odmltables Documentation

Release 1.0.1.dev0

['odMLtables authors and contributors']

May 23, 2023

CONTENTS

CHAPTER

ONE

SYNOPSIS

odMLtables is a tool to support working with metadata collections for electrophysiological data and is described in detail in Sprenger et al. (2019) odMLtables: A User-Friendly Approach for Managing Metadata of Neurophysiological Experiments.

The odML file format and library API provides a means to store hierarchical metadata collections for electrophysiological data. Such collections typically consist of a large number of key-value pairs organized by a hierarchy of sections (see Grewe et al. (2011) Frontiers in Neuroinformatics 5:16). However, for editing and viewing metadata the use of standard spreadsheet software offering a flat tabular representation of a selected subset of metadata is desireable (cf., Zehl et al. (2016) Frontiers in Neuroinformatics 10:26). *odMLtables* provides a set of library functions as well as a graphical user interface that offers to swtich between hierarchical and flat representations of their metadata collection, and provides functions that assist in working with these files.

Currently, odMLtables supports:

- converting metadata collections between the hierarchical odML format and table-based representations (i.e., xls, csv)
- creating a new table for starting a metadata collection
- comparing sections within a metadata collections
- filtering metadata collections to extract a specific subcollection
- merging multiple metadata collections into one file

CHAPTER

TWO

CONTENTS:

2.1 Prerequisites / Installation

odMLtables is a pure Python package so that it should be easy to install on any system.

2.1.1 Dependencies

The following packages are required to use odMLtables:

xlrd >= 0.9.4 * xlwt >= 1.0.0 * numpy >= 1.8.2 * quantities >= 0.10.1 * odml >= 1.4.2 * future >= 0.16.0 * argparse >= 1.0.0

For installation of the graphical user interface the additional requirements are:

PyQt5>=5.0.0

and for building the documentation locally you need

numpydoc>=0.5.0 * sphinx>=1.2.2

and running the odMLtables tests requires

pytest>=3.0.0

2.1.2 Installation of released versions

Independent of the operating system odMLtables can be installed using pip:

```
pip install odMLtables
```

For installation of the graphical user interface run:

```
pip install odMLtables[gui]
```

Especially for the graphical user interface we recommend using Python 3. To be able to run the graphical user interface with Python 2.x please install PyQt version 5.0 or later manually (see below).

2.1.3 Installation from source

The latest version of odMLtables is available on GitHub. You can either use git and download odMLtables directly under Linux using:

```
cd /home/usr/toolbox/
git clone https://github.com/INM-6/python-odmltables.git
```

or alternatively download odMLtables as ZIP file and unzip it.

Linux

On Linux, to set up odMLtables navigate to your odMLtables folder and install odMLtables core via:

```
cd /home/usr/toolbox/python-odmltables/
python setup.py install
```

For installing also the grapical user interface run:

```
cd /home/usr/toolbox/python-odmltables/
pip install .[gui]
```

Please note that when using Python 2, the PyQt5 module needs to be manually installed beforehand, eg using conda:

```
conda install pyqt5
```

Now you can start the odMLtables graphical wizard by calling:

```
odmltables
```

Alternatively, you may navigate to the odMLtables folder and run:

```
./odmltables-gui
```

Windows/Mac OS X

On non-Linux operating systems we recommend using the Anaconda Python distribution, and installing all dependencies in a Conda environment, e.g.:

```
conda create -n metadataenv python numpy scipy pip six

activate metadataenv
```

Then navigate to the folder where you downloaded odMLtables and run:

```
python setup.py install
```

or:

```
pip install .
```

For installing also the odMLtables gui, please run:

```
pip install .[gui]
```

Please note, that for Python 2 you need to manually install PyQt when using the graphical user interface, e.g. using conda:

```
conda install PyQt>=5.0.0
```

On Windows, to run the graphical wizard, execute the *odmltables.exe* in the *Anaconda/Envs/metadataenv/Scripts* in your *User* directory.

Alternatively, on Windows or Mac OS X you may navigate to the odMLtables folder and run:

```
python odmltables-gui.py
```

2.1.4 Bugs

If you observe a bug in odMLtables please add a bug report at the GitHub issue tracker

2.2 Tutorial

At its core, odMLtables is a tool to convert hierarchical representations of metadata stored in the odML format to flat, tabluar formats. While the former is ideal to store, group, and structure large metadata collections, the latter is easier to visualize for the human eye and may be edited using powerful spreadsheet software. In either case, the data are structured as property-value pairs. Please refer to the documentation of odML for an in-depth tutorial of the format.

In general, there are two types of tables you can create yet: First, a table that represents a plain flattened overview of the entire odML, referred to as an *flattened odML table*. Second, a table that compares a specific set of properties (keys) across sections of the odML, referred to as a *comparative odML table*. Note that only the flattened odML table can be converted back to the hierarchical odML format, while the comparative odML table is intended for visualization of a specific part of the metadata collection, only.

