python-odml Documentation

Release 1.5.3

Hagen Fritsch

May 11, 2023

Contents

Contents:

CHAPTER 1

odML Tutorial

Author Lyuba Zehl, Michael Sonntag; based on work by Hagen Fritsch

Release 1.5

1.1 odML (open metadata Markup Language)

odML (open metadata Markup Language) is a framework, proposed by Grewe et al. (2011), to organize and store experimental metadata in a human- and machine-readable, XML based format (odml). In this tutorial we will illustrate the conceptual design of the odML framework and show hands-on how you can generate your own odML metadata file collection. A well organized metadata management of your experiment is a key component to guarantee the reproducibility of your research and facilitate the provenance tracking of your analysis projects.

What are metadata and why are they needed? Metadata are data about data. They describe the conditions under which the actual raw-data of an experimental study were acquired. The organization of such metadata and their accessibility may sound like a trivial task, and most laboratories developed their home-made solutions to keep track of their metadata. Most of these solutions, however, break down if data and metadata need to be shared within a collaboration, because implicit knowledge of what is important and how it is organized is often underestimated.

While maintaining the relation to the actual raw-data, odML can help to collect all metadata which are usually distributed over several files and formats, and to store them unitedly which facilitates sharing data and metadata.

Key features of odML

- open, XML based language, to collect, store and share metadata
- Machine- and human-readable
- Python-odML library
- Interactive odML-Editor

1.2 Structure of this tutorial

The scientific background of the possible user community of odML varies enormously (e.g. physics, informatics, mathematics, biology, medicine, psychology). Some users will be trained programmers, others probably have never learned a programming language.

To cover the different demands of all users, we provide a slow introduction to the odML framework that even allows programming beginners to learn the basic concepts. We will demonstrate how to generate an odML file and present more advanced possibilities of the Python-odML library (e.g., how to search for certain metadata or how to integrate existing terminologies).

At the end of this tutorial we will provide a few guidelines that will help you to create an odML file structure that is optimised for your individual experimental project and complements the special needs of your laboratory.

The code for the example odML files, which we use within this tutorial is part of the documentation package (see doc/example_odMLs/).

A summary of available odML terminologies and templates can be found at the G-Node odML terminology and odML template pages.

1.3 Download and Installation

The odML framework is an open source project of the German Neuroinformatics Node (G-Node, odML project website) of the International Neuroinformatics Coordination Facility (INCF). The source code for the Python-odML library is available on GitHub under the project name python-odml.

1.3.1 Dependencies

The Python-odML library (version 1.5+) is tested and fully supported using Python 3.7+.

Additionally, the Python-odML library depends on the lxml, pyyaml and rdflib python packages.

When the odML-Python library is installed via pip or the setup.py, these packages will be automatically downloaded and installed. Alternatively, they can be installed from the OS package manager.

On Ubuntu, the dependency packages are available as `python-lxml`, `python-yaml` and `python-rdflib`.

Note that on Ubuntu 14.04, the latter package additionally requires the installation of `libxml2-dev`, `libxslt1-dev`, and `lib32z1-dev`.

Python 2 has reached end of life. Current and future versions of odml are not Python 2 compatible. We removed support for Python 2 in August 2020 with version 1.5.2. We also recommend using a Python version >= 3.7. If a Python version < 3.7 is a requirement, the following dependency needs to be installed as well:

The `enum34` package with a `pip` installation or `python-enum` using the OS package manager.

1.3.2 Installation...

... via pip:

The simplest way to install the Python-odML library is from PyPI using pip:

```
$ pip install odml
```

The appropriate Python dependencies will be automatically downloaded and installed.

If you are not familiar with PyPI and pip, please have a look at the available online documentation.

... from source:

To download the Python-odML library please either use git and clone the repository from GitHub:

```
$ cd /home/usr/toolbox/
$ git clone https://github.com/G-Node/python-odml.git
```

... or if you don't want to use git, download the ZIP file also provided on GitHub to your computer (e.g. as above on your home directory under a "toolbox" folder).

To install the Python-odML library, enter the corresponding directory and run:

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

1.3.3 Bugs & Questions

Should you find a behaviour that is likely a bug, please file a bug report at the github bug tracker.

1.4 Basic knowledge on odML

Before we start, it is important to know the basic structure of an odML file. Within an odML file metadata are grouped and stored in a hierarchical tree structure which consists of three basic odML objects.

Document:

- description: *root of the tree*
- parent: *no parent*
- children: *Section*

Section:

- description: *branches of the tree*
- parent: *Document or Section*
- children: *Section and/or Property*

Property:

- description: *leafs of the tree (contains metadata values)*
- parent: *Section*
- children: *none*

Each of these odML objects has a certain set of attributes where the user can describe the object and its contents. Which attribute belongs to which object and what the attributes are used for is better explained in an example odML file (cf., "THGTTG.odml").

1.5 A first look

If you want to get familiar with the concept behind the odML framework and how to handle odML files in Python, you can have a first look at the example odML file provided in the Python-odML library. For this you first need to run the python code ("thgttg.py") to generate the example odML file ("THGTTG.odml"). When using the following commands, make sure you adapt the paths to the python-odml module to your own!:

```
$ cd /home/usr/.../python-odml
$ ls doc/example_odMLs
thgttg.py
$ python doc/example_odMLs/example_odMLs.py "/home/usr/.../python-odml"
$ ls doc/example_odMLs
THGTTG.odml  thgttg.py
```

Now open a Python shell within the Python-odML library directory, e.g. with IPython:

```
$ ipython
```

In the IPython shell, first import the odml package:

```
>>> import odml
```

Second, load the example odML file with the following command lines:

```
>>> to_load = './doc/example_odMLs/THGTTG.odml'
>>> odmlEX = odml.load(to_load)
```

If you open a Python shell outside of the Python-odML library directory, please adapt your Python-Path and the path to the "THGTTG.odml" file accordingly.

How you can access the different odML objects and their attributes once you loaded an odML file and how you can make use of the attributes is described in more detail in the following chapters for each odML object type (Document, Section, Property).

How you can create the different odML objects on your own and how to connect them to build your own metadata odML file will be described in later chapters. Further advanced functions you can use to navigate through your odML files, or to create an odML template file, or to make use of common odML terminologies provided via the G-Node repository can also be found later on in this tutorial.

But now, let us first have a look at the example odML file (THGTTG.odml)!

1.5.1 The Document

If you loaded the example odML file, let's have a first look at the Document:

```
>>> print odmlEX
Document 42 {author = D. N. Adams, 2 sections}
```

As you can see, the printout gives you a short summary of the Document of the loaded example odML file.

The print out gives you already the following information about the odML file:

- `Document` tells you that you are looking at an odML Document
- `42` is the user defined version of this odML file
- `{...}` provides `author` and number of attached sections

- `author` states the author of the odML file, "D. N. Adams" in the example case
- `2 sections` tells you that this odML Document has 2 Section directly appended

Note that the Document printout tells you nothing about the depth of the complete tree structure, because it is not displaying the children of its directly attached Sections. It also does not display all Document attributes. In total, a Document has the following attributes:

author

- Returns the author (returned as string) of an odML document.

date

- Returns a user defined date. Could for example be used to state the date of the document creation or the date of the latest change.

document

- Returns the current Document object.

parent

- Returns the parent object (which is `None` for a Document).

repository

- Returns the URL (returned as string) to a user defined repository of terminologies used in this Document. Could be the URL to the G-Node terminologies or to a user defined template.

version

- Returns the user defined version (returned as string) of this odML file.

id

- id is a UUID (universally unique identifier) that uniquely identifies the current document. If not otherwise specified, this id is automatically created and assigned.

Let's check out all attributes with the following commands:

```
>>> print(odmlEX.author)
D. N. Adams
>>> print(odmlEX.date)
1979-10-12
>>> print(odmlEX.document)
Document 42 {author = D. N. Adams, 2 sections}
>>> print(odmlEX.parent)
None
>>> print(odmlEX.repository)
https://terminologies.g-node.org/v1.1/terminologies.xml
>>> print(odmlEX.version)
42
```

As expected for a Document, the attributes `author` and `version` match the information given in the Document printout, the document attribute just returns the Document, and the parent attribute is `None`.

As you learned in the beginning, Sections can be attached to a Document. They represent the next hierarchy level of an odML file. Let's have a look which Sections were attached to the Document of our example odML file using the following command:

```
>>> print(odmlEX.sections)
[Section[4|2] {name = TheCrew, type = crew, id = ...},
 Section[1|7] {name = TheStarship, type = starship, id = ...}]
```

As expected from the Document printout our example contains two Sections. The printout and attributes of a Section are explained in the next chapter.

1.5.2 The Sections

There are several ways to access Sections. You can either call them by name or by index using either explicitly the function that returns the list of Sections (see last part of *The Document* chapter) or using again a short cut notation. Let's test all the different ways to access a Section, by having a look at the first Section in the sections list attached to the Document in our example odML file:

```
>>> print(odmlEX.sections['TheCrew'])
Section[4|2] {name = TheCrew, type = crew, id = ...}
>>> print(odmlEX.sections[0])
Section[4|2] {name = TheCrew, type = crew, id = ...}
>>> print(odmlEX['TheCrew'])
Section[4|2] {name = TheCrew, type = crew, id = ...}
>>> print(odmlEX[0])
Section[4|2] {name = TheCrew, type = crew, id = ...}
```

In the following we will call Sections explicitly by their name using the short cut notation.

The printout of a Section is similar to the Document printout and gives you already the following information:

- `Section` tells you that you are looking at an odML Section
- `[4|2]` states that this Section has four Sections and two Properties directly attached to it
- `{...}` provides `name`, `type` and `id` of the Section
- `name` is the name of this Section, 'TheCrew' in the example case
- `type` provides the type of the Section, 'crew' in the example case
- `id` provides the uuid of the Section, the actual value has been omitted in the example to improve readability.

Note that the Section printout tells you nothing about the depth of a possible sub-Section tree below the directly attached ones. It also only lists the type of the Section as one of the Section attributes. In total, a Section can be defined by the following attributes:

name

- Returns the name of this Section. Should indicate what kind of information can be found in this Section.

definition

- Returns the definition of the content within this Section. Should describe what kind of information can be found in this Section.

document

- Returns the Document to which this Section belongs to. Note that this attribute is set automatically for a Section and all its children when it is attached to a Document.

parent

- Returns the parent to which this Section was directly attached to. Can be either a Document or another Section.

type

- Returns the classification type which allows to connect related Sections due to a superior semantic context.

reference

- Returns a reference that can be used to state the origin or source file of the metadata stored in the Properties that are grouped by this Section.

repository

- Returns the URL (returned as string) to a user defined repository of terminologies used in this Document. Could be the URL to the G-Node terminologies or to a user defined template.

id

- id is a UUID (universally unique identifiers) that uniquely identifies the current section. If not otherwise specified, this id is automatically created and assigned.

Let's have a look at the attributes for the Section 'TheCrew':

```
>>> print(odmlEX['TheCrew'].name)
TheCrew
>>> print(odmlEX['TheCrew'].definition)
Information on the crew
>>> print(odmlEX['TheCrew'].document)
Document 42 {author = D. N. Adams, 2 sections}
>>> print(odmlEX['TheCrew'].parent)
Document 42 {author = D. N. Adams, 2 sections}
>>> print(odmlEX['TheCrew'].type)
crew
>>> print(odmlEX['TheCrew'].reference)
None
>>> print(odmlEX['TheCrew'].repository)
None
>>> print(odmlEX['TheCrew'].id)
6df940b5-b502-4749-8ad9-33d7432064f3
```

As expected for this Section, the name and type attribute match the information given in the Section printout, and the document and parent attributes return the same object, namely our example Document.

To see which Sections are directly attached to the Section 'TheCrew' again use the following command:

```
>>> print(odmlEX['TheCrew'].sections)
[Section[0|5] {name = Arthur Philip Dent, type = crew/person, id = ...},
 Section[0|5] {name = Zaphod Beeblebrox, type = crew/person, id = ...},
 Section[0|5] {name = Tricia Marie McMillan, type = crew/person, id = ...},
 Section[0|5] {name = Ford Prefect, type = crew/person, id = ...}]
```

Or, for accessing these sub-Sections:

```
>>> print(odmlEX['TheCrew'].sections['Ford Prefect'])
Section[0|5] {name = Ford Prefect, type = crew/person, id = ...}
>>> print(odmlEX['TheCrew'].sections[3])
Section[0|5] {name = Ford Prefect, type = crew/person, id = ...}
>>> print(odmlEX['TheCrew']['Ford Prefect'])
Section[0|5] {name = Ford Prefect, type = crew/person, id = ...}
>>> print(odmlEX['TheCrew'][3])
Section[0|5] {name = Ford Prefect, type = crew/person, id = ...}
```

As you learned, besides sub-Sections, a Section can also have Properties attached. Let's see which Properties are attached to the Section 'TheCrew':

```
>>> print(odmlEX['TheCrew'].properties)
[Property: {name = NameCrewMembers},
 Property: {name = NoCrewMembers}]
```

The printout and attributes of a Property are explained in the next chapter.

1.5.3 The Properties

Properties need to be called explicitly via the properties function of a Section. You can then either call a Property by name or by index:

```
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'])
Property: {name = NoCrewMembers}
>>> print(odmlEX['TheCrew'].properties[1])
Property: {name = NoCrewMembers}
```

In the following we will only call Properties explicitly by their name.

The Property printout is reduced and only gives you information about the following:

- `Property` tells you that you are looking at an odML Property
- `{...}` provides the `name` of the Property
- `NoCrewMembers` is the name of this Property

Note that the Property printout tells you nothing about the number of Values, and very little about the Property attributes. In total, a Property can be defined by the following attributes:

name

- Returns the name of the Property. Should indicate what kind of metadata are stored in this Property.

definition

- Returns the definition of this Property. Should describe what kind of metadata are stored in this Property.

document

- Returns the Document to which the parent Section of this Property belongs to. Note that this attribute is set automatically for a Section and all its children when it is attached to a Document.

parent

- Returns the parent Section to which this Property was attached to.

values

- Returns the metadata of this Property. Can be either a single metadata or multiple, but homogeneous metadata (all with the same dtype, unit and uncertainty). For this reason, the output is always provided as a list.

dtype

- Returns the odml data type of the stored metadata.

unit

- Returns the unit of the stored metadata.

uncertainty

- recommended
- Can be used to specify the uncertainty of the given metadata value.

reference

- Returns a reference that can be used to state an external definition of the metadata value.

dependency

- optional
- A name of another Property within the same section, which this property depends on.

dependency_value

- optional
- Value of the other Property specified in the 'dependency' attribute on which this Property depends on.

value_origin

- A reference to state the origin of the metadata value e.g. a file name.

