Sample rst2pdf doc

version

COOP

janvier 22, 2020

Contents

Welcome to opentea's documentation!

This is the documentation for the *default* branch of OPENTEA.

Contents:

Introduction

Opentea is a package of open source python modules developped by CERFACS-Team COOP. It works on nested objects.

- otinker: a GUI engine in tkinter, creating forms upon a SCHEMA specification
- noob: a library of tools working on nested objects

Quick introduction

OpenTEA description

` .. image:: https://nitrox.cerfacs.fr/opentea/opentea/badges/develop/build.svg

:target: https://nitrox.cerfacs.fr/opentea/opentea/badges/develop/build.svg
:alt: build

status

<https://nitrox.cerfacs.fr/opentea/opentea/commits/develop>`_

` .. image:: https://nitrox.cerfacs.fr/opentea/opentea/badges/develop/coverage.svg

:target: https://nitrox.cerfacs.fr/opentea/opentea/badges/develop/coverage.svg
:alt: coverage

report

<https://nitrox.cerfacs.fr/opentea/opentea/commits/develop>`_

Welcome to the OpenTEA GUI Engine and nested objects handling!

The documentation is currently in nitrox pages, soon moving to readTheDocs.

Installation

Opentea is OpenSource (Cecill-B) available on PiPY.

```
pip install opentea
```

Basic Usage

OpenTEA is, at first a Graphival User Interface engine, based on the json-SCHEMA description.

Assume a nested information conforming to the following SCHEMA :

```
---
title: "Trivial form..."
type: object
properties:
  first_tab:
    type: object
    title: Only tab.
    process: custom_callback.py
    properties:
      first_block:
        type: object
```

```
        title: Custom Block
        properties:
          number_1:
            title: "Number 1"
            type: number
            default: 32.
          operand:
            title: "Operation"
            type: string
            default: "+"
            enum: ["+", "-", "*", "/"]
          number_2:
            title: "Number 2"
            type: number
            default: 10.
          result:
            title: "result"
            state: disabled
            type: string
            default: "-"
```

The openTEA GUI wil show as :

In this form, a callback can be added to each tab. The corresponding `custom_callback.py` script is :

```
"""Module for the first tab."""

from opentea.noob.noob import nob_get, nob_set
from opentea.process_utils import process_tab


def custom_fun(nob_in):
    """Update result."""
    nob_out = nob_in.copy()
    operation = nob_get(nob_in, "operand")
    nb1 = nob_get(nob_in, "number_1")
    nb2 = nob_get(nob_in, "number_2")
    if operation == "+":
        res = nb1 + nb2
    elif operation == "-":
        res = nb1 - nb2
    elif operation == "*":
        res = nb1 * nb2
    elif operation == "/":
        res = nb1 / nb2
    else:
        res = None
    nob_set(nob_out, res, "result")
    return nob_out

if __name__ == "__main__":
    process_tab(custom_fun)
```

Finally, the data recoded by the GUI is available as a YAML file, conforming to the SCHEMA Validation:

```
first_tab:
  first_block:
    number_1: 32.0
    number_2: 10.0
```

```
    operand: +
    result: 42.0
```

GUI Building

Simple Example

We start with the following simple example, step by step, on the SCHEMA specification

The basic structure of the GUI is a graph. The nodes of the graphs are spread over 3 levels, root, tabs and blocks.

```
- root
  - tab 1
    - block 1.1
    - block 1.2
 - tab 2
    - block 2.1
    - block 2.2
    - block 2.3
```

Root Node level, the window

A this level, we only create a SCHEMA object (`type:object`) which can store, as `properties` , one or several tabs

```
---
title: "All you can Eat..."
type: object
properties:
    first_tab:
        ...
    second_tab:
        ...
```

Second Node level, the tab

We define here again a SCHEMA object (`type:object`) which can store, as `properties` , one or several hlder objects called **blocks**.

```
...(root)
first_tab:
  type: object
  title: First tab.
  order: 20
  properties:
    first_block:
      ...
    second_block:
      ...
```

Third Node level, the block

We define here again a SCHEMA object (`type:object`) which can store, as `properties` , one or several hlder objects called **blocks**.

```
... (tab , or block)
    first_block:
      type: object
      title: Customer Info
      properties:
        name:
```

```
        ...
      age:
        ...
      membership:
        ...
```

You can nest more blocks under blocks if needed.

