stored in its own VectorData object. Rows of the table are assigned unique ids with the required `id` column of type `ElementIdentifier`. The *colnames* attribute indicates the order of the columns.

1.2 VectorData

`VectorData` is the datatype used to store a column in a DynamicTable. If unpaired with a `VectorIndex` object the first dimension is the row dimension, which must be the same across all of the columns in that `DynamicTable`.

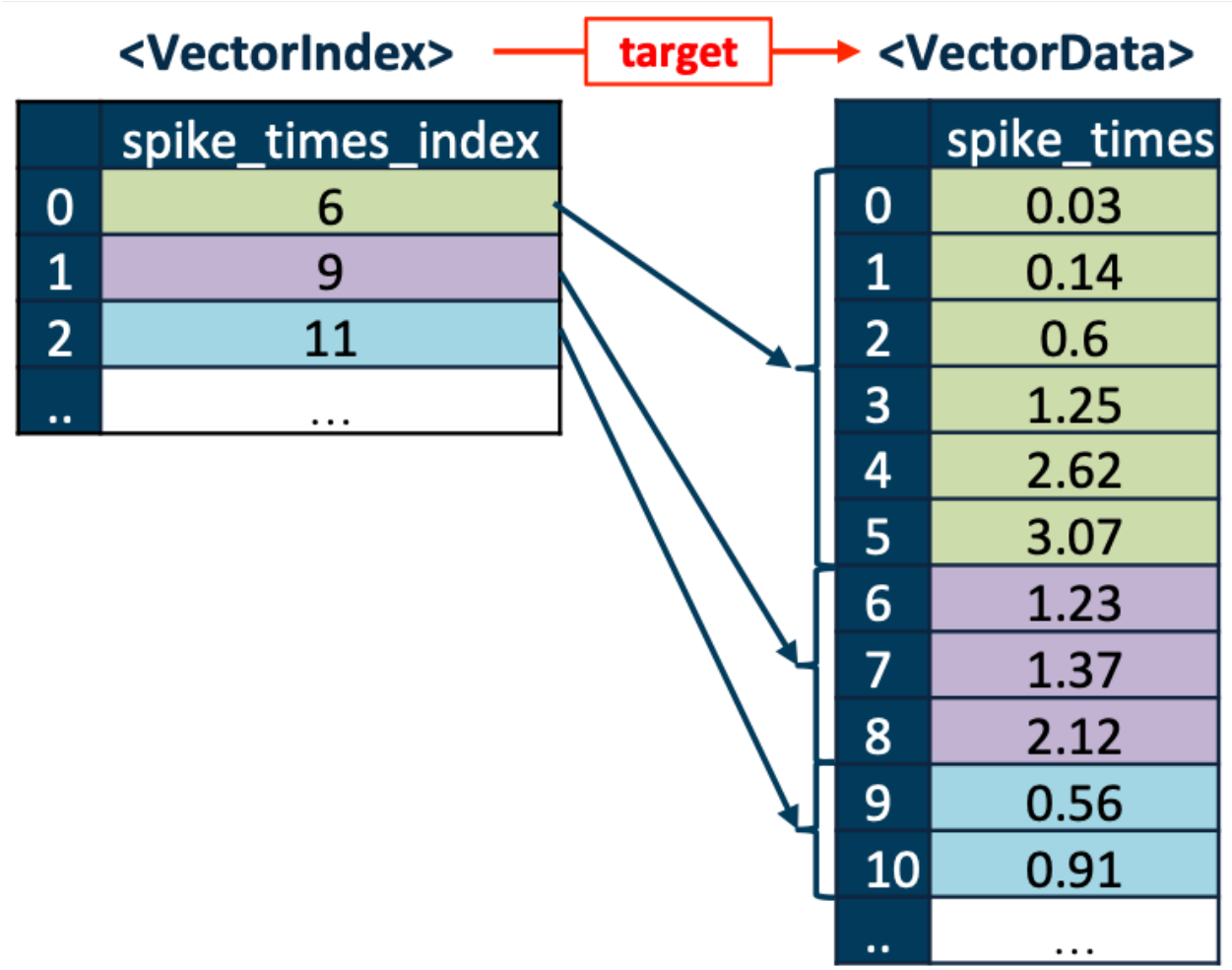
1.3 Ragged Arrays

(also known as Jagged Arrays)

Sometimes, you want to have a 2-d array where each row of the array has a different number of elements. For instance, in neuroscience, when storing the action potential times of sorted neurons, you might want to store them as a neuron x times matrix, but the problem is that each neuron will have a different number of spikes, so the second dimension will be inconsistent.

There are a number of possible solutions to this problem. Some solve it by NaN-padding the array. You might want to store the spike times of each neuron in a separate dataset, but that will not scale well if you have many neurons. In HDMF, you would store this using a pair of objects a VectorData and a `VectorIndex` object. The VectorData array holds all of the data concatenated as a 1-d array, and it is paired with a link to a `VectorIndex` object that indexes the data, forming a map between the rows of the ragged array and the indices of VectorData.

0	0.03	0.14	0.6	1.25	2.62	3.07
1	1.23	1.37	2.12			
2	0.56	0.91				
..	..	..	..	..		



These objects are generally stored inside a DynamicTable, and the elements of `VectorIndex` map onto the rows of the table. The VectorData object may be n-dimensional, but only the first dimension is ragged.

CHAPTER 2

Experimental data structures

The following data structures are currently available under the HDMF-experimental schema. These are subject to change! They are not guaranteed to exist in the future nor maintain backward compatibility.