Let's check which attributes were defined for the Property 'NoCrewMembers':

```
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].name)
NoCrewMembers
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].definition)
Number of crew members
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].document)
Document 42 {author = D. N. Adams, 2 sections}
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].values)
[4]
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].dtype)
int
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].unit)
None
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].uncertainty)
1
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].reference)
The Hitchhiker's guide to the Galaxy (novel)
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].dependency)
None
>>> print(odmlEX['TheCrew'].properties['NoCrewMembers'].dependency_value)
None
```

As mentioned the `values` attribute of a Property can only contain multiple metadata when they have the same `dtype` and `unit`, as it is the case for the Property 'NameCrewMembers':

```
>>> print(odmlEX['TheCrew'].properties['NameCrewMembers'].values)
['Arthur Philip Dent',
 'Zaphod Beeblebrox',
 'Tricia Marie McMillan',
 'Ford Prefect']
>>> print(odmlEX['TheCrew'].properties['NameCrewMembers'].dtype)
person
>>> print(odmlEX['TheCrew'].properties['NameCrewMembers'].unit)
None
```

NOTE: `property.values` will always return a copy! Any direct changes to the returned list will have no affect on the actual Property values. If you want to make changes to a Property value, either use the `append`, `extend` and `remove` methods or assign a new value list to the property.

1.5.4 Printing overviews to navigate the contents of an odML document

The odML entities `Property`, `Section` and `Document` feature a method that allows to print a tree-like representation of all child entities to get an overview of the file structure.

```
>>> MYodML.pprint()
>>> sec = MYodML['TheCrew']
>>> sec.pprint()
>>> prop = odmlEX['TheCrew'].properties['NameCrewMembers']
>>> prop.pprint()
```

1.6 Generating an odML-file

After getting familiar with the different odML objects and their attributes, you will now learn how to generate your own odML file by reproducing some parts of the example THGTTG.odml.

We will show you first how to create the different odML objects with their attributes. Please note that some attributes are obligatory, some are recommended and others are optional when creating the corresponding odML objects. A few are automatically generated in the process of creating an odML file. Furthermore, all attributes of an odML object can be edited at any time.

If you opened a new IPython shell, please import first again the odml package:

```
>>> import odml
```

1.6.1 Create a document

Let's start by creating the Document. Note that none of the Document attributes are obligatory:

```
>>> MYodML = odml.Document()
```

You can check if your new Document contains actually what you created by using some of the commands you learned before:

```
>>> MYodML
>>> Document None {author = None, 0 sections}
```

As you can see, we created an "empty" Document where the version and the author attributes are not defined and no section is yet attached. You will learn how to create and add a Section to a Document in the next chapter. Let's focus here on defining the Document attributes:

```
>>> MYodML.author = 'D. N. Adams'
>>> MYodML.version = 42
```

For the date attribute you require a datetime object as entry. For this reason, you need to first import the Python package datetime:

```
>>> import datetime as dt
```

Now, let's define the date attribute of the Document:

```
>>> MYodML.date = dt.date(1979, 10, 12)
```

Next, let us also add a repository attribute. Exemplary, we can import the Python package `os` to extract the absolute path to our previously used example odML file and add this as repository:

```
>>> import os
>>> url2odmlEX = 'file:///' + os.path.abspath(to_load)
>>> MYodML.repository = url2odmlEX
```

The document and parent attribute are automatically set and should not be fiddled with.

Check if your new Document actually contains all attributes now:

```
>>> print(MYodML.author)
D. N. Adams
>>> print(MYodML.date)
1979-10-12
>>> print(MYodML.document)
Document 42 {author = D. N. Adams, 0 sections}
>>> print(MYodML.parent)
None
>>> print(MYodML.repository)
file:///home/usr/.../python-odml/doc/example_odMLs/THGTTG.odml
>>> print(MYodML.version)
42
```

Note that you can also define all attributes when first creating a Document:

```
>>> MYodML = odml.Document(author='D. N. Adams',
                           version=42,
                           date=dt.date(1979, 10, 12),
                           repository=url2odmlEX)
```

Our newly created Document is, though, still "empty", because it does not contain Sections yet. Let's change this!

1.6.2 Create a section

We now create a Section by reproducing the Section "TheCrew" of the example odML file from the beginning:

```
>>> sec1 = odml.Section(name="TheCrew",
                        definition="Information on the crew",
                        type="crew")
```

Note that only the attribute name is obligatory. The attributes `definition` and `type` are recommended, but could be either not defined at all or defined later on.

Let us now attach this Section to our previously generated Document. With this, the attribute document and parent of our new Section are automatically updated:

```
>>> MYodML.append(sec1)

>>> print(MYodML)
Document 42 {author = D. N. Adams, 1 sections}
>>> print(MYodML.sections)
[Section[0|0] {name = TheCrew, type = crew, id = ...}]

>>> print(sec1.document)
Document 42 {author = D. N. Adams, 1 sections}
>>> print(sec1.parent)
Document 42 {author = D. N. Adams, 1 sections}
```

It is also possible to connect a Section directly to a parent object. Let's try this with the next Section we create:

```
>>> sec2 = odml.Section(name="Arthur Philip Dent",
                        definition="Information on Arthur Dent",
                        type="crew/person",
                        parent=sec1)

>>> print(sec2)
Section[0|0] {name = Arthur Philip Dent, type = crew/person, id = ...}

>>> print(sec2.document)
Document 42 {author = D. N. Adams, 1 sections}
>>> print(sec2.parent)
[Section[1|0] {name = TheCrew, type = crew, id = ...}
```

Note that all of our created Sections do not contain any Properties yet. Let's see if we can change this...

1.6.3 Create a Property:

Let's create our first Property:

```
>>> prop1 = odml.Property(name="Gender",
                          definition="Sex of the subject",
                          values="male")
```

Note that again, only the `name` attribute is obligatory for creating a Property. The remaining attributes can be defined later on, or are automatically generated in the process.

If a value is defined, but the `dtype` is not, as it is the case for our example above, the `dtype` is deduced automatically:

```
>>> print(prop1.dtype)
string
```

Generally, you can use the following odML data types to describe the format of the stored metadata:

dtype	required data examples
odml.DType.int or 'int'	42
odml.DType.float or 'float'	42.0
odml.DType.boolean or 'boolean'	True or False
odml.DType.string or 'string'	'Earth'
odml.DType.date or 'date'	dt.date(1979, 10, 12)
odml.DType.datetime or 'datetime'	dt.datetime(1979, 10, 12, 11, 11, 11)
odml.DType.time or 'time'	dt.time(11, 11, 11)
odml.DType.person or 'person'	'Zaphod Beeblebrox'
odml.DType.text or 'text'	'any text containing n linebreaks'
odml.DType.url or 'url'	"https://en.wikipedia.org/wiki/Earth"
odml.DType.tuple	"(39.12; 67.19)" cf. usage note below

The available types are implemented in the 'odml.dtypes' Module. Note that the last four data types, if not defined, cannot be deduced, but are instead always interpreted as string.

If we append now our new Property to the previously created sub-Section 'Arthur Philip Dent', the Property will also inherit the document attribute and automatically update its parent attribute:

```
>>> MYodML['TheCrew']['Arthur Philip Dent'].append(prop1)
```

(continues on next page)

(continued from previous page)

```
>>> print(prop1.document)
Document 42 {author = D. N. Adams, 1 sections}
>>> print(prop1.parent)
Section[0|1] {name = Arthur Philip Dent, type = crew/person, id = ...}
```

Next, let us create a Property with multiple metadata entries:

```
>>> prop2 = odml.Property(name="NameCrewMembers",
                          definition="List of crew members names",
                          values=["Arthur Philip Dent",
                                  "Zaphod Beeblebrox",
                                  "Tricia Marie McMillan",
                                  "Ford Prefect"],
                          dtype=odml.DType.person)
```

As you learned before, in such a case the metadata entries must be homogeneous! That means they have to be of the same `dtype`, `unit`, and `uncertainty` (here `odml.DType.person`, None, and None, respectively).

To further build up our odML file, let us attach now this new Property to the previously created Section 'TheCrew':

```
>>> MYodML['TheCrew'].append(prop2)
```

Note that it is also possible to add a metadata entry later on:

```
>>> prop2.append("Blind Passenger")
>>> print(MYodML['TheCrew'].properties['NameCrewMembers'].values)
['Arthur Philip Dent',
 'Zaphod Beeblebrox',
 'Tricia Marie McMillan',
 'Ford Prefect',
 'Blind Passenger']
```

The `tuple` datatype you might have noticed in the dtype table above has to be specially handled. It is intended to enforce a specific number of data points for each value entry. This is useful in case of 2D or 3D data, where all data points always have to be present for each entry. The dtype itself has to contain the number corresponding to the required value data points. For the value data points themselves, they have to be enclosed by brackets and separated by a semicolon.

```
>>> pixel_prop = odml.Property(name="pixel map")
>>> pixel_prop.dtype = "2-tuple"
>>> pixel_prop.values = ["(1; 2)", "(3; 4)"]
```

```
>>> voxel_prop = odml.Property(name="voxel map")
>>> voxel_prop.dtype = "3-tuple"
>>> voxel_prop.values = "(1; 2; 3)"
```

Please note, that inconsistent tuple values will raise an error:

```
>>> tprop = odml.Property(name="tuple fail")
>>> tprop.dtype = "3-tuple"
>>> tprop.values = ["(1; 2)"]
```

1.6.4 Printing the XML-representation of an odML file:

Although the XML-representation of an odML file is a bit hard to read, it is sometimes helpful to check, especially during a generation process, how the hierarchical structure of the odML file looks like.

Let's have a look at the XML-representation of our small odML file we just generated:

```
>>> print(odml.tools.xmlparser.XMLWriter(MYodML))
<odML version="1.1">
  <date>1979-10-12</date>
  <section>
    <definition>Information on the crew</definition>
    <property>
      <definition>List of crew members names</definition>
      <name>NameCrewMembers</name>
      <type>person</type>
      <value>[Arthur Philip Dent,Zaphod Beeblebrox,Tricia Marie McMillan,Ford Prefect,
↪Blind Passenger&#13;]</value>
    </property>
    <name>TheCrew</name>
    <section>
      <definition>Information on Arthur Dent</definition>
      <property>
        <definition>Sex of the subject</definition>
        <name>Gender</name>
        <type>string</type>
        <value>[male&#13;]</value>
      </property>
      <name>Arthur Philip Dent</name>
      <type>crew/person</type>
    </section>
    <type>crew</type>
  </section>
  <version>42</version>
  <repository>file:///home/usr/Projects/toolbox/python-odml/doc/example_odMLs/THGTTG.
↪odml</repository>
  <author>D. N. Adams</author>
</odML>
```

1.6.5 Saving an odML file:

You can save your odML file using the following command:

```
>>> save_to = '/home/usr/toolbox/python-odml/doc/example_odMLs/myodml.odml'
>>> odml.save(MYodML, save_to)
```

By default, every odML file will be saved using the `XML` file format. Note, that you can also choose to save an odML Document using the `JSON` or the `YAML` file format as well, specifying the corresponding option in the command.

```
>>> save_to = '/home/usr/toolbox/python-odml/doc/example_odMLs/myodml.json'
>>> odml.save(MYodML, save_to, "JSON")
>>> save_to = '/home/usr/toolbox/python-odml/doc/example_odMLs/myodml.yaml'
>>> odml.save(MYodML, save_to, "YAML")
```

1.6.6 Loading an odML file:

You already learned how to load the example odML file. Here just as a reminder you can try to reload your own saved odML file:

```
>>> my_reloaded_odml = odml.load(save_to)
```

Again, the load function by default assumes, that an odML file was saved using the `XML` format. If it was saved in either `JSON` or `YAML`, add the appropriate format option when loading the document:

```
>>> my_reloaded_odml_json = odml.load(save_to, "JSON")
>>> my_reloaded_odml_yaml = odml.load(save_to, "YAML")
```

1.6.7 Advanced Value features

Data type conversions

After creating a Property with metadata, the data type can be changed and the format of the corresponding entry will be converted to the new data type, if the new type is valid for the given metadata:

```
>>> test_dtype_conv = odml.Property('p', values=1.0)
>>> print(test_dtype_conv.values)
[1.0]
>>> print(test_dtype_conv.dtype)
float
>>> test_dtype_conv.dtype = odml.DType.int
>>> print(test_dtype_conv.values)
[1]
>>> print(test_dtype_conv.dtype)
int
```

If the conversion is invalid, a `ValueError` is raised.

Also note, that during such a process metadata loss may occur, if a float is converted to integer and then back to float:

```
>>> test_dtype_conv = odml.Property('p', values=42.42)
>>> print(test_dtype_conv.values)
[42.42]
>>> test_dtype_conv.dtype = odml.DType.int
>>> test_dtype_conv.dtype = odml.DType.float
>>> print(test_dtype_conv.values)
[42.0]
```

Links & Includes

Sections can be linked to other Sections, so that they include their defined attributes. A link can be within the document (`link` property) or to an external one (`include` property).

After parsing a document, these links are not yet resolved, but can be using the `odml.doc.BaseDocument.finalize` method:

```
>>> d = xmlparser.load("sample.odml")
>>> d.finalize()
```

Note: Only the parser does not automatically resolve link properties, as the referenced sections may not yet be available. However, when manually setting the `link` (or `include`) attribute, it will be immediately resolved. To avoid this behaviour, set the `_link` (or `_include`) attribute instead. The object remembers to which one it is linked in its `_merged` attribute. The link can be unresolved manually using `odml.section.BaseSection.unmerge` and merged again using `odml.section.BaseSection.merge`.

Unresolving means to remove sections and properties that do not differ from their linked equivalents. This should be done globally before saving using the `odml.doc.BaseDocument.clean` method:

```
>>> d.clean()
>>> xmlparser.XMLWriter(d).write_file('sample.odml')
```

Changing a `link` (or `include`) attribute will first unmerge the section and then set merge with the new object.