Leaf level , or Parameters

Parameters are defined still in acordance with the SCHEMA standard:

Entries

The most basic parameters are called Entries. Here are the most common types :

- `string` string types
- `integer`, `number`, numeric types
- `boolean`, boolean types

```
(block or tab)
   name:
       title: "Name"
       type: string
       default: "john doe"
   age:
       title: "Age"
       type: integer
       default: 42
   age:
       title: "Weight"
       type: number
       default: 13.2
   membership:
       title: "Membership"
       type: boolean
       default: False
```

Special blocks

Special blocks are structures allowing mor complexity in the nested object

eXclusive OR objects

The exclusive OR mean that the structure can be either one graph or another, *but nothing else*. This stems from the SCHEMA oneOf, which is much more permissive : one graph, or another *or a void graph*.

To achieve a proper validation with the SCHEMA standard, the XOR structure is the following:

```
... (block or tab)
        purchase:
          title: "select purchase"
          type: object
          oneOf:
          - type: object
            required: [takeaway]
            properties:
              takeaway:
                type: object
                properties:
                  ...
```

```
            - type: object
              required: [lobby]
              properties:
                lobby:
                  type: object
                  properties:
                    ...
```

Here the `oneOf` takes a list of options. Each option is an `object` with a `required` single property. :

```
...(oneOf)
              type: object
              required: [lobby]
              properties:
                lobby:
                  type: object
                  properties:
                      ...
```

Multiple objects

This structure is the SCHEMA array, using `required`properties:

```
... (block or tab)
  vegetables:
    title: Edible vegetable (Multiple example)
    type: array
    items:
      type: object
      required:
      - name
      - veggieLike
      properties:
        name:
          type: string
          description: The name of the vegetable.
          default: dummy_vegetable
        veggieLike:
          type: boolean
          description: Do I like this vegetable?
          default: False
```

Data output

The data is saved as a YAML serialized nested object. The data saved by the GUI “simple_example” in `./src/opentea/examples/simple/` is looking like this :

```
irst_tab:
  first_block:
    age: 42
    membership: false
    name: john doe
  second_block:
    purchase:
      takeaway:
        bag: false
second_tab:
  first_block:
    vegetables:
    - name: dummy_vegetable
      veggieLike: false
```

opentea package

OpenTEA scientific GUI library. Documentation is hosted at: http://cerfacs.fr/opentea

Subpackages

opentea.gui_browser package

Submodules

opentea.gui_browser.otbrowser module

opentea.gui_forms package

Submodules

opentea.gui_forms.constants module

Constants definitions.

exception opentea.gui_forms.constants.**GetException**
Bases: **Exception**
Define an exception on the widget getters.

exception opentea.gui_forms.constants.**SetException**
Bases: **Exception**
Define an exception on the widget setters.

class opentea.gui_forms.constants.**SwitchForm**(master=None, **kw)
Bases: **tkinter.ttk.Frame**
Overriden Frame class to mimick notebooks without tabs.

add(name, title=None)
Add a tab-like Frame.

sf_del(tab_name)
Destroy tab_id tab.

sf_raise(tab_name)
Forget current view and repack tab_name tab.

class opentea.gui_forms.constants.**TextConsole**(holder, content, height=None, width=None)
Bases: **object**
Text widget with search and auto -refresh capabilities.

highlight_pattern(*args)
Highlight the pattern.

update(*args)
Update the content

opentea.gui_forms.constants.**create_scrollable_canvas**(holder_frame)
Create a scollable canvas.

opentea.gui_forms.constants.**load_icons**()
Load icons.
Load all ./otinker_images/*_icon.gif as icons
load_icons : dictionnary of ImageTk objects

opentea.gui_forms.leaf_widgets module

Module for leaf widgets.