2.1 ExternalResources

The ExternalResources type is used to store references to data stored in external, web-accessible databases. This information is maintained using four row-based tables.

Part II

Format Specification

Version v1.5.1 Jan 10, 2022

CHAPTER 3

Format Overview

3.1 Namespace – HDMF Common

- **Description:** Common data structures provided by HDMF
- **Name:** hdmf-common
- **Full Name:** HDMF Common
- **Version:** 1.5.1
- **Authors:**
  - Andrew Tritt
  - Oliver Ruebel
  - Ryan Ly
  - Ben Dichter
- **Contacts:**
  - ajtritt@lbl.gov
  - oruebel@lbl.gov
  - rly@lbl.gov
  - bdichter@lbl.gov
- **Schema:**
  - **doc:** base data types
  - **source:** base.yaml
  - **title:** Base data types
  - **doc:** data types for a column-based table
  - **source:** table.yaml

- **title:** Table data types
- **doc:** data types for different types of sparse matrices
- **source:** sparse.yaml
- **title:** Sparse data types

CHAPTER 4

Type Specifications

4.1 Base data types

base data types

4.1.1 Data

Overview: An abstract data type for a dataset.

- **Primitive Type:** Dataset
- **Subtypes:** *VectorIndex*, *ElementIdentifiers*, *VectorData*, *DynamicTableRegion*
- **Source filename:** base.yaml
- **Source Specification:** see Section 5.2.1

4.1.2 Container

Overview: An abstract data type for a group storing collections of data and metadata. Base type for all data and metadata containers.

- **Primitive Type:** Group
- **Subtypes:** *AlignedDynamicTable*, *DynamicTable*, *CSRMatrix*, *SimpleMultiContainer*
- **Source filename:** base.yaml
- **Source Specification:** see Section 5.2.2

4.1.3 SimpleMultiContainer

Overview: A simple Container for holding onto multiple containers.

`SimpleMultiContainer` extends `Container` and includes all elements of *Container* with the following additions or changes.

- **Extends:** *Container*
- **Primitive Type:** Group
- **Inherits from:** *Container*
- **Source filename:** base.yaml
- **Source Specification:** see Section 5.2.3

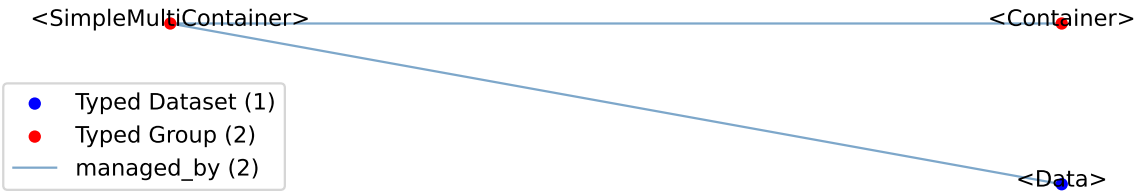


Table 4.1: Datasets, Links, and Attributes contained in <SimpleMultiContainer>

Id	Type	Description
<SimpleMultiContainer>	Group	Top level Group for <SimpleMultiContainer> • **Neurodata Type:** *SimpleMultiContainer* • **Extends:** *Container*
.<*Data*>	Dataset	Data objects held within this SimpleMultiContainer. • **Extends:** *Data* • **Quantity:** 0 or more

Table 4.2: Groups contained in <SimpleMultiContainer>

Id	Type	Description
<SimpleMultiContainer>	Group	Top level Group for <SimpleMultiContainer> • **Neurodata Type:** *SimpleMultiContainer* • **Extends:** *Container*
.<*Container*>	Group	Container objects held within this SimpleMultiContainer. • **Extends:** *Container* • **Quantity:** 0 or more

4.1.3.1 Groups: <Container>

Container objects held within this SimpleMultiContainer.

- **Extends:** *Container*
- **Quantity:** 0 or more

4.2 Table data types

data types for a column-based table

4.2.1 VectorData

Overview: An n-dimensional dataset representing a column of a DynamicTable. If used without an accompanying VectorIndex, first dimension is along the rows of the DynamicTable and each step along the first dimension is a cell of the larger table. VectorData can also be used to represent a ragged array if paired with a VectorIndex. This allows for storing arrays of varying length in a single cell of the DynamicTable by indexing into this VectorData. The first vector is at VectorData[0:VectorIndex[0]]. The second vector is at VectorData[VectorIndex[0]:VectorIndex[1]], and so on.

`VectorData` extends `Data` and includes all elements of *Data* with the following additions or changes.

- **Extends:** *Data*
- **Primitive Type:** Dataset
- **Dimensions:** [[‘dim0’], [‘dim0’, ‘dim1’], [‘dim0’, ‘dim1’, ‘dim2’], [‘dim0’, ‘dim1’, ‘dim2’, ‘dim3’]]
- **Shape:** [[None], [None, None], [None, None, None], [None, None, None, None]]
- **Inherits from:** *Data*
- **Subtypes:** *VectorIndex*, *DynamicTableRegion*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.1

Table 4.3: Datasets, Links, and Attributes contained in <VectorData>

Id	Type	Description
<VectorData>	Dataset	Top level Dataset for <VectorData> • **Neurodata Type:** VectorData • **Extends:** *Data* • **Dimensions:** [[‘dim0’], [‘dim0’, ‘dim1’], [‘dim0’, ‘dim1’, ‘dim2’], [‘dim0’, ‘dim1’, ‘dim2’, ‘dim3’]] • **Shape:** [[None], [None, None], [None, None, None], [None, None, None, None]]
.description	Attribute	Description of what these vectors represent. • **Data Type:** text • **Name:** description

4.2.2 VectorIndex

Overview: Used with VectorData to encode a ragged array. An array of indices into the first dimension of the target VectorData, and forming a map between the rows of a DynamicTable and the indices of the VectorData. The name of the VectorIndex is expected to be the name of the target VectorData object followed by "_index".