Terminologies

odML supports terminologies that are data structure templates for typical use cases. Sections can have a `repository` attribute. As repositories can be inherited, the current applicable one can be obtained using the `odml.section.BaseSection.get_repository` method.

To see whether an object has a terminology equivalent, use the `odml.property.BaseProperty.get_terminology_equivalent` method, which returns the corresponding object of the terminology.

CHAPTER 2

Advanced odML features

2.1 Working with odML Validations

odML Validations are a set of pre-defined checks that are run against an odML document automatically when it is saved or loaded. A document cannot be saved, if a Validation fails a check that is classified as an Error. Most validation checks are Warnings that are supposed to raise the overall data quality of the odml Document.

When an odML document is saved or loaded, tha automatic validation will print a short report of encountered Validation Warnings and it is up to the user whether they want to resolve the Warnings. The odML document provides the `validate` method to gain easy access to the default validations. A Validation in turn provides not only a specific description of all encountered warnings or errors within an odML document, but it also provides direct access to each and every odML entity i.e. an `odml.Section` or an `odml.Property` where an issue has been found. This enables the user to quickly access and fix an encountered issue.

A minimal example shows how a workflow using default validations might look like:

```
>>> # Create a minimal document with Section issues: name and type are not assigned
>>> doc = odml.Document()
>>> sec = odml.Section(parent=doc)
>>> odml.save(doc, "validation_example.odml.xml")
```

This minimal example document will be saved, but will also print the following Validation report:

```
>>> UserWarning: The saved Document contains unresolved issues. Run the Documents
↪'validate' method to access them.
>>> Validation found 0 errors and 2 warnings in 1 Sections and 0 Properties.
```

To fix the encountered warnings, users can access the validation via the documents' `validate` method:

```
>>> validation = doc.validate()
>>> for issue in validation.errors:
>>>     print(issue)
```

This will show that the validation has encountered two Warnings and also displays the offending odml entity.

```
>>> ValidationWarning: Section[73f29acd-16ae-47af-afc7-371d57898e28] 'Section type␣
↪not specified'
>>> ValidationWarning: Section[73f29acd-16ae-47af-afc7-371d57898e28] 'Name not␣
↪assigned'
```

To fix the "Name not assigned" warning the Section can be accessed via the validation entry and used to directly assign a human readable name to the Section in the original document. Re-running the validation will show, that the warning has been removed.

```
>>> validation.errors[1].obj.name = "validation_example_section"
>>> # Check that the section name has been changed in the document
>>> print(doc.sections)
>>> # Re-running validation
>>> validation = doc.validate()
>>> for issue in validation.errors:
>>>     print(issue)
```

Similarly the second validation warning can be resolved before saving the document again.

Please note that the automatic validation is run whenever a document is saved or loaded using the `odml.save` and `odml.load` functions as well as the `ODMLWriter` or the `ODMLReader` class. The validation is not run when using any of the lower level `xmlparser`, `dict_parser` or `rdf_converter` classes.

2.1.1 List of available default validations

The following contains a list of the default odml validations, their message and the suggested course of action to resolve the issue.

Validation: `object_required_attributes`
Message: "Missing required attribute 'xyz'"
Applies to: `Document`, `Section`, `Property`
Course of action: Add an appropriate value to attribute 'xyz' for the reported odml entity.

Validation: `section_type_must_be_defined`
Message: "Section type not specified"
Applies to: `Section`
Course of action: Fill in the `type` attribute of the reported Section.

Validation: `section_unique_ids`
Message: "Duplicate id in Section 'secA' and 'secB'"
Applies to: `Section`
Course of action: IDs have to be unique and a duplicate id was found. Assign a new id for the reported Section.

Validation: `property_unique_ids`
Message: "Duplicate id in Property 'propA' and 'propB'"
Applies to: `Property`
Course of action: IDs have to be unique and a duplicate id was found. Assign a new id for the reported Property

Validation: `section_unique_name_type`
Message: "name/type combination must be unique"
Applies to: `Section`
Course of action: The combination of Section.name and Section.type has to be unique on the same level. Change either name or type of the reported Section.

Validation: `object_unique_name`
Message: "Object names must be unique"
Applies to: `Document`, `Section`, `Property`
Course of action: Property name has to be unique on the same level. Change the name of the reported Property.

Validation: `object_name_readable`
Message: "Name not assigned"
Applies to: `Section`, `Property`
Course of action: When Section or Property names are left empty on creation or set to None, they are automatically assigned the entities uuid. Assign a human readable name to the reported entity.

Validation: `property_terminology_check`
Message: "Property 'prop' not found in terminology"
Applies to: `Property`
Course of action: The reported entity is linked to a repository but the repository is not available. Check if the linked content has moved.

Validation: `property_dependency_check`
Message: "Property refers to a non-existent dependency object" or "Dependency-value is not equal to value of the property's dependency"
Applies to: `Property`
Course of action: The reported entity depends on another Property, but this dependency has not been satisfied. Check the referenced Property and its value to resolve the issue.

Validation: `property_values_check`
Message: "Tuple of length 'x' not consistent with dtype 'dtype'!" or "Property values not of consistent dtype!".
Applies to: `Property`
Course of action: Adjust the values or the dtype of the referenced Propery.

Validation: `property_values_string_check`
Message: "Dtype of property "prop" currently is "string", but might fit dtype "dtype"!"
Applies to: `Property`
Course of action: Check if the datatype of the referenced Property.values has been loaded correctly and change the Property.dtype if required.

Validation: `section_properties_cardinality`
Message: "cardinality violated x values, y found)"

Applies to: `Section`
Course of action: A cardinality defined for the number of Properties of a Section does not match. Add or remove Properties until the cardinality has been satisfied or adjust the cardinality.

Validation: `section_sections_cardinality`
Message: "cardinality violated x values, y found)"
Applies to: `Section`
Course of action: A cardinality defined for the number of Sections of a Section does not match. Add or remove Sections until the cardinality has been satisfied or adjust the cardinality.

Validation: `property_values_cardinality`
Message: "cardinality violated x values, y found)"
Applies to: `Property`
Course of action: A cardinality defined for the number of Values of a Property does not match. Add or remove Values until the cardinality has been satisfied or adjust the cardinality.

Validation: `section_repository_present`
Message: "A section should have an associated repository" or "Could not load terminology" or "Section type not found in terminology"
Applies to: `Section`
Course of action: Optional validation. Will report any section that does not specify a repository. Add a repository to the reported Section to resolve.

2.1.2 Custom validations

Users can write their own validation and register them either with the default validation or add it to their own validation class instance.

A custom validation handler needs to `yield` a `ValidationError`. See the `validation.ValidationError` class for details.

Custom validation handlers can be registered to be applied on "odML" (the odml Document), "section" or "property".

```
>>> import odml
>>> import odml.validation as oval
>>>
>>> # Create an example document
>>> doc = odml.Document()
>>> sec_valid = odml.Section(name="Recording-20200505", parent=doc)
>>> sec_invalid = odml.Section(name="Movie-20200505", parent=doc)
>>> subsec = odml.Section(name="Sub-Movie-20200505", parent=sec_valid)
>>>
>>> # Define a validation handler that yields a ValidationError if a section name␣
↪does not start with 'Recording-'
>>> def custom_validation_handler(obj):
>>>     validation_id = oval.IssueID.custom_validation
>>>     msg = "Section name does not start with 'Recording-'"
>>>     if not obj.name.startswith("Recording-"):
>>>         yield oval.ValidationError(obj, msg, oval.LABEL_ERROR, validation_id)
>>>
```

(continues on next page)

(continued from previous page)

```
>>> # Create a custom, empty validation with an odML document 'doc'
>>> custom_validation = oval.Validation(doc, reset=True)
>>> # Register a custom validation handler that should be applied on all Sections of␣
↪a Document
>>> custom_validation.register_custom_handler("section", custom_validation_handler)
>>> # Run the custom validation and return a report
>>> custom_validation.report()
>>> # Display the errors reported by the validation
>>> print(custom_validation.errors)
```

2.2 Defining and working with feature cardinality

The odML format allows users to define a cardinality for the number of subsections and properties of Sections and the number of values a Property might have.

A cardinality is checked when it is set, when its target is set and when a document is saved or loaded. If a specific cardinality is violated, a corresponding warning will be printed.

2.2.1 Setting a cardinality

A cardinality can be set for sections or properties of sections or for values of properties. By default every cardinality is None, but it can be set to a defined minimal and/or a maximal number of an element.

A cardinality is set via its convenience method:

```
>>> # Set the cardinality of the properties of a Section 'sec' to
>>> # a maximum of 5 elements.
>>> sec = odml.Section(name="cardinality", type="test")
>>> sec.set_properties_cardinality(max_val=5)
```

```
>>> # Set the cardinality of the subsections of Section 'sec' to
>>> # a minimum of one and a maximum of 2 elements.
>>> sec.set_sections_cardinality(min_val=1, max_val=2)
```

```
>>> # Set the cardinality of the values of a Property 'prop' to
>>> # a minimum of 1 element.
>>> prop = odml.Property(name="cardinality")
>>> prop.set_values_cardinality(min_val=1)
```

```
>>> # Re-set the cardinality of the values of a Property 'prop' to not set.
>>> prop.set_values_cardinality()
>>> # or
>>> prop.val_cardinality = None
```

Please note that a set cardinality is not enforced. Users can set less or more entities than are specified allowed via a cardinality. Instead whenever a cardinality is not met, a warning message is displayed and any unment cardinality will show up as a Validation warning message whenever a document is saved or loaded.

2.3 View odML documents in a web browser

By default all odML files are saved in the XML format without the capability to view the plain files in a browser. By default you can use the command line tool `odmlview` to view saved odML files locally. Since this requires the start of a local server, there is another option to view odML XML files in a web browser.

You can use an additional feature of the `odml.tools.XMLWriter` to save an odML document with an embedded default stylesheet for local viewing:

```
>>> import odml
>>> from odml.tools import XMLWriter
>>> doc = odml.Document() # minimal example document
>>> filename = "viewable_document.xml"
>>> XMLWriter(doc).write_file(filename, local_style=True)
```

Now you can open the resulting file 'viewable_document.xml' in any current web-browser and it will render the content of the odML file.

If you want to use a custom style sheet to render an odML document instead of the default one, you can provide it as a string to the XML writer. Please note, that it cannot be a full XSL stylesheet, the outermost tag of the XSL code has to be `<xsl:template match="odML"> [your custom style here] </xsl:template>`:

```
>>> import odml
>>> from odml.tools import XMLWriter
>>> doc = odml.Document() # minimal example document
>>> filename = "viewable_document.xml"
>>> own_template = """<xsl:template match="odML"> [your custom style here] </
↪xsl:template>"""
>>> XMLWriter(doc).write_file(filename, custom_template=own_template)
```

Please note that if the file is saved using the '.odml' extension and you are using Chrome, you will need to map the '.odml' extension to the browsers Mime-type database as 'application/xml'.

Also note that any style that is saved with an odML document will be lost, when this document is loaded again and changes to the content are added. In this case the required style needs to be specified again when saving the changed file as described above.

CHAPTER 3

odML and RDF - Export and usage

3.1 Semantic Web and graph database searches

Searches within odML documents are part of the library implementation and imports from linked, external sources into odML documents can be easily done with the core library functionality. With the option to export odML documents to the RDF format, users also gain the option to search across multiple documents using tools from the Semantic Web technology.

If you are unfamiliar with it, we linked additional information to the Semantic web, RDF and SPARQL for your convenience and give the briefest introduction below.

RDF was designed by the World Wide Web Consortium (W3C) as a standard model for data representation and exchange on the web with the heterogeneity of data in mind. Even tough the RDF file format might vary, the underlying concept features two key points. The first is that information is structured in subject-predicate-object triples e.g. “apple hasColor red”. The second key point is that multiple subjects and objects can be connected to form a graph e.g. “tree hasFruit apple” can be combined with the previous example to form a minimal graph. These graphs can contain very heterogeneous data, but can still be queried using the SPARQL query language due to the semantic structure of the underlying data.

3.2 odML to RDF export

Without further ado the next sections will expose you to the range of odML to RDF features the core library provides. To check how to create a graph database from exported odML documents and how to query such a database please refer to the section below.

3.2.1 Default odML to XML RDF export

Once an odML document is available, it can most easily be exported to RDF by the odml.save feature.