class opentea.gui_forms.leaf_widgets.**LeafWidget** (schema, root_frame, name)
Bases: **object**
Factory for OpenTea Widgets.

get_status ()
Return current attribute self.status.

class opentea.gui_forms.leaf_widgets.**OTBoolean** (schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT booleans.

get ()
Return python boolean.

set (value**)**
Set boolean to widget.

class opentea.gui_forms.leaf_widgets.**OTChoice** (schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT choices widget.

get ()
Return python string.

set (value**)**
Set choice to widget.

class opentea.gui_forms.leaf_widgets.**OTComment** (schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT Comment field.

get ()
Return data.

set (value**)**
Set content.

class opentea.gui_forms.leaf_widgets.**OTDescription** (schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT descriptin field.

get ()
Return data.

set (value**)**
Set content.

class opentea.gui_forms.leaf_widgets.**OTDocu** (schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OTinteger variable.

get ()
Void return.

set (value**)**
Set value to documentation content.

class opentea.gui_forms.leaf_widgets.**OTEmpty**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT widget for unimplemented types.

get()
Return data.

set(value)
Set content.

class opentea.gui_forms.leaf_widgets.**OTFileBrowser**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
OT file/folder browser widget.

get()
Return data.

set(value)
Set content.

class opentea.gui_forms.leaf_widgets.**OTInteger**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets._OTEntry**
OTinteger variable.

get()
Return python integer.

set(value)
Set integer to widget.

class opentea.gui_forms.leaf_widgets.**OTList**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets.LeafWidget**
Factory for OpenTea Lists.

additem()
Add an item at the end of the array.

delitem()
Delete item at the end of the array

get()
Return data.

memory_changed(event)
Trigger virtual event on memory change.

set(value)
Set content.

class opentea.gui_forms.leaf_widgets.**OTNumber**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets._OTEntry**
OTNumber floats.

get()
Return python integer.

set(value)
Set integer to widget.

class opentea.gui_forms.leaf_widgets.**OTString**(schema, root_frame, name)
Bases: **opentea.gui_forms.leaf_widgets._OTEntry**

OTinteger variable.

get ()
Return python integer.

set (value**)**
Set integer to widget.

opentea.gui_forms.node_widgets module

Module for containers widgets.

class opentea.gui_forms.node_widgets.**OTContainerWidget** (schema, root_frame, name, n_width=1)
Bases: **opentea.gui_forms.node_widgets.OTNodeWidget**
OT container widget.

class opentea.gui_forms.node_widgets.**OTMultipleItem** (multiple, name)
Bases: **opentea.gui_forms.node_widgets.OTContainerWidget**
OT multiple widget.

class opentea.gui_forms.node_widgets.**OTMultipleWidget** (schema, root_frame, name)
Bases: **object**
OT multiple widget.

add_item_on_cursel ()
Add an item in the multiple.
Item will be added before the current selection.

del_item_by_name (name**)**
Delete a Multiple item by its name.

del_item_on_cursel ()
Delete a Multiple item from tv selection.

get ()
Get the data of children widgets.
a list with the get result of childrens

get_status ()
Compute the minimal status in children.

index_of_item (name**)**
Find index of a Multiple item by its name.

refresh_view (event=None**)**
Refresh items values on tree view.

rename_callback (item_name**)**
Trigger renaming if dialog conditions are met.

rename_item (item_name**,** new_name**)**
Rename one element of the multiple.

set (list_**)**
Get the data of children widgets.
a list with the value of the childrens

class opentea.gui_forms.node_widgets.**OTNodeWidget** (schema)
Bases: **object**
Factory for OpenTea Widgets Containers.

get ()
Get the data of children widgets.
a dictionnary with the get result of childrens

get_status ()
Return the minimal status of children.

set (dict_**)**
Get the data of children widgets.
a dictionnary with the value of the childrens

class opentea.gui_forms.node_widgets.**OTTabWidget** (schema, root, name)
Bases: **opentea.gui_forms.node_widgets.OTNodeWidget**
OT Tab widget container.
Called for the 1st layer of nodes in the global schema

on_memory_change (event**)**
Check if the sender is child of this tab.process. set to unknown if so

on_memory_check (event**)**
Update content upon status of children.