`VectorIndex` extends `VectorData` and includes all elements of *VectorData* with the following additions or changes.

- **Extends:** *VectorData*
- **Primitive Type:** Dataset
- **Data Type:** uint8
- **Dimensions:** ['num_rows']
- **Shape:** [None]
- **Inherits from:** *VectorData*, *Data*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.2

Table 4.4: Datasets, Links, and Attributes contained in <VectorIndex>

Id	Type	Description
<VectorIndex>	Dataset	Top level Dataset for <VectorIndex> • **Neurodata Type:** VectorIndex • **Extends:** *VectorData* • **Data Type:** uint8 • **Dimensions:** ['num_rows'] • **Shape:** [None]
.target	Attribute	Reference to the target dataset that this index applies to. • **Data Type:** object reference to *VectorData* • **Name:** target

4.2.3 ElementIdentifiers

Overview: A list of unique identifiers for values within a dataset, e.g. rows of a DynamicTable.

`ElementIdentifiers` extends `Data` and includes all elements of *Data* with the following additions or changes.

- **Extends:** *Data*
- **Primitive Type:** Dataset
- **Data Type:** int
- **Dimensions:** ['num_elements']
- **Shape:** [None]
- **Default Name:** element_id
- **Inherits from:** *Data*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.3

4.2.4 DynamicTableRegion

Overview: DynamicTableRegion provides a link from one table to an index or region of another. The *table* attribute is a link to another *DynamicTable*, indicating which table is referenced, and the data is int(s) indicating the row(s) (0-indexed) of the target array. *DynamicTableRegion's can be used to associate rows with repeated meta-data without data duplication. They can also be used to create hierarchical relationships between multiple 'DynamicTable's. 'DynamicTableRegion* objects may be paired with a *VectorIndex* object to create ragged references, so a single cell of a *DynamicTable* can reference many rows of another *DynamicTable*.

`DynamicTableRegion` extends `VectorData` and includes all elements of *VectorData* with the following additions or changes.

- **Extends:** *VectorData*
- **Primitive Type:** Dataset
- **Data Type:** int
- **Dimensions:** ['num_rows']
- **Shape:** [None]
- **Inherits from:** *VectorData*, *Data*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.4

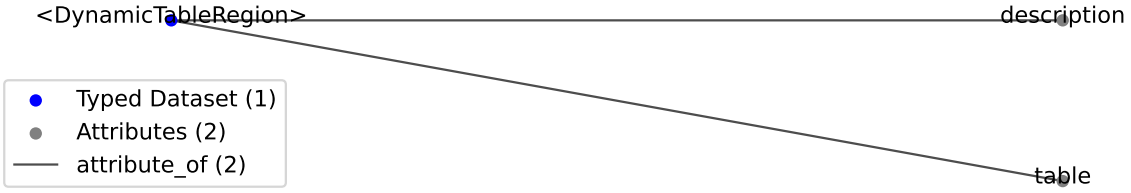


Table 4.5: Datasets, Links, and Attributes contained in <DynamicTableRegion>

Id	Type	Description
<DynamicTableRegion>	Dataset	Top level Dataset for <DynamicTableRegion> • **Neurodata Type:** DynamicTableRegion • **Extends:** *VectorData* • **Data Type:** int • **Dimensions:** ['num_rows'] • **Shape:** [None]
.table	Attribute	Reference to the DynamicTable object that this region applies to. • **Data Type:** object reference to *DynamicTable* • **Name:** table
.description	Attribute	Description of what this table region points to. • **Data Type:** text • **Name:** description

4.2.5 DynamicTable

Overview: A group containing multiple datasets that are aligned on the first dimension (Currently, this requirement if left up to APIs to check and enforce). These datasets represent different columns in the table. Apart from a column that contains unique identifiers for each row, there are no other required datasets. Users are free to add any number of custom VectorData objects (columns) here. DynamicTable also supports ragged array columns, where each element can be of a different size. To add a ragged array column, use a VectorIndex type to index the corresponding VectorData type. See documentation for VectorData and VectorIndex for more details. Unlike a compound data type, which is analogous to storing an array-of-structs, a DynamicTable can be thought of as a struct-of-arrays. This provides an alternative structure to choose from when optimizing storage for anticipated access patterns. Additionally, this type provides a way of creating a table without having to define a compound type up front. Although this convenience may be attractive, users should think carefully about how data will be accessed. DynamicTable is more appropriate for column-centric access, whereas a dataset with a compound type would be more appropriate for row-centric access. Finally, data size should also be taken into account. For small tables, performance loss may be an acceptable trade-off for the flexibility of a DynamicTable.

`DynamicTable` extends `Container` and includes all elements of *Container* with the following additions or changes.