Given below is a minimal example:

```
import odml

doc = odml.Document()
sec = odml.Section(name="rdf_export_section", parent=doc)
prop = odml.Property(name="rdf_export_property", parent=sec)

odml.save(doc, "./rdf_export.rdf", "RDF")
```

This will export the odML document to the RDF format in the XML flavor and will save it to the file *./rdf_export.rdf*. The content of the file will look something like this (the UUIDs of the individual nodes will differ):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns:odml="https://g-node.org/odml-rdf#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="https://g-node.org/odml-rdf#281c5aa7-8fea-4852-85ec-
↪db127f753647">
    <odml:hasName>rdf_export_property</odml:hasName>
    <rdf:type rdf:resource="https://g-node.org/odml-rdf#Property"/>
  </rdf:Description>
  <rdf:Description rdf:about="https://g-node.org/odml-rdf#08c6e31a-533f-443b-acd2-
↪8e961215d38e">
    <odml:hasSection rdf:resource="https://g-node.org/odml-rdf#eebe4bf7-af10-4321-
↪87ec-2cdf77289478"/>
    <odml:hasFileName>None</odml:hasFileName>
    <rdf:type rdf:resource="https://g-node.org/odml-rdf#Document"/>
  </rdf:Description>
  <rdf:Description rdf:about="https://g-node.org/odml-rdf#eebe4bf7-af10-4321-87ec-
↪2cdf77289478">
    <odml:hasName>rdf_export_section</odml:hasName>
    <odml:hasType>n.s.</odml:hasType>
    <odml:hasProperty rdf:resource="https://g-node.org/odml-rdf#281c5aa7-8fea-4852-
↪85ec-db127f753647"/>
    <rdf:type rdf:resource="https://g-node.org/odml-rdf#Section"/>
  </rdf:Description>
  <rdf:Description rdf:about="https://g-node.org/odml-rdf#Hub">
    <odml:hasDocument rdf:resource="https://g-node.org/odml-rdf#08c6e31a-533f-443b-
↪acd2-8e961215d38e"/>
  </rdf:Description>
</rdf:RDF>
```

3.2.2 Specific RDF format export

The `RDFWriter` class is used to convert odML documents to one of the supported RDF formats:

`xml, pretty-xml, trix, n3, turtle, ttl, ntriples, nt, nt11, trig`

`turtle` is the format that is best suited for storage and human readability while for cross-tool usage, saving RDF in its `XML` variant is probably the safest choice.

The exported output can be returned as a string:

```
from odml.tools.rdf_converter import RDFWriter

print(RDFWriter(doc).get_rdf_str('turtle'))
```

This will print the content of the odML document in the Turtle flavor of RDF:

```
@prefix odml: <https://g-node.org/odml-rdf#> .

odml:Hub odml:hasDocument odml:08c6e31a-533f-443b-acd2-8e961215d38e .

odml:08c6e31a-533f-443b-acd2-8e961215d38e a odml:Document ;
    odml:hasFileName "None" ;
    odml:hasSection odml:eebe4bf7-af10-4321-87ec-2cdf77289478 .

odml:281c5aa7-8fea-4852-85ec-db127f753647 a odml:Property ;
    odml:hasName "rdf_export_property" .

odml:eebe4bf7-af10-4321-87ec-2cdf77289478 a odml:Section ;
    odml:hasName "rdf_export_section" ;
    odml:hasProperty odml:281c5aa7-8fea-4852-85ec-db127f753647 ;
    odml:hasType "n.s." .
```

The output can of course also be written to a file with a specified RDF output format; the output file will autmatically be assigned the appropriate file ending:

```
from odml.tools.rdf_converter import RDFWriter

RDFWriter(doc).write_file("./rdf_export_turtle", "turtle")
```

All available RDF output formats can be viewed via `odml.tools.parser_utils.RDF_CONVERSION_FORMATS.keys()`.

3.2.3 Bulk export to XML RDF

Existing odML files can be exported to XML RDF in bulk using the `odmltordf` command line tool that is automatically installed with the core library.

odmlToRDF searches for odML files within a provided SEARCHDIR and converts them to the newest odML format version and exports all found and resulting odML files to XML formatted RDF. Original files will never be overwritten. New files will be written either to a new directory at the current or a specified location.

Usage: odmltordf [-r] [-o OUT] SEARCHDIR

The command line option `-r` enables recursive search, `-o OUT` specifies a dedicated output folder for the created output files.

3.3 Advanced features

3.3.1 RDF subclassing of odml.Section.type

By default a set of pre-defined odml.Section.types will export Sections not as an odml:Section but as a specific RDF subclass of an odml:Section. This is meant to simplify SPARQL query searches on graph databases that contain odml specific RDF.

As an example an odml.Section normally gets exported as RDF class type odml-rdf:Section:

```
<rdf:type rdf:resource="https://g-node.org/odml-rdf#Section"/>
```

An odml.Section with the odml.Section.type=”protocol” will by default be exported as a different RDF class type:

```
<rdf:type rdf:resource="https://g-node.org/odml-rdf#Protocol"/>
```

In an RDF query this can now be searched for directly by asking for RDF class "odml-rdf:Protocol" instead of asking for RDF class "odml-rdf:Section" with type "Protocol".

On install the core library already provides a list of odml.Section.type mappings to RDF subclasses. On initialisation the `RDFWriter` loads all subclasses that are available and uses them by default when exporting an odML document to RDF. The available terms and the mappings of odml.Section.types to RDF subclasses can be viewed by accessing the `section_subclasses` attribute of an initialised `RDFWriter`:

```
rdf_export = RDFWriter(doc)
rdf_export.section_subclasses
```

This export also adds all used subclass definitions to the resulting file to enable query reasoners to makes sense of the introduced subclasses upon a query.

Currently the following mappings of `odml.Section.type` values to odml-rdf:Section subclass are available:

```
analysis: Analysis
analysis/power_spectrum: PowerSpectrum
analysis/psth: PSTH
cell: Cell
datacite/alternate_identifier: AlternateIdentifier
datacite/contributor: Contributer
datacite/contributor/affiliation: Affiliation
datacite/contributor/named_identifier: NamedIdentifier
datacite/creator: Creator
datacite/creator/affiliation: Affiliation
datacite/creator/named_identifier: NamedIdentifier
datacite/date: Date
datacite/description: Description
datacite/format: Format
datacite/funding_reference: FundingReference
datacite/geo_location: GeoLocation
datacite/identifier: Identifier
datacite/related_identifier: RelatedIdentifier
datacite/resource_type: ResourceType
datacite/rights: Rights
datacite/size: Size
datacite/subject: Subject
datacite/title: Title
dataset: Dataset
data_reference: DataReference
blackrock: Blackrock
electrode: Electrode
event: Event
event_list: EventList
experiment: Experiment
experiment/behavior: Behavior
experiment/electrophysiology: Electrophysiology
experiment/imaging: Imaging
experiment/psychophysics: Psychophysics
hardware_properties: HardwareProperties
hardware_settings: HardwareSettings
hardware: Hardware
hardware/amplifier: Amplifier
hardware/attenuator: Attenuator
hardware/camera_objective: CameraObjective
```

(continues on next page)

(continued from previous page)

```
hardware/daq: DataAcquisition
hardware/eyetracker: Eyetracker
hardware/filter: Filter
hardware/filter_set: Filterset
hardware/iaq: ImageAcquisition
hardware/light_source: Lightsource
hardware/microscope: Microscope
hardware/microscope_objective: MicroscopeObjective
hardware/scanner: Scanner
hardware/stimulus_isolator: StimulusIsolator
model/lif: LeakyIntegrateAndFire
model/pif: PerfectIntegrateAndFire
model/multi_compartment: MultiCompartmentModel
model/single_compartment: SingleCompartmentModel
person: Person
preparation: Preparation
project: Project
protocol: Protocol
recording: Recording
setup: Setup
stimulus: Stimulus
stimulus/dc: DC
stimulus/gabor: Gabor
stimulus/grating: Grating
stimulus/pulse: Pulse
stimulus/movie: Movie
stimulus/ramp: Ramp
stimulus/random_dot: RandomDot
stimulus/sawtooth: Sawtooth
stimulus/sine_wave: Sinewave
stimulus/square_wave: Squarewave
stimulus/white_noise: Whitenoise
subject: Subject
```

3.3.2 Custom RDF subclassing

The default list of odml.Section.types can be supplemented or even replaced by custom type to RDF subclass mappings.

All required is to provide a dictionary of the format `{"odml.Section.type value": "RDF subclass value"}`. Please note that the `odml.Section.type` value should be provided lower case, while the `RDF subclass value` should be provided upper case:

```
custom_class_dict = {"species": "Species", "cell": "Neuron"}
rdf_export = RDFWriter(doc, custom_subclasses=custom_class_dict)
```

Please note that entries in a custom subclass dictionary will overwrite entries in the default subclass dictionary.

3.3.3 Disable RDF subclassing

The subclassing feature can be disabled to export all odml.Sections as plain odml-rdf:Sections instead. This might be necessary if for e.g. a graph database is used that does not provide proper SPARQL reasoning and cannot make sense of RDF subclasses:

```
rdf_export = RDFWriter(doc, rdf_subclassing=False)
```

3.4 odML RDF graph search

The following section gives a basic example how multiple odML RDF files can be loaded into a single graph database (a so called "triple store") and how queries can be done to retrieve information from such a database.

Please note, that the rdflib library provides just basic implementation of a triple store and query features via SPARQL. To make full use of SPARQL additional RDF reasoning libraries are required. In our case the owlrl library is used to provide proper reasoning and enable searches for the RDF subclassing feature.

The following is a basic example how to load odML RDF documents into a single graph and provide the required to namespace to make the odml specific content of the graph accessible:

```
import odml

file_A = "./rdf_recordings.rdf"
file_B = "./rdf_protocols.rdf"

doc_A = odml.Document(author="MS")
sec_A = odml.Section(name="recording_A", type="paradigm_A", parent=doc_A)
_ = odml.Property(name="protocol", values="recording_protocol_A", parent=sec_A)
sec_B = odml.Section(name="recording_B", type="paradigm_A", parent=doc_A)
_ = odml.Property(name="protocol", values="recording_protocol_B", parent=sec_B)
_ = odml.Section(name="analysis_A", type="paradigm_A", parent=doc_A)

odml.save(doc_A, file_A, "RDF")

doc_B = odml.Document(author="MS")
_ = odml.Section(name="recording_protocol_A", type="protocol", parent=doc_B)
_ = odml.Section(name="recording_protocol_B", type="protocol", parent=doc_B)

odml.save(doc_B, file_B, "RDF")
```

Please note, that every odML Document exported to RDF has a special `odml-rdf:Hub` node at the very root of the document. This node is identical in every exported odML Document and is used as the root Node connecting all individual odML RDF documents into a single graph.

The documents saved above can now be loaded into single graph:

```
from rdflib import Graph

curr_graph = Graph()
curr_graph.parse(file_A)
curr_graph.parse(file_B)
```

The graph is now ready to accept simple SPARQL queries. Queries need the odML RDF namespace though to process the odml specific entries:

```
from odml.tools.rdf_converter import ODML_NS

from rdflib import Namespace, RDF, RDFS
from rdflib.plugins.sparql import prepareQuery

# preparing the query namespace
```

(continues on next page)

(continued from previous page)

```
NAMESPACE_MAP = {"odml": Namespace(ODML_NS), "rdf": RDF, "rdfs": RDFS}

# preparing a query requesting the name of all sections in the graph
q_string = "SELECT * WHERE {?s rdf:type odml:Section . ?s odml:hasName ?sec_name .}"
sec_query = prepareQuery(q_string, initNs=NAMESPACE_MAP)

for row in curr_graph.query(sec_query):
    print("Section name: '%s'" % row.sec_name)
```

The query returns:

```
Section name: 'recording_A'
Section name: 'recording_B'
Section name: 'analysis_A'
```

This query returns all sections from the first file, since reasoning is not yet enabled. This can be changed by adding reasioning to the query:

```
from owlrl import DeductiveClosure, RDFS_Semantics

DeductiveClosure(RDFS_Semantics).expand(curr_graph)

for row in curr_graph.query(sec_query):
    print("Section name: '%s'" % row.sec_name)
```

This query now returns the sections from both files:

```
Section name: 'recording_B'
Section name: 'recording_A'
Section name: 'recording_protocol_B'
Section name: 'recording_protocol_A'
Section name: 'analysis_A'
```

CHAPTER 4

Class-Reference

4.1 odML-Base Classes

These classes are the core data-structures of odML. To sum things up, an odML-Document consists of several Sections. Each Section may contain other Sections and Properties. Again each Property can have multiple Values.

The odml Module contains wrappers, that are shortcuts for creating the main objects:

```
>>> from odml import Document, Section, Property
>>> Document(version=0.9, author="Kermit")
<Doc 0.9 by Kermit (0 sections)>
```

Several modules exist to extend the implementation. The ones included in the library are those:

4.1.1 Document

class odml.doc.**BaseDocument**(*author=None*, *date=None*, *version=None*, *repository=None*, *oid=None*)

A representation of an odML document in memory. Its odml attributes are: *author*, *date*, *version* and *repository*. A Document behaves very much like a section, except that it cannot hold properties.

append(*section*)

Method appends a single Section to the section child-lists of the current Object.

Parameters **section** – odML Section object.

author

The author of the document.

clean()

Runs clean() on all immediate child-sections causing any resolved links or includes to be unresolved.

This should be called for the document prior to saving.

clone(*children=True*, *keep_id=False*)

Clones this object recursively allowing to copy it independently to another document.

contains(*obj*)
Checks if a subsection of name&type of *obj* is a child of this section if so return this child

create_section(*name=None*, *type='n.s.'*, *oid=None*, *definition=None*, *reference=None*, *repository=None*, *link=None*, *include=None*)
Creates a new subsection that is a child of this section.

Parameters

- **name** – The name of the section to create. If the name is not provided, the object id of the Section is assigned as its name. Section name is a required attribute.
- **type** – String providing a grouping description for similar Sections. Section type is a required attribute and will be set to the string 'n.s.' by default.
- **oid** – object id, UUID string as specified in RFC 4122. If no id is provided, an id will be generated and assigned.
- **definition** – String defining this Section.
- **reference** – A reference (e.g. an URL) to an external definition of the Section.
- **repository** – URL to a repository in which the Section is defined.
- **link** – Specifies a soft link, i.e. a path within the document.
- **include** – Specifies an arbitrary URL. Can only be used if *link* is not set.

Returns The new section.

date
The date the document was created.

document
Returns the parent-most node (if its a document instance) or None.

extend(*sec_list*)
Method adds Sections to the section child-list of the current object.

Parameters **sec_list** – Iterable containing odML Section entries.

finalize()
This needs to be called after the document is set up from parsing it will perform additional operations, that need the complete document. In particular, this method will resolve all *link* and *include* attributes accordingly.

find(*key=None*, *type=None*, *findAll=False*, *include_subtype=False*)
Returns the first subsection named *key* of type *type*.

Parameters

- **key** – string to search against an odML objects name.
- **type** – type of an odML object.
- **findAll** – include further matches after the first one in the result.
- **include_subtype** – splits an objects type at '/' and matches the parts against the provided type.

find_related(*key=None*, *type=None*, *children=True*, *siblings=True*, *parents=True*, *recursive=True*, *findAll=False*)
Finds a related section named *key* and/or *type*

- by searching its children's children if *children* is True if *recursive* is true all leave nodes will be searched

- by searching its siblings if *siblings* is True
- by searching the parent element if *parents* is True if *recursive* is True all parent nodes until the root are searched
- if *findAll* is True, returns a list of all matching objects

format()
Returns the format class of the current object.

get_path()
Returns the absolute path of this section.

get_property_by_path(*path*)
Find a Property through a path like "../name1/name2:property_name"

Parameters **path** (*str*) – path like "../name1/name2:property_name"

get_relative_path(*section*)
Returns a relative (file)path to point to section like (e.g. ../other_section)

If the common parent of both sections is the document (i.e. /), return an absolute path.

get_repository()
Return the current applicable repository (may be inherited from a parent) or None

get_section_by_path(*path*)
Find a Section through a path like "../name1/name2"

Parameters **path** (*str*) – path like "../name1/name2"

get_terminology_equivalent()
Returns the equivalent object in an terminology (should there be one defined) or None

id
The uuid for the document.

insert(*position*, *section*)
Insert a Section at the child-list position. A ValueError will be raised, if a Section with the same name already exists in the child-list.

Parameters

- **position** – index at which the object should be inserted.
- **section** – odML Section object.

iterproperties(*max_depth=None*, *filter_func=<function Sectionable.<lambda>>*)
Iterate each related property (recursively)

Example: return all children properties which name contains "foo" >>> filter_func = lambda x: getattr(x, 'name').find("foo") > -1 >>> sec_or_doc.iterproperties(filter_func=filter_func)

Parameters

- **max_depth** (*bool*) – iterate all properties recursively if None, only to a certain level otherwise
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True

itersections(*recursive=True*, *yield_self=False*, *filter_func=<function Sectionable.<lambda>>*, *max_depth=None*)
Iterate each child section

Example: return all subsections which name contains "foo" >>> filter_func = lambda x: getattr(x, 'name').find("foo") > -1 >>> sec_or_doc.itersections(filter_func=filter_func)

Parameters

- **recursive** (*bool*) – iterate all child sections recursively (deprecated)
- **yield_self** (*bool*) – includes itself in the iteration
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True
- **max_depth** – number of layers in the document tree to include in the search.

itervalues(*max_depth=None*, *filter_func=<function Sectionable.<lambda>>*)

Iterate each related value (recursively)

 # Example: return all children values which string converted version has "foo"