process_button ()
Procees the main tab button.

update_tab_icon (icon_name**)**
Update the Tab icon upon status.

class opentea.gui_forms.node_widgets.**OTXorWidget** (schema, root_frame, name, n_width=1)
Bases: **object**
OT Or-exclusive / oneOf widget.

get ()
Get the data of children widgets.
a dictionnary with the get result of current children

get_status ()
Proxy to the get_status of the current child.

set (dict_**)**
Get the data of children widgets.
a dictionnary with the value of the childrens

update_xor_content (name_child**,** data_in=None**)**
Reconfigure XOR button.
name_child : sting, naming the child object data_in : dictionary used to pre-fill the data

xor_callback (name_child**)**
Event on XOR menu selection.

opentea.gui_forms.node_widgets.**redirect_string** (schema, root_frame, name)
Redirect to string widgets.
The schema attributes trigger which string widget will be in use.
schema : a schema object root_frame : a Tk object were the widget will be grafted name : name of the element none

opentea.gui_forms.node_widgets.**redirect_widgets** (schema, root_frame, name)
Redirect to widgets.
The schema attributes trigger which widget will be in use.
schema : a schema object root_frame : a Tk object were the widget will be grafted name : name of the element none

opentea.gui_forms.otinker module

Generate a Tk from upon a Gui schema.

A GUI schema is a JSON-Schema dictionnary, with tags require and existifs added to declare explicit cyclic depenencies

opentea.gui_forms.otinker.**main_otinker** (schema, calling_dir=None, start_mainloop=True)
Startup the gui generation.
schema : dictionary compatible with json-schema calling_dir : directory from which otinker was called test_only : only for testing
a tkinter GUI

opentea.gui_forms.root_widget module

Root widget.

class opentea.gui_forms.root_widget.**OTRoot** (schema, tksession, calling_dir, start_mainloop=True)
Bases: **opentea.gui_forms.node_widgets.OTNodeWidget**
OT root widget.

execute (script)
execute a script

opentea.gui_forms.wincanvas module

Opentea module for wincanvas.

class opentea.gui_forms.wincanvas.**WinCanvasImage** (schema, root_frame, name)
Bases: **object**
Class for Image handling in dynamic canvases

opentea.gui_forms.wincanvas.**redirect_canvas_items** (schema, root_frame, name)
Redirect to wincanvas widgets.
The schema attributes trigger which string widget will be in use.
schema : a schema object root_frame : a Tk object were the widget will be grafted name : name of the element none

opentea.noob package

Submodules

opentea.noob.asciigraph module

String represeantion of a nested object

opentea.noob.asciigraph.**asciigraph_rec** (nob, level)
Pretty printing of a nested object Inputs: ———- nob : The nested object level : The initial level to start with None

opentea.noob.asciigraph.**nob_asciigraph** (nob)
Pretty printing of a nested object Inputs: ———- nob : The nested object
None

opentea.noob.check_schema module

Test the structure of a schema.

The first one is for generic schema

The second is for the use of gui_forms

exception opentea.noob.check_schema.**NobSchemaError**

Bases: **Exception**
Error due to schema structure

opentea.noob.check_schema.**boolean_check_schema** (schema)
Same as check schema with a boolean output.

opentea.noob.check_schema.**nob_check_schema** (schema)
Check if a schema is valid agains opentea requirements
schema : a schema object

opentea.noob.check_schema.**read_serialized_data** (fname)
read any serialized data file

opentea.noob.check_schema.**rec_check_array** (schema, path)
Recursive check specific to arrays.

opentea.noob.check_schema.**rec_check_leafs** (schema, path)
Recursive check specific to leafs.

opentea.noob.check_schema.**rec_check_properties** (schema, path)
Recursive check specific to properties.