- **Extends:** *Container*
- **Primitive Type:** Group
- **Inherits from:** *Container*
- **Subtypes:** *AlignedDynamicTable*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.5

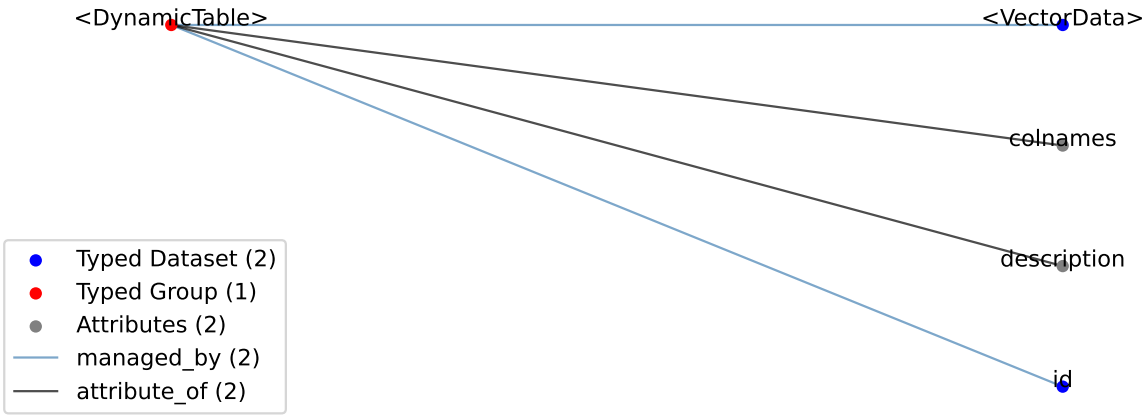


Table 4.6: Datasets, Links, and Attributes contained in <DynamicTable>

Id	Type	Description
<DynamicTable>	Group	Top level Group for <DynamicTable> • **Neurodata Type:** *DynamicTable* • **Extends:** *Container*

Continued on next page

Table 4.6 – continued from previous page

Id	Type	Description
.colnames	Attribute	The names of the columns in this table. This should be used to specify an order to the columns. • **Data Type:** text • **Dimensions:** [‘num_columns’] • **Shape:** [None] • **Name:** colnames
.description	Attribute	Description of what is in this dynamic table. • **Data Type:** text • **Name:** description
.id	Dataset	Array of unique identifiers for the rows of this dynamic table. • **Extends:** *ElementIdentifiers* • **Data Type:** int • **Dimensions:** [‘num_rows’] • **Shape:** [None] • **Name:** id
.<*VectorData*>	Dataset	Vector columns, including index columns, of this dynamic table. • **Extends:** *VectorData* • **Quantity:** 0 or more

4.2.6 AlignedDynamicTable

Overview: DynamicTable container that supports storing a collection of sub-tables. Each sub-table is a DynamicTable itself that is aligned with the main table by row index. I.e., all DynamicTables stored in this group MUST have the same number of rows. This type effectively defines a 2-level table in which the main data is stored in the main table implemented by this type and additional columns of the table are grouped into categories, with each category being represented by a separate DynamicTable stored within the group.

`AlignedDynamicTable` extends `DynamicTable` and includes all elements of *DynamicTable* with the following additions or changes.

- **Extends:** *DynamicTable*
- **Primitive Type:** Group
- **Inherits from:** *DynamicTable*, *Container*
- **Source filename:** table.yaml
- **Source Specification:** see Section 5.3.6

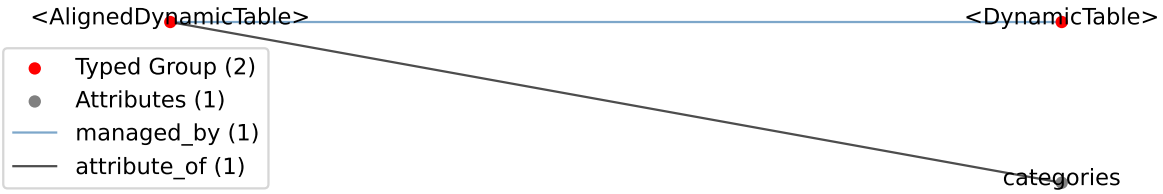


Table 4.7: Datasets, Links, and Attributes contained in <AlignedDynamicTable>

Id	Type	Description
<AlignedDynamicTable>	Group	Top level Group for <AlignedDynamicTable> • **Neurodata Type:** *AlignedDynamicTable* • **Extends:** *DynamicTable*
.categories	Attribute	The names of the categories in this AlignedDynamicTable. Each category is represented by one DynamicTable stored in the parent group. This attribute should be used to specify an order of categories and the category names must match the names of the corresponding DynamicTable in the group. • **Data Type:** text • **Dimensions:** [‘num_categories’] • **Shape:** [None] • **Name:** categories

Table 4.8: Groups contained in <AlignedDynamicTable>

Id	Type	Description
<AlignedDynamicTable>	Group	Top level Group for <AlignedDynamicTable> • **Neurodata Type:** *AlignedDynamicTable* • **Extends:** *DynamicTable*

Continued on next page

Table 4.8 – continued from previous page

Id	Type	Description
.<*DynamicTable*>	Group	A DynamicTable representing a particular category for columns in the AlignedDynamicTable parent container. The table MUST be aligned with (i.e., have the same number of rows) as all other DynamicTables stored in the AlignedDynamicTable parent container. The name of the category is given by the name of the DynamicTable and its description by the description attribute of the DynamicTable. • **Extends:** *DynamicTable* • **Quantity:** 0 or more

4.2.6.1 Groups: <DynamicTable>

A DynamicTable representing a particular category for columns in the AlignedDynamicTable parent container. The table MUST be aligned with (i.e., have the same number of rows) as all other DynamicTables stored in the AlignedDynamicTable parent container. The name of the category is given by the name of the DynamicTable and its description by the description attribute of the DynamicTable.