In this tutorial we will guide you through the creation of both table types using the odMLtables libary API using both, the comma-separated value (csv) and Excel (xls) formats. Finally, we will present a concrete example of how to embed odMLtables into a workflow. For verification of your odml files you can view the content of an odml file using the metadataSylesheet for odML file version 1.0 and 1.1 provided by the G-Node.

In addition to the detailed step-by-step instructions presented here, there are also two **interactive tutorials** available as jupyter notebooks. Both tutorials can be directly executed using binder or run locally from the odmltables tutorial folder. The first notebook () is giving a quick overview on how odMLtables can be used in a metadata workflow by presenting a number of small application scenarios. The second notebook () shows the usage of odMLtables for handling large metadata collections and is based on two published experimental datasets.

2.2.1 Flattened odML table

This table is basically just a flat version of the hierarchical odML file. Every row of the table represents a property-value relationship of the odML (as you will see later, that does not mean you have to print every value). The columns represent information about each individual value. Possible columns are:

- **Path** The path to the section next to the value. Every value belongs to exactly one property, and every property to exactly one section. Thus, the path to the section and the property name uniquely identify the origin of the value (required).

- **SectionName** The name of the section (optional). This column is provided for better readability only, since the section name is also encoded in the Path.
- **SectionDefinition** The definition of the section (optional).
- **SectionType** The type of the section (optional).
- **PropertyName** The name of the property the value belongs to (required).
- **PropertyDefinition** The definition of the property (optional).
- **Value** The metadata value itself. Every row must have a value (required).
- **DataUnit** The unit of measurement of the value (optional).
- **DataUncertainty** The uncertainty of the value (optional).
- **odmlDatatype** The odML data type of the value (required). Note that this may differ from the datatypes used to represent the value in Python or Excel.

The required columns are the minimum number of columns required in order to convert the table back to a hierarchical odML representation. These also represent the default columns used by odMLtables: 'Path', 'Property Name', 'Value' and 'odML Data Type'.

csv

There are different formats you can save your tabular representation to, at the moment those are csv (comma-separated value) or xls (Excel). Since xls provides more possibilities concerning the appearance of the table we will start with the easier csv format.

Converting from odML to table

To create a csv table from an odML file you have to import the class *odml_csv_table.OdmlCsvTable* and create an instance of that class:

```
from odmltables import OdmlCsvTable

myFirstTable = OdmlCsvTable()
```

Then you can load your odML file:

```
myFirstTable.load_from_file('testfile.odml')
```

Now you can already write it to a csv-file by using the following command:

```
myFirstTable.write2file('testtable.csv')
```

You will get a table with the four columns; 'Path', 'Property Name', 'Value' and 'odML Data Type'.

Loading odML from other sources

You can not only load the odML from an odML-file, as shown in the example above. There are several other possibilities:

1. load from an `odml.Document` (class of the odML-Python-library):

```
import odml

doc = odml.Document()
# now append some sections, properties and values to the document

myTable = OdmlCsvTable()
myTable.load_from_odmldoc(doc)
```

2. load from another table – but this option will be explained later!

Changing the table header

The next step is to change the header to match your specific requirements for the table. In particular, you can choose which of the possible table columns (see above) will be in the table, their order, and also what the column headers are.

Warning: If you miss out one of the columns 'Path', 'Property Name', 'Value' and 'odML Data Type' in your table, it cannot be converted back to an odML-file. Also, if you change the names of the columns you will have to use the same settings to convert it back.

By using the function *odml_table.OdmlTable.change_header_titles()* you can choose a custom title for every column:

```
myFirstTable.change_header_titles(Path='my path',
                                  PropertyName='my property',
                                  Value='my value',
                                  odmlDatatype='my datatype')
```

The table should now look exactly as the old one, with the only difference that the names of the columns have changed. If you want to print additional columns, you can specify this by using the function *odml_table.OdmlTable.change_header()*:

```
myFirstTable.change_header(Path=1,
                           SectionName=2,
                           SectionDefinition=3,
                           PropertyName=4,
                           Value=5)
```

As you can see, in this function you can not only decide which columns to show, but also their order, by giving them numbers starting from 1. To include all possible headers, set the header to *full*:

```
myFirstTable.change_header('full')
```

Avoiding unnessaccery entries

You might already have noticed that not every cell of the tables is filled. To make a table better human-readable, redundant information about the Section (Path, SectionName and SectionDefinition) or the Property (PropertyName, PropertyDefinition) will not be printed if it is already contained in the previous row. To change this behaviour use the options `showall_sections` and `showall_properties`:

```
myFirstTable.showall_sections = True
myFirstTable.showall_properties = True
```

Now everything should be there.

xls

All the functions already shown for the csv table also work with xls tables. However, there are some additional features concerning the Style of cells. Again, first you need import the modul and create a new table:

```
from odml_xls_table import OdmlXlsTable
myXlsTable = OdmlXlsTable()
```

Choosing styles

There are some styles you can easily change in the table. First, there is the style of the header. You can choose the backcolor and fontcolor and the style of the font:

```
myXlsTable.header_style.backcolor = 'blue'
myXlsTable.header_style.fontcolor = ''
myXlsTable.header_style.fontstyle = 'bold 1'
```

The same way you can adapt the styles `first_style` and `second_style`. Those are the styles used for the orginary rows of the table. For a better visual representation, two style attributes exist that can be used in an alternating fashion (see section about *Changing grid patterns*).