opentea.noob.check_schema.**rec_check_schema** (schema, path)
Recursive inference.
schema : a schema object
list_err : a list of errors encountered

opentea.noob.check_schema.**rec_check_xor** (schema, path)
Recursive check specific to exlusive or.

opentea.noob.inferdefault module

Infer a nested object from a schema.

opentea.noob.inferdefault.**avoid_string_duplication** (list_str)
Individualize repated string items.

opentea.noob.inferdefault.**infer_number** (schema)
Return default number if not provided by schema

opentea.noob.inferdefault.**nob_complete** (schema, update_data=None)
Infer a nested object from a schema.
schema : a schema object update_data : nested object, providing known

parts of the object to infer
nob_out : nested object

opentea.noob.inferdefault.**recursive_infer** (schema, path, update_data=None)
Recursive inference.
schema : a schema object update_data : nested object, providing known

parts of the object to infer
nob_out : nested object

opentea.noob.inferdefault.**recursive_infer_array** (schema, path, update_data=None)
Recursive inference specific to arrays.

opentea.noob.inferdefault.**recursive_infer_leafs** (schema, update_data=None)
Recursive inference specific to leafs.

opentea.noob.inferdefault.**recursive_infer_oneof** (schema, path, update_data=None)
Recursive inference specific to oneOfs.

opentea.noob.inferdefault.**recursive_infer_properties** (schema, path, update_data=None)
Recursive inference specific to properties.

opentea.noob.noob module

Nested object services.

A nested object is here:

-nested dicts -nested lists -a mix of nested lists and nested dicts

An address is a list of strings and/or integers giving a position in the nested object

Hereafter, the -address, complete or not- statement refer to an address with potentially missing elements.

EXAMPLE:@ for a dict such as : d[“a”][“b”][“c”][“d”][“e”][“f”] this the full address [[“a”,”b”,”c”,”d”,”e”]] can be found with either: nob_find(d, “a”,”b”,”c”,”d”,”e”) (full path) nob_find(d, “b”,”d”,”e”) (partial path) nob_find(d, “e”) (only one hint)

exception `opentea.noob.noob.`**`NobReferenceError`**
Bases: **`Exception`**
TO BE ADDED

`opentea.noob.noob.`**`nob_del`**(`obj_`, `*keys`, `verbose=False`)
Delete all matchinf addresses in the nested object. Not a deletion in place, only the output argument is cropped.
obj_ : nested object keys : address, complete or not
obj_ : nested object without the matching keys

`opentea.noob.noob.`**`nob_find`**(`obj_`, `*keys`)
Find all occurences matching a serie of keys in a nested object.
obj_ : nested object keys : address, complete or not
list of addresses matching the input address

`opentea.noob.noob.`**`nob_find_unique`**(`obj_`, `*keys`)
Find a unique occurences of a key in a nested object. Raise exceptions if problems
obj_ : nested object keys : address, complete or not
one single address matching the input address

`opentea.noob.noob.`**`nob_get`**(`obj_`, `*keys`, `failsafe=False`)
Access a nested object by keys.
obj_ : nested object keys : address, complete or not failsafe : what to do if the node is missing

- False : raise an exception
- **True :** *return None is missing*
  return the shortes path is several matches

if points to a leaf:

immutable, the value stored in the leaf

if points to a node:

mutable : the object (dict or list) found at this address

`opentea.noob.noob.`**`nob_get_only_child`**(`obj_`, `*keys`)
Return the only key of a single child dict.

`opentea.noob.noob.`**`nob_merge_agressive`**(`base_obj`, `obj_to_add`)
Merge two nested objects.
In case of conflict, the object to add is prevalent
base_obj : the initial object obj_to_add : the object to add
merged_obj : the merged dictionnaries

`opentea.noob.noob.`**`nob_node_exist`**(`obj_`, `*keys`)
Test if one node exist in a nested object
obj_ : nested object keys : address, complete or not
boolean

`opentea.noob.noob.`**`nob_pprint`**(`obj_`, `max_lvl=None`)
return a pretty print of a nested object. yaml.dump() in use for the display
obj_ : nested object max_lvel : optional : maximum nber of levels to show
out : string showing the nested_object structure