- **Extends:** *DynamicTable*
- **Quantity:** 0 or more

4.3 Sparse data types

data types for different types of sparse matrices

4.3.1 CSRMatrix

Overview: A compressed sparse row matrix. Data are stored in the standard CSR format, where column indices for row i are stored in indices[indptr[i]:indptr[i+1]] and their corresponding values are stored in data[indptr[i]:indptr[i+1]].

`CSRMatrix` extends `Container` and includes all elements of *Container* with the following additions or changes.

- **Extends:** *Container*
- **Primitive Type:** Group
- **Inherits from:** *Container*
- **Source filename:** sparse.yaml
- **Source Specification:** see Section 5.4.1

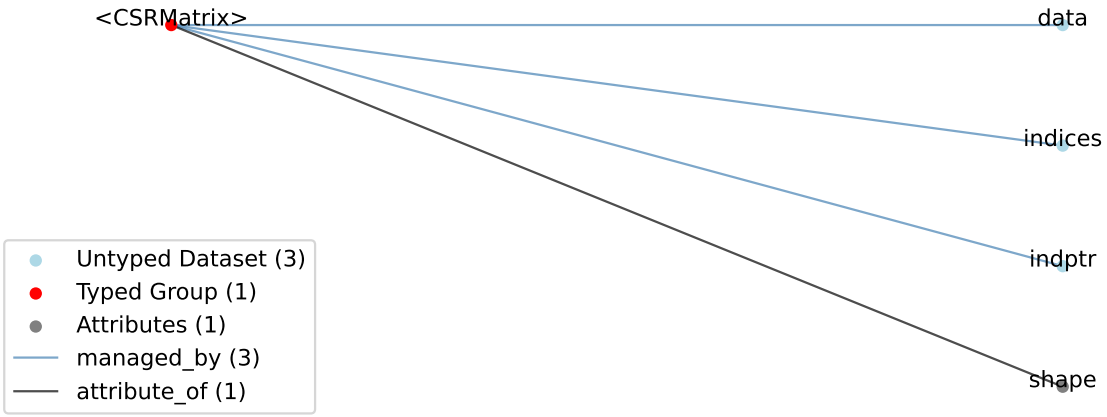


Table 4.9: Datasets, Links, and Attributes contained in <CSRMatrix>

Id	Type	Description
<CSRMatrix>	Group	Top level Group for <CSRMatrix> • **Neurodata Type:** *CSRMatrix* • **Extends:** *Container*
.shape	Attribute	The shape (number of rows, number of columns) of this sparse matrix. • **Data Type:** uint • **Dimensions:** [‘number of rows, number of columns’] • **Shape:** [2] • **Name:** shape
.indices	Dataset	The column indices. • **Data Type:** uint • **Dimensions:** [‘number of non-zero values’] • **Shape:** [None] • **Name:** indices

Continued on next page

Table 4.9 – continued from previous page

Id	Type	Description
.indptr	Dataset	The row index pointer. • **Data Type:** uint • **Dimensions:** [‘number of rows in the matrix + 1’] • **Shape:** [None] • **Name:** indptr
.data	Dataset	The non-zero values in the matrix. • **Dimensions:** [‘number of non-zero values’] • **Shape:** [None] • **Name:** data

CHAPTER 5

Schema Sources

Source Specification: see Section 5.1

5.1 Namespace – HDMF Common

Description: see Section 3.1

YAML Specification:

```yaml
author:
- Andrew Tritt
- Oliver Ruebel
- Ryan Ly
- Ben Dichter
contact:
- ajtritt@lbl.gov
- oruebel@lbl.gov
- rly@lbl.gov
- bdichter@lbl.gov
doc: Common data structures provided by HDMF
full_name: HDMF Common
name: hdmf-common
schema:
- doc: base data types
  source: base.yaml
  title: Base data types
- doc: data types for a column-based table
  source: table.yaml
  title: Table data types
- doc: data types for different types of sparse matrices
  source: sparse.yaml
  title: Sparse data types
version: 1.5.1
```

5.2 Base data types

base data types

5.2.1 Data

Description: see Section 4.1.1

YAML Specification:

```
data_type_def: Data
doc: An abstract data type for a dataset.
```

5.2.2 Container

Description: see Section 4.1.2

YAML Specification:

```
data_type_def: Container
doc: An abstract data type for a group storing collections of data and metadata. Base
  type for all data and metadata containers.
```

5.2.3 SimpleMultiContainer

Extends: *Container*

Description: see Section 4.1.3

YAML Specification:

```yaml
data_type_def: SimpleMultiContainer
data_type_inc: Container
datasets:
- data_type_inc: Data
  doc: Data objects held within this SimpleMultiContainer.
  quantity: '*'
doc: A simple Container for holding onto multiple containers.
groups:
- data_type_inc: Container
  doc: Container objects held within this SimpleMultiContainer.
  quantity: '*'
```