```
>>> filter_func = lambda x: str(x).find("foo") > -1
>>> sec_or_doc.itervalues(filter_func=filter_func)
```

Parameters

- **max_depth** (*bool*) – iterate all properties recursively if None, only to a certain level otherwise
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True

new_id(*oid=None*)

new_id sets the id of the current object to a RFC 4122 compliant UUID. If an id was provided, it is assigned if it is RFC 4122 UUID format compliant. If no id was provided, a new UUID is generated and assigned. :param oid: UUID string as specified in RFC 4122.

oid

The uuid for the document. Required for entity creation and comparison, saving and loading.

origin_file_name

If available, the file name from where the document has been loaded. Will not be serialized to file when saving the document.

parent

The parent of a document is always None.

pprint(*indent=2*, *max_depth=1*, *max_length=80*, *current_depth=0*)

Pretty print method to visualize Document-Section trees.

Parameters

- **indent** – number of leading spaces for every child Section or Property.
- **max_depth** – maximum number of hierarchical levels printed from the starting Section.
- **max_length** – maximum number of characters printed in one line.
- **current_depth** – number of hierarchical levels printed from the starting Section.

remove(*section*)

Removes the specified child-section

repository

A URL to a terminology.

sections
The list of sections contained in this section/document.

validate()
Runs a validation on itself and returns the Validation object.

Returns odml.Validation

version
A personal version-specifier that can be used to track different versions of the same document.

4.1.2 Section

class odml.section.**BaseSection**(*name=None*, *type='n.s.'*, *parent=None*, *definition=None*, *reference=None*, *repository=None*, *link=None*, *include=None*, *oid=None*, *sec_cardinality=None*, *prop_cardinality=None*)

An odML Section.

Parameters

- **name** – string providing the name of the Section. If the name is not provided, the object id of the Section is assigned as its name. Section name is a required attribute.
- **type** – String providing a grouping description for similar Sections. Section type is a required attribute and will be set to the string 'n.s.' by default.
- **parent** – the parent object of the new Section. If the object is not an odml.Section or an odml.Document, a ValueError is raised.
- **definition** – String defining this Section.
- **reference** – A reference (e.g. an URL) to an external definition of the Section.
- **repository** – URL to a repository in which the Section is defined.
- **link** – Specifies a soft link, i.e. a path within the document.
- **include** – Specifies an arbitrary URL. Can only be used if *link* is not set.
- **oid** – object id, UUID string as specified in RFC 4122. If no id is provided, an id will be generated and assigned. An id has to be unique within an odML Document.
- **sec_cardinality** – Section cardinality defines how many Sub-Sections are allowed for this Section. By default unlimited Sections can be set. A required number of Sections can be set by assigning a tuple of the format "(min, max)"
- **prop_cardinality** – Property cardinality defines how many Properties are allowed for this Section. By default unlimited Properties can be set. A required number of Properties can be set by assigning a tuple of the format "(min, max)".

append(*obj*)
Method adds single Sections and Properties to the respective child-lists of the current Section.

Parameters **obj** – Section or Property object.

can_be_merged
Returns True if either a *link* or an *include* attribute is specified

clean()
Runs clean() on all immediate child-sections causing any resolved links or includes to be unresolved.

This should be called for the document prior to saving.

clone(*children=True*, *keep_id=False*)
Clone this Section allowing to copy it independently to another document. By default the id of any cloned object will be set to a new uuid.

Parameters

- **children** – If True, also clone child sections and properties recursively.
- **keep_id** – If this attribute is set to True, the uuids of the Section and all child objects will remain unchanged.

Returns The cloned Section.

contains(*obj*)
If the child-lists of the current Section contain a Section with the same *name* and *type* or a Property with the same *name* as the provided object, the found Section or Property is returned.

Parameters **obj** – Section or Property object.

create_property(*name*, *values=None*, *dtype=None*, *oid=None*, *value=None*)
Create a new property that is a child of this section.

Parameters

- **name** – The name of the property.
- **values** – Some data value, it can be a single value or a list of homogeneous values.
- **dtype** – The data type of the values stored in the property, if dtype is not given, the type is deduced from the values. Check odml.DType for supported data types.
- **oid** – object id, UUID string as specified in RFC 4122. If no id is provided, an id will be generated and assigned.
- **value** – Deprecated alias of 'values'. Any content of 'value' is ignored, if 'values' is set.

Returns The new property.

create_section(*name=None*, *type='n.s.'*, *oid=None*, *definition=None*, *reference=None*, *repository=None*, *link=None*, *include=None*)
Creates a new subsection that is a child of this section.

Parameters

- **name** – The name of the section to create. If the name is not provided, the object id of the Section is assigned as its name. Section name is a required attribute.
- **type** – String providing a grouping description for similar Sections. Section type is a required attribute and will be set to the string 'n.s.' by default.
- **oid** – object id, UUID string as specified in RFC 4122. If no id is provided, an id will be generated and assigned.
- **definition** – String defining this Section.
- **reference** – A reference (e.g. an URL) to an external definition of the Section.
- **repository** – URL to a repository in which the Section is defined.
- **link** – Specifies a soft link, i.e. a path within the document.
- **include** – Specifies an arbitrary URL. Can only be used if *link* is not set.

Returns The new section.

definition
The definition of the Section.

document
Returns the parent-most node (if its a document instance) or None.

export_leaf()
Exports only the path from this section to the root. Include all properties for all sections, but no other subsections.

Returns cloned odml tree to the root of the current document.

extend(*obj_list*)
Method adds Sections and Properties to the respective child-lists of the current Section.

Parameters **obj_list** – Iterable containing Section and Property entries.

find(*key=None*, *type=None*, *findAll=False*, *include_subtype=False*)
Returns the first subsection named *key* of type *type*.

Parameters

- **key** – string to search against an odML objects name.
- **type** – type of an odML object.
- **findAll** – include further matches after the first one in the result.
- **include_subtype** – splits an objects type at '/' and matches the parts against the provided type.

find_related(*key=None*, *type=None*, *children=True*, *siblings=True*, *parents=True*, *recursive=True*, *findAll=False*)
Finds a related section named *key* and/or *type*

- by searching its children's children if *children* is True if *recursive* is true all leave nodes will be searched
- by searching its siblings if *siblings* is True
- by searching the parent element if *parents* is True if *recursive* is True all parent nodes until the root are searched
- if *findAll* is True, returns a list of all matching objects

format()
Returns the format class of the current object.

get_merged_equivalent()
Returns the merged object or None.

get_path()
Returns the absolute path of this section.

get_property_by_path(*path*)
Find a Property through a path like "../name1/name2:property_name"

Parameters **path** (*str*) – path like "../name1/name2:property_name"

get_relative_path(*section*)
Returns a relative (file)path to point to section like (e.g. ../other_section)

If the common parent of both sections is the document (i.e. /), return an absolute path.

get_repository()
Returns the repository responsible for this Section, which might not be the *repository* attribute, but may be inherited from a parent Section / the Document.

get_section_by_path(*path*)
Find a Section through a path like "../name1/name2"

Parameters **path** (*str*) – path like "../name1/name2"

get_terminology_equivalent()
Returns the equivalent object in an terminology (should there be one defined) or None

id
The uuid for the section.

include
The same as *odml.section.BaseSection.link*, except that include specifies an arbitrary url instead of a local path within the same document.

insert(*position*, *obj*)
Insert a Section or a Property at the respective child-list position. A ValueError will be raised, if a Section or a Property with the same name already exists in the respective child-list.

Parameters

- **position** – index at which the object should be inserted.
- **obj** – Section or Property object.

is_merged
Returns True if the section is merged with another one (e.g. through *odml.section.BaseSection.link* or *odml.section.BaseSection.include*) The merged object can be accessed through the *_merged* attribute.

iterproperties(*max_depth=None*, *filter_func=<function Sectionable.<lambda>>*)
Iterate each related property (recursively)

Example: return all children properties which name contains "foo" >>> filter_func = lambda x: getattr(x, 'name').find("foo") > -1 >>> sec_or_doc.iterproperties(filter_func=filter_func)

Parameters

- **max_depth** (*bool*) – iterate all properties recursively if None, only to a certain level otherwise
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True

itersections(*recursive=True*, *yield_self=False*, *filter_func=<function Sectionable.<lambda>>*, *max_depth=None*)
Iterate each child section

Example: return all subsections which name contains "foo" >>> filter_func = lambda x: getattr(x, 'name').find("foo") > -1 >>> sec_or_doc.itersections(filter_func=filter_func)

Parameters

- **recursive** (*bool*) – iterate all child sections recursively (deprecated)
- **yield_self** (*bool*) – includes itself in the iteration
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True
- **max_depth** – number of layers in the document tree to include in the search.

itervalues(*max_depth=None*, *filter_func=<function Sectionable.<lambda>>*)
Iterate each related value (recursively)

 # Example: return all children values which string converted version has "foo"