You can find a table with all possible colors and their names `here`.

Highlighting columns

Sometimes there might be columns you want to lay a special focus on. So, to mark columns that they differ from the other, there is the option `mark_columns`:

```
myXlsTable.mark_columns('Path', 'Value')
```

Those marked columns will have a different style, which is determined by the attributes `first_marked_style` and `second_marked_style` (those can also be changed, as shown above).

Changing grid patterns

By default the two different styles for the rows will alternate when a new section starts. However, you can also change this behavior to change for each new property or even new value. If you dont want different colors at all, just turn it off. All this works by setting `changing_point` to either 'sections', 'properties', 'values' or None:

```
myXlsTable.changing_point = 'values'
```

Also, for a better distinctness between the columns , you can choose a 'chessfield'- pattern, so the styles will switch with every row.:

```
myXlsTable.pattern = 'chessfield'
```

2.2.2 Comparative odML table

It may happen that you have several sections with similar properties, for example one section per training day of an animal containing that days training parameters . To create a table in which you can easily compare values across different sections of an odML, you can use the comparative table representation.

csv

The easiest format here is, again, csv. So for the beginning, here is how you create a table to compare properties across sections.

Starting out

To create a csv-file with the table, import the class:

```
from compare_section_csv_table import CompareSectionCsvTable
myCompareTable = CompareSectionCsvTable()
```

Now you can load the table:

```
myCompareTable.load_from_file('somefile.odml')
```

Choosing sections

Next you have to decide which sections of the table you want to compare. You can either just choose all sections out of a list of sectionnames or you can select all sections with a specific beginning:

```
myCompareTable.choose_sections('s1', 's2', 's3')

# or

myCompareTable.choose_sections_startwith('s')
```

The latter would select all sections starting with an 's'. In the example above, this could be helpful if the sections were called 'Training_Day_01', 'Training_Day_02',... such that you could select alls sections starting with 'Training_Day'.

You can already write this table to a file:

```
myCompareTable.write2file('compare.csv')
```

The resulting file will have the properties in the header, and each following row represents one of the sections.

Switch the table

Now, assume we want to have the section names in the header and the property names in the first column. For example, if you have many sections to compare you might get a better overview by switching the table this way. This can be realized by setting `switch` to True:

```
myCompareTable.switch = True
```

Including all properties

If the sections you compare dont have exactly the same structure there might be properties appearing in one section but not in another. If you only want to compare those properties that are present in all of your chosen sections, set the option include_all to False:

```
myCompareTable.include_all = False
```

xls

In this part you will find the additional options for an xls-table.

Creating a table

To create a new table use the command:

```
from compare_section_xls_table import CompareSectionXlsTable()
xlsCompareTable = CompareSectionXlsTable()
```

Changing styles

There are again different styles you can adjust in this table:

1. **headerstyle** The style used for the captions of rows and columns.
2. **first_style** The style used for the values inside the table.
3. **second_style** The alternate style used for the values inside the table.
4. **missing_value_style** If `include_all` is True, this style will be used if a property doesnt exist in the section, so they distinguish from properties with empty values.

As already shown for the flattened table (*Choosing styles*), you may also adjust backcolor, fontcolor and fontstyle for each of the styles.

2.2.3 Practical examples

In these three short examples you will learn how to:

1. Generate a template odML starting from a table, which will then be used to
2. Manually enrich the odML via a tabular representation like it could be done in a daily workflow and finally how to
3. Reduce an odML, such that it can be used for a laboratory notebook or specific overviews

All source files can be found in the examples folder of the python-odmltables package .

Example 1: Generating a template odML

In this example you will learn how to generate an odML template file starting from an empty xls file. First you need to create an empty xls file 'example1.xls' using your preferred spreadsheet software and fill the first row with the header titles. In principle only four header titles are necessary to generate an odML from an xls table ('Path to Section', 'Property Name', 'Value' and 'odML Data Type'). Here we use two additional header titles ('Data Unit', 'Property Definition') as this information is important later in understanding of the metadata structure. The table should now look like this:

Path to Section	Property Name	Value	Data Unit	odML Data Type	Property Definition

Next, you need to decide on a structure of your odML. Here, we will implement only a small branch of an odML, which describes an animal, its attributes and the surgery. First of all, we choose properties we want to cover in the odML:

The animal

- **AnimalID** ID of the animal used for this experiment
- **Species** Species of the animal
- **Sex** Sex of the animal
- **Birthdate** Birthdate of the animal
- **Litter** ID of the litter
- **Seizures** Occurrence of seizures (observed / not observed)

The surgery

- **Surgeon** Name of the surgeon
- **Date** Date of surgery conduction (yyyy-mm-dd)
- **Weight** Weight of the animal (g)
- **Quality** Quality of the surgery (good / ok / bad)
- **Anesthetic** Type of anaesthetic

- **Painkiller** Name of painkiller, if used
- **Link** URL or folder containing surgery protocol

By describing the meaning of the properties, we also covered the property definition we need to provide. As the surgery is typically specific to the animal, we are going to use one main section for the animal ('/Animal') and a subsection for the description of the surgery ('/Animal/Surgery'). These are the 'Path to Section' values we need to provide in the xls table. In the next step we need to define the data types of the values we are going to put in the odml file. For most of the values a string is the best option (AnimalID, Species, Sex, Litter, Seizures, Surgeon, Quality, Anaesthestic, Painkiller), however some properties need different datatypes:

- **Birthdate / Date** date
- **Weight** float, this can be an arbitrary non-integer number
- **Link** url, this basically a string, but with special formatting.

Finally we are also able to define units for the values we are going to enter in this odML. In this example a unit is only necessary for the weight value, as the interpretation of this value highly depends on the unit. We define the unit of the weight as gram (g). If you now enter all the information discussed above in the xls table, this should look like below:

Path to Section	Property Name	Value	Data Unit	odML Data Type	Property Definition
/Animal	AnimalID			string	ID of the animal used for this experiment
	Species			string	Species of the animal
	Sex			string	Sex of the animal
	Birthdate			date	Birthdate of the animal
	Litter			string	ID of the litter
	Seizures			string	Occurrence of seizures (observed / not observed)
/Animal/Surgery	Surgeon			string	Name of the surgeon
	Date			date	Date of surgery conduction
	Weight		g	float	Weight of the animal
	Quality			string	Quality of the surgery (good / ok / bad)
	Anaesthetic			string	Type of anaesthetic
	Painkiller			string	Name of painkiller, if used
	Link			url	URL or folder containing surgery protocol

For the conversion of the xls file to an odML template file, you need to generate an OdmlXlsTable object and load the xls file:

```
import odmltables.odml_xls_table as odxlstable
# create OdmlXlsTable object
```

(continues on next page)

(continued from previous page)

```
xlstable = odxlstable.OdmlXlsTable()

# loading the data
xlstable.load_from_xls_table('example1.xls')
```

Now you can save it directly as odML file:

```
xlstable.write2odml('example1.odml')
```

This new odML file can now be used for multiple repetitions of the experiment and provides a standardized frame for recording metadata in this experiment.

Example 2: Manual enrichment of odML

In this example you are going to manually add data to an already existing odML document (see *Example 1: Generating a template odML*). In the best case, this odML document was already automatically enriched with digitally accessible values by custom, automatic enrichment routines. Then only few non-digitally available data need to be entered manually to complete the odML in terms of a complete description of the data and experiment. However, in principle the manual enrichment method presented here can also be used to start from a new odML table, and all metadata is manually entered.

We start from the odML generated in *Example 1: Generating a template odML*. If you don't have the resulting file, you can instead use `odml_tables/examples/example1/example1-2.odml` or generate an already pre-enriched odml (`odml_tables/examples/example2/example2-1.odml`) by running:

```
'python example2.py'
```

To generate an xls representation of the odML, load the odML and save it again using `odml.odml_xls_table.OdmlXlsTable`:

```
import odmltables.odml_xls_table as odml_xls_table

# create OdmlXlsTable object
xlstable = odml_xls_table.OdmlXlsTable()

# loading data from odml
xlstable.load_from_file(pre_enriched_file)

# save in xls format
xlstable.write2file('automatically_enriched.xls')
```

Now you need to manually enter the data you generated during the surgery into the xls file using your preferred spreadsheet software:

Path to Section	Property Name	Value	odML Data Type
/Animal	AnimalID	2A	string
	Species	Meriones unguiculatus	string
	Sex	female	string
	Birthdate	21-10-2015	date
	Litter	1A-01	string
	Seizures	not observed	string
/Animal/Surgery	Surgeon	Surgeon1	string
	Date	29-01-2016	date
	Weight	100	float
	Quality	good	string
	Anaesthetic	urethane	string
	Painkiller		string
	Link	../../surgery/protocols/protocol1.pdf	url

The completed xls file can then be saved as 'manually_enriched.xls' and converted back to the odML format via:

```
import odmltables.odml_xls_table as odml_xls_table

# create OdmlXlsTable object
xlstable = odml_xls_table.OdmlXlsTable()

# load data from manually enriched xls file
xlstable.load_from_xls_table('manually_enriched.xls')

# save data as odml document
xlstable.write2odml('example2-2.odml')
```

The 'example2-2.odml' file is now complete with manually entered metadata and can used for long term metadata storage and easy and fast metadata access for further analyses.