5.3 Table data types

data types for a column-based table

5.3.1 VectorData

Extends: *Data*

Description: see Section 4.2.1

YAML Specification:

```yaml
attributes:
- doc: Description of what these vectors represent.
  dtype: text
  name: description
data_type_def: VectorData
data_type_inc: Data
dims:
- - dim0
- - dim0
  - dim1
- - dim0
  - dim1
  - dim2
- - dim0
  - dim1
  - dim2
  - dim3
doc: An n-dimensional dataset representing a column of a DynamicTable. If used without
  an accompanying VectorIndex, first dimension is along the rows of the DynamicTable
  and each step along the first dimension is a cell of the larger table. VectorData
  can also be used to represent a ragged array if paired with a VectorIndex. This
  allows for storing arrays of varying length in a single cell of the DynamicTable
  by indexing into this VectorData. The first vector is at␣
↪VectorData[0:VectorIndex[0]].
  The second vector is at VectorData[VectorIndex[0]:VectorIndex[1]], and so on.
shape:
- - null
- - null
  - null
- - null
  - null
  - null
- - null
  - null
  - null
  - null
```

5.3.2 VectorIndex

Extends: *VectorData*

Description: see Section 4.2.2

YAML Specification:

```yaml
attributes:
- doc: Reference to the target dataset that this index applies to.
  dtype:
    reftype: object
    target_type: VectorData
  name: target
data_type_def: VectorIndex
data_type_inc: VectorData
dims:
- num_rows
doc: Used with VectorData to encode a ragged array. An array of indices into the first
  dimension of the target VectorData, and forming a map between the rows of a␣
↪DynamicTable
  and the indices of the VectorData. The name of the VectorIndex is expected to be
  the name of the target VectorData object followed by "_index".
dtype: uint8
shape:
- null
```

5.3.3 ElementIdentifiers

Extends: *Data*

Description: see Section 4.2.3

YAML Specification:

```yaml
data_type_def: ElementIdentifiers
data_type_inc: Data
default_name: element_id
dims:
- num_elements
doc: A list of unique identifiers for values within a dataset, e.g. rows of a
  ↪DynamicTable.
dtype: int
shape:
- null
```

5.3.4 DynamicTableRegion

Extends: *VectorData*

Description: see Section 4.2.4

YAML Specification:

```yaml
attributes:
- doc: Reference to the DynamicTable object that this region applies to.
  dtype:
    reftype: object
    target_type: DynamicTable
  name: table
- doc: Description of what this table region points to.
  dtype: text
  name: description
data_type_def: DynamicTableRegion
data_type_inc: VectorData
dims:
- num_rows
doc: DynamicTableRegion provides a link from one table to an index or region of another.
  The `table` attribute is a link to another `DynamicTable`, indicating which table
  is referenced, and the data is int(s) indicating the row(s) (0-indexed) of the target
  array. `DynamicTableRegion`s can be used to associate rows with repeated meta-data
  without data duplication. They can also be used to create hierarchical relationships
  between multiple `DynamicTable`s. `DynamicTableRegion` objects may be paired with
  a `VectorIndex` object to create ragged references, so a single cell of a `DynamicTable`
  can reference many rows of another `DynamicTable`.
dtype: int
shape:
- null
```

5.3.5 DynamicTable

Extends: *Container*

Description: see Section 4.2.5

YAML Specification:

```yaml
attributes:
- dims:
  - num_columns
  doc: The names of the columns in this table. This should be used to specify an order
    to the columns.
  dtype: text
  name: colnames
  shape:
  - null
- doc: Description of what is in this dynamic table.
  dtype: text
  name: description
data_type_def: DynamicTable
data_type_inc: Container
datasets:
- data_type_inc: ElementIdentifiers
  dims:
  - num_rows
  doc: Array of unique identifiers for the rows of this dynamic table.
  dtype: int
  name: id
  shape:
  - null
- data_type_inc: VectorData
  doc: Vector columns, including index columns, of this dynamic table.
  quantity: '*'
doc: A group containing multiple datasets that are aligned on the first dimension
  (Currently, this requirement if left up to APIs to check and enforce). These␣
↪datasets
  represent different columns in the table. Apart from a column that contains unique
  identifiers for each row, there are no other required datasets. Users are free to
  add any number of custom VectorData objects (columns) here. DynamicTable also␣
↪supports
  ragged array columns, where each element can be of a different size. To add a ragged
  array column, use a VectorIndex type to index the corresponding VectorData type.
  See documentation for VectorData and VectorIndex for more details. Unlike a compound
  data type, which is analogous to storing an array-of-structs, a DynamicTable can
  be thought of as a struct-of-arrays. This provides an alternative structure to␣
↪choose
  from when optimizing storage for anticipated access patterns. Additionally, this
  type provides a way of creating a table without having to define a compound type
  up front. Although this convenience may be attractive, users should think carefully
  about how data will be accessed. DynamicTable is more appropriate for column-centric
  access, whereas a dataset with a compound type would be more appropriate for row-
↪centric
  access. Finally, data size should also be taken into account. For small tables,
  performance loss may be an acceptable trade-off for the flexibility of a␣
↪DynamicTable.
```