```
>>> filter_func = lambda x: str(x).find("foo") > -1
>>> sec_or_doc.itervalues(filter_func=filter_func)
```

Parameters

- **max_depth** (*bool*) – iterate all properties recursively if None, only to a certain level otherwise
- **filter_func** (*function*) – accepts a function that will be applied to each iterable. Yields iterable if function returns True

link

A softlink, i.e. a path within the document. When the merge()-method is called, the link will be resolved creating according copies of the section referenced by the link attribute. When the unmerge() method is called (happens when running clean()) the link is unresolved, i.e. all properties and sections that are completely equivalent to the merged object will be removed. (They will be restored accordingly when calling merge()). When changing the *link* attribute, the previously merged section is unmerged, and the new reference will be immediately resolved. To avoid this side-effect, directly change the *_link* attribute.

merge(*section=None*, *strict=True*)

Merges this section with another *section*. See also: *odml.section.BaseSection.link* If section is none, sets the link/include attribute (if _link or _include are set), causing the section to be automatically merged to the referenced section.

Parameters

- **section** – an odML Section. If section is None, *link* or *include* will be resolved instead.
- **strict** – Bool value to indicate whether the attributes of affected child Properties except their ids and values have to be identical to be merged. Default is True.

merge_check(*source_section*, *strict=True*)

Recursively checks whether a source Section and all its children can be merged with self and all its children as destination and raises a ValueError if any of the Section attributes definition and reference differ in source and destination.

Parameters

- **source_section** – an odML Section.
- **strict** – If True, definition and reference attributes of any merged Sections as well as most attributes of merged Properties on the same tree level in source and destination have to be identical.

name

The name of the Section.

new_id(*oid=None*)

new_id sets the id of the current object to a RFC 4122 compliant UUID. If an id was provided, it is assigned if it is RFC 4122 UUID format compliant. If no id was provided, a new UUID is generated and assigned.

Parameters **oid** – UUID string as specified in RFC 4122.

oid

The uuid for the Section. Required for entity creation and comparison, saving and loading.

parent

The parent Section, Document or None.

pprint(*indent=2*, *max_depth=1*, *max_length=80*, *current_depth=0*)

Pretty prints Section-Property trees for nicer visualization.

Parameters

- **indent** – number of leading spaces for every child Section or Property.
- **max_depth** – number of maximum child section layers to traverse and print.
- **max_length** – maximum number of characters printed in one line.
- **current_depth** – number of hierarchical levels printed from the starting Section.

prop_cardinality
The Property cardinality of a Section. It defines how many Properties are minimally required and how many Properties should be maximally stored. Use the 'set_properties_cardinality' method to set.

properties
The list of all properties contained in this Section,

props
The list of all properties contained in this Section; NIXpy format style alias for 'properties'.

reference
A reference (e.g. an URL) to an external definition of the Section. :returns: The reference of the Section.

remove(*obj*)
Remove a Section or a Property from the respective child-lists of the current Section and sets the parent attribute of the handed in object to None. Raises a ValueError if the object is not a Section or a Property or if the object is not contained in the child-lists.

Parameters **obj** – Section or Property object.

reorder(*new_index*)
Move this object in its parent child-list to the position *new_index*.

Returns The old index at which the object was found.

repository
A URL to a terminology.

sec_cardinality
The Section cardinality of a Section. It defines how many Sections are minimally required and how many Sections should be maximally stored. Use the 'set_sections_cardinality' method to set.

sections
The list of all child-sections of this Section.

set_properties_cardinality(*min_val=None*, *max_val=None*)
Sets the Properties cardinality of a Section.

Parameters

- **min_val** – Required minimal number of values elements. None denotes no restrictions on properties elements minimum. Default is None.
- **max_val** – Allowed maximal number of values elements. None denotes no restrictions on properties elements maximum. Default is None.

set_sections_cardinality(*min_val=None*, *max_val=None*)
Sets the Sections cardinality of a Section.

Parameters

- **min_val** – Required minimal number of values elements. None denotes no restrictions on sections elements minimum. Default is None.

- **max_val** – Allowed maximal number of values elements. None denotes no restrictions on sections elements maximum. Default is None.

type = None

unmerge(*section*)
Clean up a merged section by removing objects that are totally equal to the linked object

4.1.3 Property

class odml.property.**BaseProperty**(*name=None*, *values=None*, *parent=None*, *unit=None*, *uncertainty=None*, *reference=None*, *definition=None*, *dependency=None*, *dependency_value=None*, *dtype=None*, *value_origin=None*, *oid=None*, *val_cardinality=None*, *value=None*)
An odML Property.

If a value without an explicitly stated dtype has been provided, dtype will be inferred from the value.

Example: >>> p = Property("property1", "a string") >>> p.dtype >>> str >>> p = Property("property1", 2) >>> p.dtype >>> int >>> p = Property("prop", [2, 3, 4]) >>> p.dtype >>> int

Parameters

- **name** – The name of the Property.
- **values** – Some data value, it can be a single value or a list of homogeneous values.
- **parent** – the parent object of the new Property. If the object is not an odml.Section a ValueError is raised.
- **unit** – The unit of the stored data.
- **uncertainty** – The uncertainty (e.g. the standard deviation) associated with a measure value.
- **reference** – A reference (e.g. an URL) to an external definition of the value.
- **definition** – The definition of the Property.
- **dependency** – Another Property this Property depends on.
- **dependency_value** – Dependency on a certain value.
- **dtype** – The data type of the values stored in the property, if dtype is not given, the type is deduced from the values. Check odml.DType for supported data types.
- **value_origin** – Reference where the value originated from e.g. a file name.
- **oid** – object id, UUID string as specified in RFC 4122. If no id is provided, an id will be generated and assigned. An id has to be unique within an odML Document.
- **val_cardinality** – Value cardinality defines how many values are allowed for this Property. By default unlimited values can be set. A required number of values can be set by assigning a tuple of the format "(min, max)".
- **value** – Legacy code to the 'values' attribute. If 'values' is provided, any data provided via 'value' will be ignored.

append(*obj*, *strict=True*)
Append a single value to the list of stored values. Method will raise a ValueError if the passed value cannot be converted to the current dtype.

Parameters

- **obj** – the additional value.
- **strict** – a Bool that controls whether dtypes must match. Default is True.

clean()
Stub that doesn't do anything for this class.

clone(*keep_id=False*)
Clone this property to copy it independently to another document. By default the id of the cloned object will be set to a different uuid.

Parameters **keep_id** – If this attribute is set to True, the uuid of the object will remain unchanged.

Returns The cloned property

definition

Returns the definition of the Property

dependency
Another Property this Property depends on. :returns: the dependency of the Property.

dependency_value
Dependency on a certain value in a dependency Property. :returns: the required value to be found in a dependency Property.

document
Returns the Document object in which this object is contained.

dtype
The data type of the value. Check odml.DType for supported data types.

export_leaf()
Export the path including all direct parents from this Property to the root of the document. Section properties are included, Subsections are not included.

Returns Cloned odml tree to the root of the current document.

extend(*obj*, *strict=True*)
Extend the list of values stored in this property by the passed values. Method will raise a ValueError, if values cannot be converted to the current dtype. One can also pass another Property to append all values stored in that one. In this case units must match!

Parameters

- **obj** – single value, list of values or a Property.
- **strict** – a Bool that controls whether dtypes must match. Default is True.

format()
Returns the format class of the current object.

get_merged_equivalent()
Return the merged object (i.e. if the parent section is linked to another one, return the corresponding property of the linked section) or None.

get_path()
Return the absolute path to this object.

get_terminology_equivalent()
Returns the equivalent object in an terminology (should there be one defined) or None

id
The uuid of the Property.

insert(*index*, *obj*, *strict=True*)
Insert a single value to the list of stored values. Method will raise a ValueError if the passed value cannot be converted to the current dtype.

Parameters

- **obj** – the additional value.
- **index** – position of the new value
- **strict** – a Bool that controls whether dtypes must match. Default is True.

merge(*other*, *strict=True*)
Merges the Property 'other' into self, if possible. Information will be synchronized. By default the method will raise a ValueError when the information in this property and the passed property are in conflict.

Parameters

- **other** – an odML Property.
- **strict** – Bool value to indicate whether types should be implicitly converted even when information may be lost. Default is True, i.e. no conversion, and a ValueError will be raised if types or other attributes do not match. If a conflict arises with strict=False, the attribute value of self will be kept, while the attribute value of other will be lost.

merge_check(*source*, *strict=True*)
Checks whether a source Property can be merged with self as destination and raises a ValueError if the values of source and destination are not compatible. With parameter *strict=True* a ValueError is also raised, if any of the attributes unit, definition, uncertainty, reference or value_origin and dtype differ in source and destination.

Parameters

- **source** – an odML Property.
- **strict** – If True, the attributes dtype, unit, uncertainty, definition, reference and value_origin of source and destination must be identical.

name
The name of the Property.

new_id(*oid=None*)
new_id sets the object id of the current object to an RFC 4122 compliant UUID. If an id was provided, it is assigned if it is RFC 4122 UUID format compliant. If no id was provided, a new UUID is generated and assigned.

Parameters **oid** – UUID string as specified in RFC 4122.

oid
The uuid of the Property. Required for entity creation and comparison, saving and loading.

parent
The Section containing this Property.

pprint(*indent=2*, *max_length=80*, *current_depth=-1*)
Pretty print method to visualize Properties and Section-Property trees.

Parameters

- **indent** – number of leading spaces for every child Property.
- **max_length** – maximum number of characters printed in one line.
- **current_depth** – number of hierarchical levels printed from the starting Section.

reference
A reference (e.g. an URL) to an external definition of the value. :returns: the reference of the Property.

remove(*value*)
Remove a value from this property. Only the first encountered occurrence of the passed in value is removed from the properties list of values.

reorder(*new_index*)
Move this object in its parent child-list to the position *new_index*.

Returns The old index at which the object was found.

set_values_cardinality(*min_val=None*, *max_val=None*)
Sets the values cardinality of a Property.

Parameters

- **min_val** – Required minimal number of values elements. None denotes no restrictions on values elements minimum. Default is None.
- **max_val** – Allowed maximal number of values elements. None denotes no restrictions on values elements maximum. Default is None.

uncertainty
The uncertainty (e.g. the standard deviation) associated with the values of the Property. :returns: the uncertainty of the Property.

unit
The unit associated with the values of the Property. :returns: the unit of the Property.

unmerge(*other*)
Stub that doesn't do anything for this class.

val_cardinality
The value cardinality of a Property. It defines how many values are minimally required and how many values should be maximally stored. Use the 'set_values_cardinality' method to set.

value
Deprecated alias of 'values'. Will be removed with the next minor release.

value_origin
Reference where the value originated from e.g. a file name. :returns: the value_origin of the Property.

value_str(*index=0*)
Used to access typed data of the value at a specific index position as a string.

values
Returns the value(s) stored in this property. Method always returns a list that is a copy (!) of the stored value. Changing this list will NOT change the property. For manipulation of the stored values use the append, extend, and direct access methods (using brackets).

For example: >>> p = odml.Property("prop", values=[1, 2, 3]) >>> print(p.values) [1, 2, 3] >>> p.values.append(4) >>> print(p.values) [1, 2, 3]

Individual values can be accessed and manipulated like this: >>> print(p[0]) [1] >>> p[0] = 4 >>> print(p[0]) [4]

The values can be iterated e.g. with a loop: >>> for v in p.values: >>> print(v) 4 2 3

4.2 odML-Support Classes

These classes are

4.2.1 Validation

class odml.validation.**Validation**(*obj*, *validate=True*, *reset=False*)

Validation provides a set of default validations that can used to validate odml objects. Custom validations can be added via the 'register_handler' method.

Parameters **obj** – odml object the validation will be applied to.

error(*validation_error*)

Registers an error found during the validation process.

register_custom_handler(*klass*, *handler*)

Adds a validation handler for an odml class. The handler is called in the validation process for each corresponding object. The *handler* is assumed to be a generator function yielding all ValidationErrors it finds.

Section handlers are only called for sections and not for the document node. If both are required, the handler needs to be registered twice.

Parameters

- **klass** – string corresponding to an odml class. Valid strings are 'odML', 'section' and 'property'.
- **handler** – validation function applied to the odml class.

static register_handler(*klass*, *handler*)

Adds a validation handler for an odml class. The handler is called in the validation process for each corresponding object. The *handler* is assumed to be a generator function yielding all ValidationErrors it finds.

Section handlers are only called for sections and not for the document node. If both are required, the handler needs to be registered twice.

Parameters

- **klass** – string corresponding to an odml class. Valid strings are 'odML', 'section' and 'property'.
- **handler** – validation function applied to the odml class.

report()

Validates the registered object and returns a results report.

run_validation()

Runs a clean new validation on the registered Validation object.

validate(*obj*)

Runs all registered handlers that are applicable to a provided odml class instance. Occurring validation errors will be collected in the Validation.error attribute.

Parameters **obj** – odml class instance.

4.2.2 IssueID

class odml.validation.**IssueID**
IDs identifying registered validation handlers.

custom_validation = 701

object_name_readable = 300

object_required_attributes = 101

property_dependency_check = 401

property_terminology_check = 400

property_unique_ids = 201

property_unique_name = 203

property_values_cardinality = 502

property_values_check = 402

property_values_string_check = 403

section_properties_cardinality = 500

section_repository_present = 600

section_sections_cardinality = 501

section_type_must_be_defined = 102

section_unique_ids = 200

section_unique_name_type = 202

unspecified = 1

4.2.3 ValidationError

class odml.validation.**ValidationError**(*obj*, *msg*, *rank='error'*, *validation_id=None*)
Represents an error found in the validation process.

The error is bound to an odML-object (*obj*) or a list of those and contains a message and a rank which may be one of: 'error', 'warning'.

is_error

Returns Boolean whether the current ValidationError has rank 'Error'.

is_warning

Returns Boolean whether the current ValidationError has rank 'Warning'.

path

Returns The absolute path to the odml object the ValidationError is bound to.

4.2.4 TemplateHandler

class odml.templates.**TemplateHandler**
TemplateHandler facilitates synchronous and deferred loading, caching, browsing and importing of full or partial odML templates.

browse(*url*)
Load, cache and pretty print an odML template XML file from a URL.

Parameters **url** – location of an odML template XML file.

Returns The odML document loaded from url.

clear() → None. Remove all items from D.

clone_section(*url*, *section_name*, *children=True*, *keep_id=False*)
Load a section by name from an odML template found at the provided URL and return a clone. By default it will return a clone with all child sections and properties as well as changed IDs for every entity. The named section has to be a root (direct) child of the referenced odML document.

Parameters

- **url** – location of an odML template XML file.
- **section_name** – Unique name of the requested Section.
- **children** – Boolean whether the child entities of a Section will be returned as well. Default is True.
- **keep_id** – Boolean whether all returned entities will keep the original ID or have a new one assigned. Default is False.

Returns The cloned odML section loaded from url.

copy() → a shallow copy of D

deferred_load(*url*)
Start a background thread to load an odML template from a URL.

Parameters **url** – location of an odML template XML file.

fromkeys()
Create a new dictionary with keys from iterable and values set to value.

get()
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

load(*url*)
Load and cache an odML template from a URL.

Parameters **url** – location of an odML template XML file.

Returns The odML document loaded from url.

loading = {}

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault()
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

4.2.5 Terminologies

class odml.terminology.**Terminologies**
Terminologies facilitates synchronous and deferred loading, caching, browsing and importing of full or partial odML terminologies.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

deferred_load(*url*)
Starts a background thread to load an odML XML file from a URL.

Parameters **url** – location of an odML XML file.

fromkeys()
Create a new dictionary with keys from iterable and values set to value.

get()
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

load(*url*)
Loads and caches an odML XML file from a URL.

Parameters **url** – location of an odML XML file.

Returns The odML document loaded from url.

loading = {}

pop(*k[, d]*) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

refresh(*url*)
Deletes and reloads all cached odML XML files given in the terminology file from a URL.

Parameters **url** – location of an odML XML file.

reload_cache = False

setdefault()
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

4.3 Data Types

Provides functionality for validation of the data-types specified for odML

class odml.dtypes.**DType**

The DType class enumerates all data types supported by odML.

boolean = 'boolean'

date = 'date'

datetime = 'datetime'

float = 'float'

int = 'int'

person = 'person'

string = 'string'

text = 'text'

time = 'time'

url = 'url'

odml.dtypes.**bool_get**(*string*)