Example 3: Creating an overview sheet / Filtering sections and properties

In this example you are going to create an overview xls table of containing only a selection of properties of the original xls document. This feature can be used to create a summary table to be included in a laboratory notebook.

To apply the filter function we first need to generate a metadata collection. Here we are going to start from an xls representation of an odML, which you can generate by executing the example3.py script in the example folder of the odmltables package:

```
'python example3.py'
```

This generates the file 'example3.xls', which should look like this:

Document Information	author	Author1	date	2016-01-25	repository
Path to Section	**Property Name**	**Value**	**Data Unit**	**odML Data Type**	**Property Definition**
/Animal	AnimalID	2A		string	ID of the animal used for this experiment
	Species	Mouse		string	Species of the animal
	Strain	C57/Blj6		string	Strain of the animal
	Sex	female		string	Sex of the animal
	Birthdate	15-01-2016		date	Birthdate of the animal
	BirthAge	20	days	float	Time of birth after conception
	MotherID	1A		string	AnimalID of the mother
/Animal/Development	EyeOpening	20-01-2016		date	Date of first eye opening
	GraspingReflex	18-01-2016		date	Date of first observation of grasping reflex
/Animal/Development/dev_measures_template	Weight	-1	g	int	Weight of the animal
	BodyLength	-1	cm	int	Distance from nose to tail tip [cm]
	TailLength	-1	cm	int	Length of the tail
	Date	11-11-1900		date	Date of recording of this set of developmental measures
	CliffAvoidance	-1	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	-1		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_0	Weight	2	g	int	Weight of the animal
	BodyLength	3	cm	int	Distance from nose to tail tip [cm]
	TailLength	1	cm	int	Length of the tail
	Date	15-01-2016		date	Date of recording of this set of developmental measures
	CliffAvoidance	5	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	0		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_1	Weight	3	g	int	Weight of the animal
	BodyLength	4	cm	int	Distance from nose to tail tip [cm]
	TailLength	2	cm	int	Length of the tail
	Date	17-01-2016		date	Date of recording of this set of developmental measures
	CliffAvoidance	4	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	2		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_2	Weight	4	g	int	Weight of the animal
	BodyLength	5	cm	int	Distance from nose to tail tip [cm]
	TailLength	3	cm	int	Length of the tail
	Date	19-01-2016		date	Date of recording of this set of developmental measures
	CliffAvoidance	3	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	4		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_3	Weight	5	g	int	Weight of the animal
	BodyLength	6	cm	int	Distance from nose to tail tip [cm]
	TailLength	4	cm	int	Length of the tail
	Date	21-01-2016		date	Date of recording of this set of developmental measures
	CliffAvoidance	2	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	6		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_4	Weight	6	g	int	Weight of the animal
	BodyLength	7	cm	int	Distance from nose to tail tip [cm]
	TailLength	5	cm	int	Length of the tail
	Date	23-01-2016		date	Date of recording of this set of developmental measures
	CliffAvoidance	1	s	float	Time after which cliff aversion reflex is observed
	DevelopmentalAge	8		int	Developmental age of the animal in days after birth (P_)

Fig. 1: Example 3: xls representation of the complete odML structure.

This example structure contains only the branch of an odML describing the animal and its development. The previously acquired information about the animal are saved in properties directly attached to the '/Animal' section. To capture the developmental data a subsection '/Animal/Development' exists, which contains those developmental properties that only consist of a single measurement value. In addition, several 'dev_measures_x' subsections are attached to the 'Animal/Development' section, which each contain a set of values measured on one day. These sections are copies of the '/Animal/Development/dev_measures_template' section. Typically the template section is copied for each day of measurement and values are entered manually (eg. in this xls sheet).

For practical purposes it can be necessary to create an overview sheet containing only a subset of these developmental measures, eg. for printing them and adding them to the laboratory notebook. Here we focus on the 'DevelopmentalAge' and 'Weight' properties. To get an odMLtables representation of the xls file we generate an OdmlXlsTable object and load the data from the xls file:

```
import odmltables.odml_xls_table as odxlstable
# create OdmlXlsTable object
xlstable = odxlstable.OdmlXlsTable()

# loading the data
xlstable.load_from_xls_table('example3.xls')
```

Now we are going to apply a filter, which only leaves the properties with name 'DevelopmentalAge' or 'Weight' in the table:

```
xlstable.filter(PropertyName=['DevelopmentalAge','Weight'], comparison_func= lambda x,y:␣
↪(x in y))
```

If we save it as 'example3_Output.xls':

```
xlstable.write2file('example3_Output.xls')
```

this looks as follows:

Document Information	author		Author date	2016-01-25	repository
Path to Section	**Property Name**	**Value**	**Data Unit**	**odML Data Type**	**Property Definition**
/Animal/Development/dev_measures_template	Weight	-1	g	int	Weight of the animal
	DevelopmentalAge	-1		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_0	Weight	2	g	int	Weight of the animal
	DevelopmentalAge	0		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_1	Weight	3	g	int	Weight of the animal
	DevelopmentalAge	2		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_2	Weight	4	g	int	Weight of the animal
	DevelopmentalAge	4		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_3	Weight	5	g	int	Weight of the animal
	DevelopmentalAge	6		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_4	Weight	6	g	int	Weight of the animal
	DevelopmentalAge	8		int	Developmental age of the animal in days after birth (P_)

Fig. 2: Example 3: xls representation of the odML structure after first filtering.