`opentea.noob.noob.`**`nob_set`**(`obj_`, `value`, `*keys`)
Assign a value to an object from a nested object.
obj_ : nested object keys : address, complete or not
change the object in argument (NOT STATELESS)

opentea.noob.noob.**str_address**(addr)
Return an address -key list- into a path-like string.

opentea.noob.schema module

Helper functions for schema handling

opentea.noob.schema.**clean_schema_addresses**(list_, udf_stages=None)
Clean a address from the addtitionnal layers of SCHEMA.
Used only when a SCHEMA address must be found in the data to validate

Parameters :

list_ : *a list of string*

address in a nested dict
udf_stages : a list of additionnal user defined stages (udf) Returns : ——— list

the same list without SCHEMA intermedaite stages

opentea.noob.validate_light module

Lightweight validate from jsonschema

exception opentea.noob.validate_light.**ValidationErrorShort**
Bases: **Exception**
Valodation error

opentea.noob.validate_light.**validate_base**(data, schema)
Validate in the default case .

opentea.noob.validate_light.**validate_in_oneof**(data, schema)
Validate in the case on an opentea oneOf schema.

opentea.noob.validate_light.**validate_light**(data, schema)
Schema validation procedure.
data : a nested dict to validate schema : the schema to validate against (jsonschema grammar)
THIS IS NOT A BOOLEAN Only exceptions are returned if any problem, else none.

opentea.noob.validation module

Module to operate a trplie layer of validation

exception opentea.noob.validation.**ErrorExistIf**
Bases: **Exception**
Errors on exist if elements.

exception opentea.noob.validation.**ErrorRequire**
Bases: **Exception**
Errors on require elements.

exception opentea.noob.validation.**OpenteaSchemaError**
Bases: **Exception**
Error in OptenTea schema structure

opentea.noob.validation.**clean_opentea_list_item**(item_in, existing_names)
Add # to list elements to avoid duplication.

opentea.noob.validation.**main_validate**(data, schema)
Main validation procedure.
data : a nested dict to validate schema : the schema to validate agains (jsonschema grammar)
Only exceptions are returned if any problem

opentea.noob.validation.**opentea_clean_data**(nobj)
Check if data is opentea proof

opentea.noob.validation.**opentea_resolve_existif**(data, schema)

Validate existif dependencies.
if an item existence depends on the value of one other item
data : a nested dict to validate schema : the schema to validate against (jsonschema grammar)
data_out : a nested dict to synchronize data with require updated

`opentea.noob.validation.`**`opentea_resolve_require`**(`data`, `schema`, `verbose=False`)
Validate require dependencies.
if children of an item depends of the value of one other item. -tgt- is the node to update -src- is the information used to update
data : a nested dict to synchronize schema : the schema to validate against (jsonschema grammar)
data_out : a nested dict to synchronize data with require updated

`opentea.noob.validation.`**`rec_validate_opentea_data`**(`nobj`)
Recursive validation for opeatea structured dict

`opentea.noob.validation.`**`rec_validate_schema`**(`schema`)
Validate if schema is compatible with opetea structure.

`opentea.noob.validation.`**`validate_array`**(`schema`)
Validate an opentea multiple structure.

`opentea.noob.validation.`**`validate_oneof`**(`schema`)
Validate an opentea multiple structure.

`opentea.noob.validation.`**`validate_opentea_schema`**(`schema`)
Check if schema is OpenTEA-proof.

- named items in arrays of dicts,
- required optin in xors
- existif and require defined

opentea.tools package

Submodules

opentea.tools.proxy_h5 module

Module container for class ProxyH5

class `opentea.tools.proxy_h5.`**`ProxyH5`**(`h5_filename`)
Bases: **`object`**
Class container for hdf5 file inspector
h5_filename : Path to hdf5 file

`get_field``(identifier)`
Get a value of a field given its identifier
identifier : the adress of the content to retrieve.