Handles an input string value and escapes None and empty collections and provides the default boolean value in these cases. String values "true", "1", True, "t" are interpreted as boolean True, string values "false", "0", False, "f" are interpreted as boolean False. A ValueError is raised if the input value cannot be interpreted as boolean.

Parameters **string** – value to convert to boolean.

Returns boolean value.

odml.dtypes.**bool_set**(*string*)

Handles an input string value and escapes None and empty collections and provides the default boolean value in these cases. String values "true", "1", True, "t" are interpreted as boolean True, string values "false", "0", False, "f" are interpreted as boolean False. A ValueError is raised if the input value cannot be interpreted as boolean.

Parameters **string** – value to convert to boolean.

Returns boolean value.

odml.dtypes.**boolean_get**(*string*)

Handles an input string value and escapes None and empty collections and provides the default boolean value in these cases. String values "true", "1", True, "t" are interpreted as boolean True, string values "false", "0", False, "f" are interpreted as boolean False. A ValueError is raised if the input value cannot be interpreted as boolean.

Parameters **string** – value to convert to boolean.

Returns boolean value.

odml.dtypes.**boolean_set**(*string*)

Handles an input string value and escapes None and empty collections and provides the default boolean value in these cases. String values "true", "1", True, "t" are interpreted as boolean True, string values "false", "0", False, "f" are interpreted as boolean False. A ValueError is raised if the input value cannot be interpreted as boolean.

Parameters **string** – value to convert to boolean.

Returns boolean value.

odml.dtypes.**date_get**(*string*)
Checks an input string against the required date format and converts it to a date object with the default format. If *string* is empty the default value for date is returned.

Parameters **string** – string value to convert to date.

Returns date object.

odml.dtypes.**date_set**(*string*)
Checks an input string against the required date format and converts it to a date object with the default format. If *string* is empty the default value for date is returned.

Parameters **string** – string value to convert to date.

Returns date object.

odml.dtypes.**datetime_get**(*string*)
Checks an input string against the required datetime format and converts it to a datetime object with the default format. If *string* is empty the default value for datetime is returned.

Parameters **string** – string value to convert to datetime.

Returns datetime object.

odml.dtypes.**datetime_set**(*string*)
Checks an input string against the required datetime format and converts it to a datetime object with the default format. If *string* is empty the default value for datetime is returned.

Parameters **string** – string value to convert to datetime.

Returns datetime object.

odml.dtypes.**default_values**(*dtype*)
Returns the default value for a provided odml data type.

Parameters **dtype** – odml.DType or string corresponding to an odml data type.

Returns default value for an identified odml data type or empty string.

odml.dtypes.**float_get**(*string*)
Converts an input string to a float value. If *string* is empty the default value for float is returned.

Parameters **string** – string value to convert to int.

Returns Float value.

odml.dtypes.**get**(*string*, *dtype=None*)
Converts *string* to the corresponding *dtype*. The appropriate function is derived from the provided dtype. If no dtype is provided, the string conversion function is used by default.

Parameters

- **string** – string to be converted into an odml specific value.
- **dtype** – odml.DType or string corresponding to an odml data type. If provided it is used to identify the appropriate conversion function.

Returns value converted to the appropriate data type.

odml.dtypes.**infer_dtype**(*value*)
Tries to identify the odml data type for a provided value.

Parameters **value** – single value to infer the odml datatype from.

Returns The identified dtype name. If it cannot be identified, “string” is returned.

odml.dtypes.**int_get**(*string*)
Converts an input string to an integer value. If *string* is empty the default value for int is returned.

Parameters **string** – string value to convert to int.

Returns Integer value.

odml.dtypes.**set**(*value*, *dtype=None*)
Serializes a *value* of type *dtype* to a unicode string. The appropriate function is derived from the provided dtype.

Parameters

- **value** – odml specific value to be converted into a string.
- **dtype** – odml.DType or string corresponding to an odml data type. If provided it is used to identify the appropriate conversion function.

Returns value converted to an appropriately formatted string.

odml.dtypes.**str_get**(*string*)
Handles an input string value and escapes None and empty collections.

Parameters **string** – value to check for None value or empty collections.

Returns string value.

odml.dtypes.**str_set**(*string*)
Handles an input string value and escapes None and empty collections.

Parameters **string** – value to check for None value or empty collections.

Returns string value.

odml.dtypes.**string_get**(*string*)
Handles an input string value and escapes None and empty collections.

Parameters **string** – value to check for None value or empty collections.

Returns string value.

odml.dtypes.**string_set**(*string*)
Handles an input string value and escapes None and empty collections.

Parameters **string** – value to check for None value or empty collections.

Returns string value.

odml.dtypes.**time_get**(*string*)
Checks an input string against the required time format and converts it to a time object with the default format. If *string* is empty the default value for time is returned.

Parameters **string** – string value to convert to time.

Returns time object.

odml.dtypes.**time_set**(*string*)
Checks an input string against the required time format and converts it to a time object with the default format. If *string* is empty the default value for time is returned.

Parameters **string** – string value to convert to time.

Returns time object.

odml.dtypes.**tuple_get**(*string*, *count=None*)
Parses a tuple string like "(1024;768)" and return a list of strings with the individual tuple elements.

Parameters

- **string** – string to be parsed into odML style tuples.
- **count** – list of strings.

odml.dtypes.**tuple_set**(*value*)
Serializes odml style tuples to a string representation.

Parameters **value** – odml style tuple values.

Returns string.

odml.dtypes.**valid_type**(*dtype*)
Checks if *dtype* is a valid odML value data type.

Parameters **dtype** – odml.DType or string corresponding to an odml data type.

Returns Boolean.

4.4 Tools

Several tools are provided with the odml.tools package.

4.4.1 DictParser

The dict_parser module provides access to the DictWriter and DictReader class. Both handle the conversion of odML documents from and to Python dictionary objects.

class odml.tools.dict_parser.**DictReader**(*show_warnings=True*, *ignore_errors=False*)
A reader to parse dictionaries with odML content into an odml.Document.

error(*msg*)
If the parsers ignore_errors property is set to False, a ParserException will be raised. Otherwise the message is passed to the parsers warning method.

Parameters **msg** – Error message.

is_valid_attribute(*attr*, *fmt*)
Checks whether a provided attribute is valid for a provided odml class (Document, Section, Property).

Parameters

- **attr** – Python dictionary tag that will be checked if it is a valid attribute for the provided format class.
- **fmt** – required odml format class format.Document, format.Section or format.Property against which the attribute is checked.

Returns the attribute if the attribute is valid, None otherwise.

parse_properties(*props_list*)
Parses a list of Python dictionary objects containing odML properties to the odml.Property equivalents.

Parameters **props_list** – list of Python dictionary objects containing odML properties.

Returns list of parsed odml.Properties

parse_sections(*section_list*)
Parses a list of Python dictionary objects containing odML sections to the odml.Section equivalents including any subsections and properties.

Parameters **section_list** – list of Python dictionary objects containing odML sections.

Returns list of parsed odml.Sections

to_odml(*parsed_doc*)
Parses a Python dictionary object containing an odML document to an odml.Document. Will raise a ParserException if the Python dictionary does not contain a valid odML document. Also raises an InvalidVersionException if the odML document is of a previous odML format version.

Parameters **parsed_doc** – Python dictionary object containing an odML document.

Returns parsed odml.Document.

warn(*msg*, *label='Warning'*)
Adds a message to the parsers warnings property. If the parsers show_warnings property is set to True, an additional error message will be written to sys.stderr.

Parameters

- **msg** – Warning message.
- **label** – Defined message level, can be 'Error' or 'Warning'. Default is 'Warning'.

class odml.tools.dict_parser.**DictWriter**
A writer to parse an odml.Document to a Python dictionary object equivalent.

static get_properties(*props_list*)
Parses a list of odml.Properties to a Python dict object.

Parameters **props_list** – list of odml.Properties.

Returns list of parsed odml.Properties as a single Python dict object.

get_sections(*section_list*)
Parses a list of odml.Sections to a Python dict object. Will also parse any contained subsections and odml.Properties.

Parameters **section_list** – list of odml.Sections.

Returns list of parsed odml.Sections as a single Python dict object.

to_dict(*odml_document*)
Parses a full odml.Document to a Python dict object. Will also parse any contained odml.Sections, their subsections and odml.Properties.

Parameters **odml_document** – an odml.Document.

Returns parsed odml.Document as a Python dict object.

odml.tools.dict_parser.**parse_cardinality**(*vals*)
Parses an odml specific cardinality from an input value.

If the input content is valid, returns an appropriate tuple. Returns None if the input is empty or the content cannot be properly parsed.

Parameters **vals** – list or tuple

Returns None or 2-tuple

4.4.2 ODMLParser

A generic odML parsing module. It parses odML files and documents. All supported formats can be found in parser_utils.SUPPORTED_PARSERS.

class odml.tools.odmlparser.**JSONDateTimeSerializer**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Required to serialize datetime objects as string objects when working with JSON as output format.

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

encode(*o*)

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

item_separator = ', '

iterencode(*o*, *_one_shot=False*)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

key_separator = ': '

class odml.tools.odmlparser.**ODMLReader**(*parser='XML'*, *show_warnings=True*)

A reader to parse odML files or strings into odml documents, based on the given data exchange format, like XML, YAML, JSON or RDF.

Usage: yaml_odml_doc = ODMLReader(parser='YAML').from_file("odml_doc.yaml") json_odml_doc = ODMLReader(parser='JSON').from_file("odml_doc.json")

from_file(*file*, *doc_format=None*)

Loads an odML document from a file. The ODMLReader.parser specifies the input file format. If the input file is an RDF file, the specific RDF format has to be provided as well. Available RDF formats: 'xml', 'n3', 'turtle', 'nt', 'pretty-xml', 'trix', 'trig', 'nquads'.

Parameters

- **file** – file path to load an odML document from.
- **doc_format** – Required for RDF files only and provides the specific format of an RDF file.

Returns parsed odml.Document

from_string(*string*, *doc_format=None*)
Loads an odML document from a string object. The ODMLReader.parser specifies the input file format. If the input string contains an RDF format, the specific RDF format has to be provided as well. Available RDF formats: 'xml', 'n3', 'turtle', 'nt', 'pretty-xml', 'trix', 'trig', 'nquads'.

Parameters

- **string** – file path to load an odML document from.
- **doc_format** – Required for RDF files only and provides the specific format of an RDF file.

Returns parsed odml.Document

class odml.tools.odmlparser.**ODMLWriter**(*parser='XML'*)
A generic odML document writer for JSON, XML, YAML and RDF. The output format is specified on init.

Usage: xml_writer = ODMLWriter(parser='XML') xml_writer.write_file(odml_document, filepath)

to_string(*odml_document*, ***kwargs*)
Parses an odml.Document to a string in the file format defined in the ODMLWriter.parser property. Supported formats are JSON, YAML and RDF.

Parameters

- **odml_document** – odml.Document.
- **kwargs** – Writer backend keyword arguments e.g. for adding specific stylesheets for xml documents or specifying an RDF format. Refer to the documentation of the available parsers to check which arguments are supported.

Returns string containing the content of the odml.Document in the specified format.

write_file(*odml_document*, *filename*, ***kwargs*)
Writes an odml.Document to a file using the format defined in the ODMLWriter.parser property. Supported formats are JSON, XML, YAML and RDF. Will raise a ParserException if the odml.Document is not valid.

Parameters

- **odml_document** – odml.Document.
- **filename** – path and filename of the output file.
- **kwargs** – Writer backend keyword arguments. Refer to the documentation of the available parsers to check which arguments are supported.

odml.tools.odmlparser.**unicode_loader_constructor**(*_*, *node*)
Constructor for PyYAML to load unicode characters

odml.tools.odmlparser.**yaml_time_serializer**(*dumper*, *data*)
This function is required to serialize datetime.time as string objects when working with YAML as output format.

4.4.3 RDFConverter

The RDF converter module provides conversion of odML documents to RDF and the conversion of odML flavored RDF to odML documents.

class odml.tools.rdf_converter.**RDFReader**(*filename=None*, *doc_format=None*)
A reader to parse odML RDF files or strings into odML documents.

Usage: file = RDFReader().from_file("/path_to_input_rdf", "rdf_format") file = RDFReader().from_string("rdf file as string", "rdf_format") RDFReader().write_file("/input_path", "rdf_format", "/output_path")

from_file(*filename*, *doc_format*)
from_file loads an odML RDF file and converts all odML documents from this RDF graph into individual odML documents.

Parameters

- **filename** – Path of the input odML RDF file.
- **doc_format** – RDF format of the input odML RDF file.

Returns list of converted odML documents

from_string(*file*, *doc_format*)
from_string loads an odML RDF file or string object and converts all odML documents from this RDF graph into individual odML documents.

Parameters

- **file** – Path of the input odML RDF file or an RDF graph string object.
- **doc_format** – RDF format of the input odML RDF graph.

Returns list of converted odML documents

parse_document(*doc_uri*)
parse_document parses an odML RDF Document node into an odML Document.

Parameters **doc_uri** – RDF URI of an odML Document node within an RDF graph.

Returns dict containing an odML Document

parse_property(*prop_uri*)
parse_property parses an odML RDF Property node into an odML Property.

Parameters **prop_uri** – RDF URI of an odML Property node within an RDF graph.

Returns dict containing an odML Property

parse_section(*sec_uri*)
parse_section parses an odML RDF Section node into an odML Section.

Parameters **sec_uri** – RDF URI of an odML Section node within an RDF graph.

Returns dict containing an odML Section

to_odml()
to_odml converts all odML documents from a common RDF graph into individual odML documents.

Returns list of converted odML documents

class odml.tools.rdf_converter.**RDFWriter**(*odml_documents*, *rdf_subclassing=True*, *custom_subclasses=None*)
A writer to parse odML files into RDF documents.

Use the 'rdf_subclassing' flag to disable default usage of Section type conversion to RDF Subclasses. Provide a custom Section type to RDF Subclass Name mapping dictionary via the 'custom_subclasses' attribute to add custom or overwrite default RDF Subclass mappings.

Usage: RDFWriter(odml_docs).get_rdf_str('turtle') RDFWriter(odml_docs).write_file("/output_path", "rdf_format")

RDFWriter(odml_docs, rdf_subclassing=False).write_file("path", "rdf_format") RDFWriter(odml_docs, custom_subclasses=custom_dict).write_file("path", "rdf_format")

convert_to_rdf()
convert_to_rdf converts all odML documents to RDF, connects them via a common "Hub" RDF node and returns the created RDF graph.

Returns An RDF graph.

get_rdf_str(*rdf_format='turtle'*)
Convert the current odML content of the parser to a common RDF graph and return the graph as a string object in the specified RDF format.

Parameters **rdf_format** – RDF output format. Default format is 'turtle'. Available formats: 'xml', 'n3', 'turtle', 'nt', 'pretty-xml', 'trix', 'trig', 'nquads', 'json-ld'.

Returns string object

save_document(*doc*, *curr_node=None*)
Add the current odML Document to the RDF graph and handle all child elements recursively.

Parameters

- **doc** – An odml Document that should be added to the RDF graph.
- **curr_node** – An RDF node that is used to append the current odml element to the Hub node of the current RDF graph.

save_odml_list(*parent_node*, *rdf_predicate*, *odml_list*)
save_odml_list adds all odml elements in a list to the current parent node and handles all child items via save_element.

Parameters

- **parent_node** – current parent node in the RDF graph.
- **rdf_predicate** – RDF predicate used to add all odml entities to the parent node.
- **odml_list** – list of odml entities.

save_odml_values(*parent_node*, *rdf_predicate*, *values*)
save_odml_values adds an RDF seq node to the parent RDF node and creates a value leaf node for every odml value.

Parameters

- **parent_node** – current parent node in the RDF graph.
- **rdf_predicate** – RDF predicate used to add the Seq node to the current parent node.
- **values** – list of odml values.

save_property(*prop*, *curr_node*)
Add the current odML Property to the RDF graph and handle all child elements.

Parameters

- **prop** – An odml Section that should be added to the RDF graph.
- **curr_node** – An RDF node that is used to append the current odml element to the current RDF graph.

save_repository_node(*parent_node*, *rdf_predicate*, *leaf_value*)
save_repository_node adds a node with a given repository url to the current graphs terminology node. If the current graph does not yet contain a terminology node, it creates one and attaches the current node to it.

Parameters

- **parent_node** – current parent node in the RDF graph.
- **rdf_predicate** – RDF predicate used to add the terminology to the parent node.
- **leaf_value** – Value that will be added to the RDF graph.

save_section(*sec*, *curr_node*)
Add the current odML Section to the RDF graph and handle all child elements recursively.

Parameters

- **sec** – An odml Section that should be added to the RDF graph.
- **curr_node** – An RDF node that is used to append the current odml element to the current RDF graph.

write_file(*filename*, *rdf_format='turtle'*)
Convert the current odML content of the parser to a common RDF graph and write the resulting graph to an output file using the provided RDF output format.

Parameters

- **filename** –
- **rdf_format** – RDF output format. Default format is 'turtle'. Available formats: 'xml', 'n3', 'turtle', 'nt', 'pretty-xml', 'trix', 'trig', 'nquads', 'json-ld'.

odml.tools.rdf_converter.**load_rdf_subclasses**()
load_rdf_subclasses loads odml section types to RDF Section subclass types mappings from a file and returns the mapping as a dictionary. Will return an empty dictionary, if the Subclasses file cannot be loaded.

Returns Dictionary of the form {'Section type': 'RDF class type'}

odml.tools.rdf_converter.**rdflib_version_major**()

4.4.4 XMLParser

The xmlparser module provides access to the XMLWriter and XMLReader classes. Both handle the conversion of odML documents from and to XML files and strings.

The parser can be invoked standalone: python -m odml.tools.xmlparser file.odml

class odml.tools.xmlparser.**XMLReader**(*ignore_errors=False*, *show_warnings=True*, *filename=None*)
A reader to parse XML files or strings into odML data structures.

Usage:

```
>>> doc = XMLReader().from_file("file.odml")
```

check_mandatory_arguments(*data*, *arg_class*, *tag_name*, *node*)
Checks if a passed attribute is required for a specific odML class. If the attribute is required and not present in the data, the parsers error method is called.

Parameters

- **data** – list of mandatory arguments.
- **arg_class** – odML class corresponding to the content of the parent node.
- **tag_name** – name of the current XML node.
- **node** – XML node.

error(*msg*, *elem*)
If the parsers ignore_errors property is set to False, a ParserException will be raised. Otherwise the message is passed to the parsers warning method.

Parameters

- **msg** – Error message.
- **elem** – XML node corresponding to the error.

from_file(*xml_file*)
Parses the datastream from a file like object and return an odML data structure. If the file cannot be parsed, a ParserException is raised.

Parameters **xml_file** – file path to an XML input file or file like object.

Returns a parsed odml.Document.

from_string(*string*)
Parses an XML string and return an odML data structure. If the string cannot be parsed, a ParserException is raised.

Parameters **string** – XML string.

Returns a parsed odml.Document.

is_valid_argument(*tag_name*, *arg_class*, *parent_node*, *child=None*)
Checks if an argument is valid in the scope of a specific odML class. If the attribute is not valid, the parsers error method is called.

Parameters

- **tag_name** – string containing the name of the current XML node.
- **arg_class** – odML class corresponding to the content of the parent node.
- **parent_node** – the parent XML node.
- **child** – current XML node.

parse_element(*node*)
Identifies the odML object corresponding to the current XML node e.g. odml.Document, odml.Section or odml.Property. It will call the parsers method corresponding to the identified odML object e.g. parse_odML, parse_section, parse_property and return the results.

Parameters **node** – XML node.

parse_odML(*root*, *fmt*)
Parses the content of an XML node into an odml.Document including all subsections and odml.Properties.

Parameters

- **root** – XML node
- **fmt** – odML class corresponding to the content of the XML node.

Returns parsed odml.Document

parse_property(*root*, *fmt*)
Parses the content of an XML node into an odml.Property.

Parameters

- **root** – XML node
- **fmt** – odML class corresponding to the content of the XML node.

Returns parsed odml.Property

parse_section(*root*, *fmt*)
Parses the content of an XML node into an odml.Section including all subsections and odml.Properties.

Parameters

- **root** – XML node
- **fmt** – odML class corresponding to the content of the XML node.

Returns parsed odml.Section

parse_tag(*root*, *fmt*, *insert_children=True*)
Parse an odml node based on the format description *fmt* and instantiate the corresponding object. :param root: lxml.etree node containing an odML object or object tree. :param fmt: odML class corresponding to the content of the root node. :param insert_children: Bool value. When True, child elements of the root node

will be parsed to their odML equivalents and appended to the odML document. When False, child elements of the root node will be ignored.

warn(*msg*, *elem*)
Adds a message to the parsers warnings property. If the parsers show_warnings property is set to True, an additional error message will be written to sys.stderr.

Parameters

- **msg** – Warning message.
- **elem** – XML node corresponding to the warning.

class odml.tools.xmlparser.**XMLWriter**(*odml_document*)
Creates XML nodes storing the information of an odML Document.

header = '<?xml version="1.0" encoding="UTF-8"?>\n<?xml-stylesheet type="text/xsl" hre

static save_element(*curr_el*)
Returns an XML node for the odML object curr_el.

Parameters **curr_el** – odML object. Supported objects are odml.Document, odml.Section, odml.Property.

Returns parsed XML Node.

write_file(*filename*, *local_style=False*, *custom_template=None*)
write_file saves the XMLWriters odML document to an XML file.

Parameters

- **filename** – location and name where the file will be written to.
- **local_style** – Optional boolean. By default an odML XML document is saved with a default header containing an external stylesheet for viewing with a local or remote server. Set up for local viewing with the 'odmlview' command line script. When set to True, the saved XML file will contain a default XSL stylesheet to render the XML file in a web-browser. Note that Chrome requires the '.odml' extension to be registered as "application/xml" in the Mime-type database.
- **custom_template** – Optional string. Provide a custom XSL template to render the odML XML file in a web-browser. Please note, that the custom XSL template must not be a full XSL stylesheet, but has to start and end with the tag: '<xsl:template match="odML">[custom]</xsl:template>'.

odml.tools.xmlparser.**from_csv**(*value_string*)
Reads a string containing odML values and parses them into a list.

Parameters **value_string** – string of odML values.

Returns list of values.

odml.tools.xmlparser.**load**(*filename*)
Shortcut function for XMLReader().from_file(filename).

odml.tools.xmlparser.**parse_cardinality**(*val*)
Parses an odml specific cardinality from a string.

If the string content is valid, returns an appropriate tuple. Returns None if the string is empty or the content cannot be properly parsed.

Parameters **val** – string

Returns None or 2-tuple

odml.tools.xmlparser.**to_csv**(*val*)
Modifies odML values for serialization to strings and files.

Parameters **val** – odML value.

Returns modified value string.

4.5 Convenience converters

Several convenience converters are provided with the odml.tools.converters package.

4.5.1 FormatConverter

The FormatConverter can be used from the command line and within scripts to convert between the different odML formats and update previous file format versions to the most recent one.

A full list of the available odML output formats is available via the CONVERSION_FORMATS constant.

Command line usage: python -m <in_dir> <odml_out_format> [-out <out_dir>] [-r]

Examples: 1) >> python -m odml.tools.converters.format_converter <in_dir> v1_1 -out <out_dir> -r

Convert files from the path <in_dir> to .xml odml version 1.1, writes them to <out_dir> including subdirectories and its files from the input path.

2) >> python -m odml.tools.converters.format_converter <in_dir> odml

Converts files from path <in_dir> to .odml and writes them to <in_dir_odml> not including subdirectories.

class odml.tools.converters.format_converter.**FormatConverter**
Class for converting between the different odML file formats.

classmethod convert(*args=None*)
Enables usage of the argparse for calling convert_dir(...)

Usage: python -m <in_dir> <odml_out_format> [-out <out_dir>] [-r]

Examples:

1) >> python -m odml.tools.converters.format_converter <in_dir> v1_1 -out <out_dir> -r

Convert files from the path <in_dir> to .xml odml version 1.1, writes them to <out_dir> including subdirectories and its files from the input path.

2) >> python -m odml.tools.converters.format_converter <in_dir> odml

Converts files from path <in_dir> to .odml and writes them to <in_dir_odml> not including subdirectories.

Parameters **args** – Command line arguments. See usage for details.

classmethod convert_dir(*input_dir*, *output_dir*, *parse_subdirs*, *res_format*)

Convert files from given input directory to the specified res_format.

Parameters

- **input_dir** – Path to input directory
- **output_dir** – Path for output directory. If None, new directory will be created on the same level as input
- **parse_subdirs** – If True enable converting files from subdirectories
- **res_format** – Format of output files. Possible choices: "v1_1" (version 1 to version 1.1 xml files)

  "odml" (version 1.1 .xml to .odml files) "turtle", "nt" etc. (version 1.1 to RDF files) (full list of RDF serializers in CONVERSION_FORMATS)

4.5.2 VersionConverter

This module provides the class VersionConverter to convert odML XML files from version 1.0 to 1.1.

class odml.tools.converters.version_converter.**VersionConverter**(*filename*)

Class for converting odML XML files from version 1.0 to 1.1.

convert(*backend='XML'*)

This method returns the content of the provided file object converted to odML version 1.1 as a string object which is directly consumable by the odml.tools.ODMLReader. Will raise an Exception, if the backend format is not supported.

Parameters **backend** – File format of the source file. 'JSON', 'YAML' and 'XML' are supported. Default backend is 'XML'.

:returns an odML v1.1 document as an XML string

write_to_file(*filename*, *backend='XML'*)

This method converts the content of the provided converter file object to odML version 1.1 and writes the results to *filename*.

Parameters

- **filename** – Output file.
- **backend** – Format of the source file, default is XML.

4.6 Command line scripts

Several cli convenience scripts are automatically installed and are available from the command line.

4.6.1 odML conversion script

odmlConvert

odmlConvert searches for odML files within a provided SEARCHDIR and converts them to the newest odML format version. Original files will never be overwritten. New files will be written either to a new directory at the current or a specified location.

Usage: odmlconvert [-r] [-o OUT] SEARCHDIR

Arguments: SEARCHDIR Directory to search for odML files.

Options:

-o OUT	Output directory. Must exist if specified. If not specified, output files will be written to the current directory.
-r	Search recursively. Directory structures will not be retained.

-h –help Show this screen. –version Show version.

odml.scripts.odml_convert.**dep_note**(*args=None*)
Print deprecation warning and call main function.

Parameters **args** – Command line arguments

odml.scripts.odml_convert.**main**(*args=None*)
Convenience script to automatically convert odML files within a directory (tree) to the newest file version. Check the cli help for details. :param args: Command line arguments

odml.scripts.odml_convert.**run_conversion**(*file_list*, *output_dir*, *report*, *source_format='XML'*)
Convert a list of odML files to the latest odML version. :param file_list: list of files to be converted. :param output_dir: Directory where odML files converted to

the latest odML version will be saved.

Parameters

- **report** – Reporting StringIO.
- **source_format** – Original file format of the odML source files. XML, JSON and YAML are supported, default is XML.

4.6.2 odML to RDF script

odmlToRDF

odmlToRDF searches for odML files within a provided SEARCHDIR and converts them to the newest odML format version and exports all found and resulting odML files to XML formatted RDF. Original files will never be overwritten. New files will be written either to a new directory at the current or a specified location.

Usage: odmltordf [-r] [-o OUT] SEARCHDIR

Arguments: SEARCHDIR Directory to search for odML files.

Options:

-o OUT	Output directory. Must exist if specified. If not specified, output files will be written to the current directory.
-r	Search recursively. Directory structures will not be retained.

-h –help Show this screen. –version Show version.

odml.scripts.odml_to_rdf.**main**(*args=None*)
Convenience script to automatically convert odML files within a directory (tree) to RDF. Check the cli help for details. :param args: Command line arguments

odml.scripts.odml_to_rdf.**run_conversion**(*file_list*, *output_dir*, *rdf_dir*, *report*, *source_format='XML'*)
Convert a list of odML files to the latest odML version if required and export all files to XML RDF files in a specified output directory. :param file_list: list of files to be exported to RDF. :param output_dir: Directory where odML files converted to

the latest odML version will be saved.

Parameters

- **rdf_dir** – Directory where exported RDF files will be saved.
- **report** – Reporting StringIO.
- **source_format** – Original file format of the odML source files. XML, JSON and YAML are supported, default is XML.

odml.scripts.odml_to_rdf.**run_rdf_export**(*odml_file*, *export_dir*)
Convert an odML file to an XML RDF file and export it to an export directory with the same name as the original file and a '.rdf' file ending. :param odml_file: odML file to be converted to RDF. :param export_dir:

4.6.3 odML view (browse odml files locally)

odmlview

odmlview sets up a minimal webserver to view odml files saved in the XML format via the webbrowser. After it is started, the webserver will open a new tab in the default webbrowser and display the content of the directory the server was started from. odML files can then be viewed from there. To properly render XML, an odML file may contain the element '<?xml-stylesheet type="text/xsl" href="odmlDocument.xsl"?>' where the 'odmlDocument.xsl' stylesheet should reside in the same directory as the odML file to be rendered. By using the '–fetch' flag the latest version of this stylesheet will be downloaded from 'templates.g-node.org' to the current directory when starting up the service.

Usage: odmlview [-p PORT] [–fetch]

Options:

-p PORT	Port the server will use. Default: 8000
--fetch	Fetch latest stylesheet from templates.g-node.org to current directory

-h –help Show this screen –version Show version

odml.scripts.odml_view.**download_file**(*repo*, *filename*)
download_file fetches 'filename' from url 'repo' and saves it in the current directory as file 'filename'.

odml.scripts.odml_view.**main**(*args=None*)

odml.scripts.odml_view.**run**(*port=8000*, *extensions=None*)
run starts a simple webserver on localhost serving the current directory. Once started, it will open a tab on the default webbrowser and will continue to serve until manually stopped.

Parameters

- **port** – server port
- **extensions** – dictionary containing additional file extension - mime type mappings the server should be aware of. e.g. {'.xml': 'application/xml'}

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y