However, the resulting table still contains the 'dev_measures_template' section and all its properties, which is not usefull in a printout for a laboratory notebook. To remove this, we apply a second filter:

```
xlstable.filter(invert=True, Path='template', comparison_func=lambda x,y: x.endswith(y))
```

This operation only leaves properties in the table, whose parent section name does not end with 'template' and therefore removes the 'dev_measures_template' section and all its properties.

Document Information	author		Author date	2016-01-27	repository
Path to Section	**Property Name**	**Value**	**Data Unit**	**odML Data Type**	**Property Definition**
/Animal/Development/dev_measures_0	Weight	2	g	int	Weight of the animal
	DevelopmentalAge	0		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_1	Weight	3	g	int	Weight of the animal
	DevelopmentalAge	2		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_2	Weight	4	g	int	Weight of the animal
	DevelopmentalAge	4		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_3	Weight	5	g	int	Weight of the animal
	DevelopmentalAge	6		int	Developmental age of the animal in days after birth (P_)
/Animal/Development/dev_measures_4	Weight	6	g	int	Weight of the animal
	DevelopmentalAge	8		int	Developmental age of the animal in days after birth (P_)

Fig. 3: Example 3: xls representation of the odML structure after second filtering.

This filtered representation of the original xls file can also be further adapted in terms of the layout of the table and finally printed or converted to pdf using a spreadsheet software.

2.2.4 Graphical Frontend

The use of the Python API as described above gives you full flexibility over the conversion processes that may be required for your project. Also, it allows you to implement workflows to initiate automated conversion steps to compile metadata from multiple sources, and merge it with manually entered metadata, as described in Zehl et al, 2016, Frontiers in Neuroinformatics 10, 26.

However, many of the functions outlined above are also accessible via a graphical front-end that allows to comfortably perform some of the most frequent steps in viewing and manipulating odML-based metadata collections, including conversion to flattened table structures or filtering. Please see the installation instructions to learn how to run the graphical front-end.

2.3 Function Reference by Module

Here you find a detailed documentation of the code.

2.3.1 Flattened odML table

class odml_table.**OdmlDtypes**(*basedtypes_dict=None*, *synonyms_dict=None*)

Class to handle odml data types, synonyms and default values.

Parameters

- **basedtypes_dict** – Dictionary containing additional basedtypes to use as keys and default values as values. Default: None
- **synonyms_dict** – Dictionary containing additional synonyms to use as keys and basedtypes to associate as values. Default: None

Returns
None

add_synonym(*basedtype*, *synonym*)

Setting user specific default synonyms :param basedtype: Accepted basedtype of OdmlDtypes or None. None delete already existing synonym :param synonym: Synonym to be connected to basedtype :return: None

to_odml_value(*value*, *dtype*)

Convert single value entry or list of value entries to odml compatible format

class odml_table.**OdmlTable**(*load_from=None*)

Class to create tables in different formats from odml-files

Parameters

- **show_all_sections** (*bool*) – if set to False, information about the section like the path or name of the section wont be in the table again, if they are same as in the line before
- **show_all_properties** (*bool*) – if set to False, information about the property like the name or definition of the property wont be in the table again, if they are same as in the line before tables with an emptycolumn

change_header(**args*, ***kwargs*)

Function to change the header of the table.

The keywordarguments of the function are the possible columns you can include into your table; they are listed below, you can also check the possible options bei looking at the keys of the header_titles dictionary.

They take the number of their position in the table, starting from left with 1. The default-header is [‘Path’, ‘Property Name’, ‘Value’, ‘odML Data Type’]. These are the columns you need to be able to convert your table back to an odml-file. Important: You can create tables wich dont contain any of those four, but they cant be converted back to odml.