Possible options:

- String : key of the value to retrieve
  (e.g ‘content’)
- String : A posix-like full address (e.g ‘full/address/to/content’)
- List : A list of adress stages, complete or not (e.g [‘full’, ‘address’, ‘to’, ‘content’] or,
  [‘content’])
- A list of lists : a list holding address stages, as it is done for instance in h5py. (e.g [[‘full’], [‘address’], [‘to’], [‘content’]]])

field : array or value of the query field

`show``(style=None)`

Pretty print of the hdf5 content

style : *style of printing, possible options*

- yaml : for a yaml formatting
- json : for a yaml formatting
- None : for a default printing

`opentea.tools.proxy_h5.`**`test`** ()
example of usage

opentea.tools.schema2md module

Translate a schema nested object into Markdown table.

`opentea.tools.schema2md.`**`array2md`** (`nob`, `path`)
Print an array item into markdown

`opentea.tools.schema2md.`**`html_colored`** (`str_`, `color='black'`)
Represent a string with html color markup.

`opentea.tools.schema2md.`**`html_list`** (`list_`)
Represent a list with html markup.

`opentea.tools.schema2md.`**`md_table_header`** ()
The regular table header

`opentea.tools.schema2md.`**`schema2md`** (`schema`)
Convert schma type nest objet into Markdown

`opentea.tools.schema2md.`**`to_table_line`** (`nob`, `path`)
Reccursive cfn to create tables.

`opentea.tools.schema2md.`**`type_color`** (`type_str`)
Redirect according to types.

`opentea.tools.schema2md.`**`xor2md`** (`nob`, `path`)
Print a eXclusive Or item into markdown

opentea.tools.visit_h5 module

Visit h5py file

exception `opentea.tools.visit_h5.`**`H5LookupError`** (`message`)
Bases: **`Exception`**
Exception class for h5 lookup

`opentea.tools.visit_h5.`**`ascii2string`** (`ascii_list`)
Ascii to string conversion
ascii_list : a list of string to be converted
a string joining the list elements

`opentea.tools.visit_h5.`**`get_node_description`** (`node`)

Get number of elements in an array or

value of a single-valued node.

node : hdf5 node
a value with a Python format None if data is not a singlevalued quantity

`opentea.tools.visit_h5.`**`h5_datasets_names`** (`node`)
Get hdf5 node datasets names
node : hdf5 node
ds_names : a list of datasets names

`opentea.tools.visit_h5.`**`h5_node_to_dict`** (`node`)
Read hdf5 node values and structure into a dictionnary
node : hdf5 node

data_dict : a dictionnary holding the data

opentea.tools.visit_h5.**hdf5_query_field**(node, address)
Get the content of the adress
node : hdf5 node address : the adress of the content to retrieve. Possible options:

- String : key of the value to retrieve (e.g ‘content’)
- String : A posix-like full address (e.g ‘full/address/to/content’)
- List : A list of adress stages, complete or not (e.g [‘full’, ‘address’, ‘to’, ‘content’] or, [‘content’])
- A list of lists : a list holding address stages, as it is done for instance in h5py. (e.g [[‘full’], [‘address’], [‘to’], [‘content’]]])

content of the address

opentea.tools.visit_h5.**log_hdf_node**(node)
Build a dictionnary with the structure of a HDF5 node
node : hdf5 node
a dictionnary

opentea.tools.visit_h5.**pprint_dict**(dict_, style=None)
Pretty pring a dictionnary using yaml or json formatting

opentea.tools.visit_h5.**visit_h5**(h5_filename)
Show hdf5 file components
h5_filename : path to hdf5 file to inspect

Submodules

opentea.cli module

cli.py

Command line interface for tools in pyavbp

opentea.process_utils module

Utilities for opentea additionnal processing in tabs

opentea.process_utils.**process_tab**(func_to_call)
Execute the function of an external process.external.
func_to_call : see above for a typical function to be called by openTea GUIS

opentea.process_utils.**template_aditional_process**(nob_in)
Template of an additionnal process.
nob_in : nested object containing the initial data
nob_out : a nested object containing the altered data

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

U

V

W

X

Python Module Index

o