5.3.6 AlignedDynamicTable

Extends: *DynamicTable*

Description: see Section 4.2.6

YAML Specification:

```yaml
attributes:
- dims:
  - num_categories
  doc: The names of the categories in this AlignedDynamicTable. Each category is represented
    by one DynamicTable stored in the parent group. This attribute should be used
    to specify an order of categories and the category names must match the names
    of the corresponding DynamicTable in the group.
  dtype: text
  name: categories
  shape:
  - null
data_type_def: AlignedDynamicTable
data_type_inc: DynamicTable
doc: DynamicTable container that supports storing a collection of sub-tables. Each
  sub-table is a DynamicTable itself that is aligned with the main table by row index.
  I.e., all DynamicTables stored in this group MUST have the same number of rows.
  This type effectively defines a 2-level table in which the main data is stored in
  the main table implemented by this type and additional columns of the table are
  grouped into categories, with each category being represented by a separate DynamicTable
  stored within the group.
groups:
- data_type_inc: DynamicTable
  doc: A DynamicTable representing a particular category for columns in the AlignedDynamicTable
    parent container. The table MUST be aligned with (i.e., have the same number of
    rows) as all other DynamicTables stored in the AlignedDynamicTable parent container.
    The name of the category is given by the name of the DynamicTable and its description
    by the description attribute of the DynamicTable.
  quantity: '*'
```

5.4 Sparse data types

data types for different types of sparse matrices

5.4.1 CSRMatrix

Extends: *Container*

Description: see Section 4.3.1

YAML Specification:

```yaml
attributes:
- dims:
  - number of rows, number of columns
  doc: The shape (number of rows, number of columns) of this sparse matrix.
  dtype: uint
  name: shape
  shape:
  - 2
data_type_def: CSRMatrix
data_type_inc: Container
datasets:
- dims:
  - number of non-zero values
  doc: The column indices.
  dtype: uint
  name: indices
  shape:
  - null
- dims:
  - number of rows in the matrix + 1
  doc: The row index pointer.
  dtype: uint
  name: indptr
  shape:
  - null
- dims:
  - number of non-zero values
  doc: The non-zero values in the matrix.
  name: data
  shape:
  - null
doc: A compressed sparse row matrix. Data are stored in the standard CSR format, where
  column indices for row i are stored in indices[indptr[i]:indptr[i+1]] and their
  corresponding values are stored in data[indptr[i]:indptr[i+1]].
```

Part III

Resources

CHAPTER 6

Making a Pull Request

Actions to take on each PR that modifies the schema and does not prepare the schema for a public release (this is also in the GitHub PR template):

If the current schema version on "main" is a public release, then:

1. Update the version string in `docs/source/conf.py` and `common/namespace.yaml` to the next version with the suffix "-alpha"
2. Add a new section in the release notes for the new version with the date "Upcoming"

Always:

1. Add release notes for the PR to `docs/source/hdmf_common_release_notes.rst` and/or `docs/source/hdmf_experimental_release_notes.rst`

Documentation or internal changes to the repo (i.e., changes that do not affect the schema files) do not need to be accompanied with a version bump or addition to the release notes.

CHAPTER 7

Merging PRs and Making Releases

Public release: a tagged release of the schema. The version string MUST NOT have a suffix indicating a pre-release, such as "-alpha". The current "dev" branch of HDMF and all HDMF releases MUST point to a public release of hdmf-common-schema. All schema that use hdmf-common-schema as a submodule MUST also point only to public releases.

Internal release: a state of the schema "main" branch where the version string ends with "-alpha".

The default branch of hdmf-common-schema is "main". **The "main" branch holds the bleeding edge version of the hdmf-common schema specification.**

PRs should be made to "main". Every PR should include an update to the namespace release notes (`docs/source/hdmf_common_release_notes.rst` and/or `docs/source/hdmf_experimental_release_notes.rst`). If the current version is a public release, then the PR should also update the version of the schema in two places: `docs/source/conf.py` and `common/namespace.yaml`. The new version should be the next bugfix/minor/major version of the schema with the suffix "-alpha". For example, if the current schema on "main" has version "2.2.0", then a PR implementing a bug fix should update the schema version from "2.2.0" to "2.2.1-alpha". Appending the "-alpha" suffix ensures that any person or API accessing the default "main" branch of the repo containing an internal release of the schema receives the schema with a version string that is distinct from public releases of the schema. If the current schema on "main" is already an internal release, then the version string does not need to be updated unless the PR requires an upgrade in the version (e.g., from bugfix to minor).

HDMF should contain a branch and PR that tracks the "main" branch of hdmf-common-schema. Before a public release of hdmf-common-schema is made, this HDMF branch should be checked to ensure that when the new release is made, the branch can be merged without issue.

Immediately prior to making a new public release, the version of the schema should be updated to remove the "-alpha" suffix and the documentation and release notes should be updated as needed (see next section).

The current "dev" branch of HDMF and all HDMF releases MUST always point to a public release of hdmf-common-schema. If a public release contains an internally released version of hdmf-common-schema, e.g., from an untagged commit on the "main" branch, then it will be difficult to find the version (commit) of hdmf-common-schema that was used to create an HDMF file when the schema is not cached.

CHAPTER 8

Making a Release Checklist

Before merging:

1. Update requirements versions as needed
2. Update legal file dates and information in `Legal.txt`, `license.txt`, `README.md`, `docs/source/conf.py`, and any other locations as needed
3. Update `README.md` as needed
4. Update the version string in `docs/source/conf.py` and `common/namespace.yaml` (remove "-alpha" suffix)
5. Update `docs/source/conf.py` as needed
6. Update release notes (set release date) in *docs/source/hdmf_common_release_notes.rst*, *docs/source/hdmf_experimental_release_notes.rst*, and any other docs as needed
7. Test docs locally (`cd docs; make fulldoc`) where the hdmf-common-schema submodule in the local version of HDMF is fully up-to-date with the head of the main branch.
8. Push changes to a new PR and make sure all PRs to be included in this release have been merged. Add `?template=release.md` to the PR URL to auto-populate the PR with this checklist.
9. Check that the readthedocs build for this PR succeeds (build latest to pull the new branch, then activate and build docs for new branch): https://readthedocs.org/projects/hdmf-common-schema/builds/