Parameters

- **Path** (*int, optional*) – Position of the ‘Path’-Column in the table.
- **SectionName** (*int, optional*) – Position of the ‘Section Name’-Column in the table
- **SectionType** (*int, optional*) – Position of the ‘Section Type’-Column in the table
- **SectionDefinition** (*int, optional*) – Position of the ‘Section Definition’-Column in the table
- **PropertyName** (*int, optional*) – Position of the ‘Property Name’-Column in the table
- **PropertyDefinition** (*int, optional*) – Position of the ‘Property Definition’-Column in the table
- **Value** (*int, optional*) – Position of the ‘Value’-Column in the table
- **DataUnit** (*int, optional*) – Position of the ‘Data Unit’-Column in the table
- **DataUncertainty** (*int, optional*) – Position of the ‘Data Uncertainty’-Column in the table
- **odmlDatatype** (*int, optional*) – Position of the ‘odML Data Type’-Column in the table

Example

mytable.change_header(Path=1, Value=3, odmlDataType=2)
=> outcoming header: [‘Path’, ‘odML Data Type’, ‘Value’]

change_header_titles(***kwargs*)

Function to change the Name of a column in your table. Be careful with this function if you want to convert the table back to an odml.

Parameters

- **Path** (*string, optional*) – Name of the ‘Path’-Column in the table
- **SectionName** (*string, optional*) – Name of the ‘Section Name’-Column in the table
- **SectionType** (*string, optional*) – Name of the ‘Section Type’-Column in the table
- **SectionDefinition** (*string, optional*) – Name of the ‘Section Definition’-Column in the table
- **ProgertyName** (*string, optional*) – Name of the ‘Property Name’-Column in the table
- **PropertyDefinition** (*string, optional*) – Name of the ‘Property Definition’-Column in the table
- **Value** (*string, optional*) – Name of the ‘Value’-Column in the table
- **DataUnit** (*string, optional*) – Name of the ‘Data Unit’-Column in the table
- **DataUncertainty** (*string, optional*) – Name of the ‘Data Uncertainty’-Column in the table
- **odmlDatatype** (*string, optional*) – Name of the ‘odML Data Type’-Column in the table

consistency_check()

check odmldict for consistency regarding dtypes to ensure that data can be loaded again.

convert2odml()

Generates odml representation of odmldict and returns it as odml document. :return:

filter(*mode='and'*, *invert=False*, *recursive=False*, *comparison_func=<function OdmlTable.<lambda>>*, ***kwargs*)

filters odml properties according to provided kwargs.

Parameters

- **mode** – Possible values: 'and', 'or'. For 'and' all keyword arguments must be satisfied for a property to be selected. For 'or' only one of the keyword arguments must be satisfied for the property to be selected. Default: 'and'
- **invert** – Inverts filter function. Previously accepted properties are rejected and the other way round. Default: False
- **recursive** – Delete also properties attached to subsections of the mother section and therefore complete branch
- **comparison_func** – Function used to compare dictionary entry to keyword. Eg. 'lambda x,y: x.startswith(y)' in case of strings or 'lambda x,y: x in y' in case of multiple permitted values. Default: lambda x,y: x==y
- **kwargs** – keywords and values used for filtering

Returns
None

static get_csv_header(*load_from*)

Providing non-empty csv header entries of first sheet for odml tables gui only :return:

static get_xls_header(*load_from*)

Providing non-empty xls header entries of first sheet for odml tables gui only :return:

load_from_csv_table(*load_from*)

loads the odmldict from a csv-file containing an odml-table. To load the odml, at least Value, Path, PropertyName and odmlDatatype must be given in the table. Also, the header_titles must be correct

Parameters
load_from (*string*) – name(path) of the csv-file

load_from_file(*load_from*)

loads the odml-data from an odml-file

Parameters
load_from (*string*) – the path to the odml-file

load_from_function(*odmlfct*)

loads the odml-data by using a function that creates an odml-document

Parameters
load_from (*function*) – function that returns an odml-document

load_from_odmldoc(*doc*)

loads the odml-data from an odml-document

Parameters
load_from (*odml-document*) – the odml-document

load_from_xls_table(*load_from*)

loads the odml-data from a xls-file. To load the odml, at least Value, Path, PropertyName and odmlDatatype must be given in the table. Also, the header_titles must be correct

Parameters
load_from (*string*) – name(path) of the xls-file

merge(*odmltable*, *overwrite_values=False*, ***kwargs*)

Merge odmltable into current odmltable.

Parameters

- **odmltable** – OdmlTable object or odML document object
- **overwrite_values** – Bool value to indicate whether values of odML Properties should be merged (appended) or overwritten by the entries of the other odmltable object. Default is False.

Returns
None

write2file(*save_to*)

write the table to the specific file

write2odml(*save_to*)

writes the loaded odmldict (e.g. from an csv-file) to an odml-file

class odml_csv_table.OdmlCsvTable(*load_from=None*)

Class to create a csv-file from an odml-file

write2file(*save_to*)

writes the data from the odml-file to a csv-file. Each line of the table represents one Value of the odml-file. By changing the header of the table you can choose, which informations about those values will be shown in the table. You can also decide, not to include information about every specific value in your header, for example if you just want to get an overview of your odml-structur. Then rows, that would be empty will be skipped and not printed in the table.

Parameters
save_to (*string*) – name of the csv-file

class odml_xls_table.OdmlXlsTable(*load_from=None*)

Class to create a csv-file from an odml-file

Parameters

- **sheetname** (*string*) – name of the excel sheet; default is 'sheet1'
- **header_style** (*XlsStyle*) – style used for the header of the table
- **first_style** (*XlsStyle*) – default style used for the rows
- **second_style** (*XlsStyle*) – used to switch styles of the rows if changing_point is not None
- **first_marked_style** (*XlsStyle*) – default style used in marked columns
- **second_marked_style** (*XlsStyle*) – used to switch styles of the rows in marked columns if changing_point is not None
- **pattern** (*string*) – can be 'alternating' or 'checkerboard'
- **changing_point** (*string*) – select the point for changing styles. this can be when a new section, property or value starts ('sections', 'properties', 'values' or None)

mark_columns(**args*)

choose the columns of the table you want to highlight by giving them another style (for example a different color). Possible Arguments are:

- 'Path'
- 'SectionName'
- 'SectionType'
- 'SectionDefinition'
- 'PropertyName'
- 'PropertyDefinition'
- 'Value'
- 'DataUnit'
- 'DataUncertainty'
- 'odmlDatatype'.

write2file(*save_to*)

writes the data from the odml-file to a xls-file

Parameters
save_to (*string*) – name of the xls-file

2.3.2 Comparative Tables

class compare_section_table.**CompareSectionTable**

class to create a table in which you compare different sections of a odml- file wich have the same properties

Parameters

- **include_all** (*bool*) – if set to false, only those properties which exist in every chosen section will be shown
- **switch** (*bool*) – when set to True, the table will be switched so the sections are in the rows and the properties in the columns

choose_sections(**args*)

choose all sections out of the list of sectionnames you give this function

Parameters
args (*strings*) – names of the sections

Example
a.choose_sections('section1', 'section2', 'section4')

choose_sections_startwith(*startwith*)

choose all sections with the same beginning

Parameters
startwith (*string*) – beginning of the sectionname of the sections that will be compared

load_from_file(*load_from*)

load the data for the table from an odml-file

Parameters
load_from (*string*) – Name of the odml-file to load from

write2file(*save_to*)

write the table to the specific file

Parameters
save_to (*string*) – path and name where the file will be saved

Raises
NotImplementedError – Implemented in the subclass

class compare_section_xls_table.**CompareSectionXlsTable**

class to write a CompareSectionTable to a xls-file

Parameters

- **sheet_name** (*string*) – name of the excel-sheet, default is 'sheet1'
- **header_style** (*XlsStyle*) – style used for the header
- **first_style** (*XlsStyle*) – style used for the values inside the table
- **second_style** (*XlsStyle*) – second style used for the values inside the table
- **missing_value_style** (*XlsStyle*) – if include_all is True, this style will be used if a property doesnt exist in the section, so they distinguish from properties with empty values

write2file(*save_to*)

writes the table to an xls-file

class compare_section_csv_table.**CompareSectionCsvTable**

class to write a CompareSectionTable to a csv-file

write2file(*save_to*)

saves the table as a csv-file

2.4 Release notes

2.4.1 odMLtables 1.0.1 release notes

October 2020

Main features:

- Improved compatibility with odML 1.5.1 and numpy 1.19.2
- style sheets for odML visualization are by default integrated in exported files

2.4.2 odMLtables 1.0.0 release notes

January 2019

Main features:

- odMLtables gui is installable via pip
- command line arguments for specifying files for odMLtables wizards
- Python 3 compatibility
- Jupyter tutorial notebooks

- Extended documentation

Other features:

- *xml* as file extension for odML files
- many bug fixes in all wizards

Updates:

- Minimal odML version 1.4.2
- PyQt5

2.5 Authors and contributors

The following people have contributed code and/or ideas to the current version of odMLtables. The institutional affiliations are those at the time of the contribution, and may not be the current affiliation of a contributor.

- Carlos Canova [1]
- Jana Pick [1]
- Julia Sprenger [1,3]
- Michael Denker [1]
- Lyuba Zehl [1]
- Michael Sonntag [2]

1. Institute of Neuroscience and Medicine (INM-6), Computational and Systems Neuroscience & Institute for Advanced Simulation (IAS-6), Theoretical Neuroscience, Jülich Research Centre and JARA, Jülich, Germany
2. Department of Biology II, Ludwig-Maximilians-Universität München, Martinsried, Germany
3. Institut de Neurosciences de la Timone (INT), Marseille, France

If we've somehow missed you off the list we're very sorry - please let us know.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FOUR

CITATION

If you are using *odMLtables* for your project please consider citing

Sprenger J, Zehl L, Pick J, Sonntag M, Grewe J, Wachtler T, Grün S and Denker M (2019) odMLtables: A User-Friendly Approach for Managing Metadata of Neurophysiological Experiments. Front. Neuroinform. 13:62. doi: 10.3389/fninf.2019.00062

PYTHON MODULE INDEX

C

O

INDEX