After merging:

1. Create a new git tag. Pull the latest main branch locally, run `git tag [version] --sign`, copy and paste the release notes into the tag message, and run `git push --tags`.
2. On the GitHub tags page, click "..." -> "Create release" for the new tag on the right side of the page. Copy and paste the release notes into the release message, update the formatting if needed (reST to Markdown), and set the title to the version string.
3. Check that the readthedocs "latest" and "stable" builds run and succeed. Delete the readthedocs build for the merged PR. https://readthedocs.org/projects/hdmf-common-schema/builds/

4. Update the HDMF submodule in the HDMF branch corresponding to this schema version to point to the tagged commit.

This checklist can also be found in the GitHub release PR template.

The time between merging this PR and creating a new public release should be minimized.

Part IV

History and Legal

CHAPTER 9

hdmf-common Release Notes

9.1 1.5.1 (January 10, 2022)

- No change in the hdmf-common namespace. See *here* for changes to the hdmf-experimental namespace.

9.2 1.5.0 (April 19, 2021)

- Added `AlignedDynamicTable`, which defines a `DynamicTable` that supports storing a collection of sub-tables. Each sub-table is itself a `DynamicTable` that is aligned with the main table by row index. Each sub-table defines a sub-category in the main table effectively creating a table with sub-headings to organize columns.

9.3 1.4.0 (March 29, 2021)

Summary: In 1.4.0, the HDMF-experimental namespace was added, which includes the `ExternalResources` and `EnumData` data types. Schema in the HDMF-experimental namespace are experimental and subject to breaking changes at any time. `ExternalResources` was changed to support storing both names and URIs for resources. The `VocabData` data type was replaced by `EnumData` to provide more flexible support for data from a set of fixed values.

- Added `EnumData` for storing data that comes from a set of fixed values. This replaces `VocabData` which could hold only string values. Also, `VocabData` could hold only a limited number of elements (~64k) when used with the HDF5 storage backend. `EnumData` gets around these restrictions by using an untyped dataset (VectorData) instead of a string attribute to hold the enumerated values.
- Removed `VocabData`.
- Renamed the "resources" table in `ExternalResources` to "entities".
- Created a new "resources" table to store the name and URI of the ontology / external resource used by the "entities" table in `ExternalResources`.

- Renamed fields in `ExternalResources`.
- Added "entities" dataset to `ExternalResources`. This is a row-based table dataset to replace the functionality of the "resources" dataset in `ExternalResources`.
- Changed the "resources" dataset in `ExternalResources` to store the name and URI of the ontology / external resource used by the "entities" dataset in `ExternalResources`.
- Added HDMF-experimental namespace.
- Moved `ExternalResources` and `EnumData` to HDMF-experimental.

9.4 1.3.0 (December 2, 2020)

- Add data type `ExternalResources` for storing ontology information / external resource references. NOTE: this data type is in beta testing and is subject to change in a later version.
- Changed dtype for datasets within `CSRMatrix` from 'int' to 'uint'. Negative values do not make sense for these datasets.

9.5 1.2.1 (November 4, 2020)

- Update software process documentation for maintainers.
- Fix missing data_type_inc for `CSRMatrix`. It now has `data_type_inc: Container`.
- Add `hdmf-schema-language` comment at the top of each yaml file.
- Add `SimpleMultiContainer`, a Container for storing other Container and Data objects together

9.6 1.2.0 (July 10, 2020)

- Add software process documentation.
- Fix missing dtype for `VectorIndex`.
- Add new `VocabData` data type.
- Move `Data`, `Index`, and `Container` to base.yaml. This change does not functionally change the schema.
- `VectorIndex` now extends `VectorData` instead of `Index`. This change allows `VectorIndex` to index other `VectorIndex` types.
- The `Index` data type is now unused and has been removed.
- Fix documentation for ragged arrays.

9.7 1.1.3 (January 21, 2020)

- Fix missing 'shape' and 'dims' key for types `VectorData`, `VectorIndex`, and `DynamicTableRegion`.

9.8 1.1.2 (January 9, 2020)

- Fix version number in namespace.yaml and docs

9.9 1.1.1 (January 9, 2020)

- Support for ReadTheDocs continuous documentation was added, and legal/license documents were also added. The schema is unchanged.

9.10 1.1.0 (January 3, 2020)

- The 'colnames' attribute of `DynamicTable` changed from data type 'ascii' to 'text'.
- Improved documentation and type docstrings.

9.11 1.0.0 (September 26, 2019)

Initial release.

CHAPTER 10

hdmf-experimental Release Notes

10.1 0.2.0 (January 10, 2022)

- In the experimental `ExternalResources`, added `relative_path` field to the "objects" table dtype. This is used in place of the previous `field` field representing the relative path to get to the dataset/attribute from the object. The previous `field` field will be used to represent a compound type field name if the dataset/attribute is a compound dtype.
- Updated contributors.

10.2 0.1.0 (March 29, 2021)

- See the release notes for *hdmf-common 1.4.0* for details.

CHAPTER 11

Credits

11.1 Authors

- Andrew Tritt
- Oliver Ruebel
- Ryan Ly
- Ben Dichter
- Matthew Avaylon

CHAPTER 12

Legal

12.1 Copyright

12.2 License