Opentea Documentation

Team COOP

Apr 16, 2020

Contents

1 Quick introduction **3**

2 OpenTEA **5**
- 2.1 Installation . . . 6
- 2.2 Basic Usage . . . 6

3 Command line **11**

4 Acknowledgments **13**

5 GUI Building **15**
- 5.1 Simple Example . . . 15
- 5.2 Special blocks . . . 27
- 5.3 Data output . . . 30
- 5.4 Style adjustments . . . 30

6 opentea package **33**
- 6.1 Subpackages . . . 33
- 6.2 Submodules . . . 34
- 6.3 opentea.cli module . . . 34
- 6.4 opentea.process_utils module . . . 34

7 Indices and tables **35**

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

Contents:

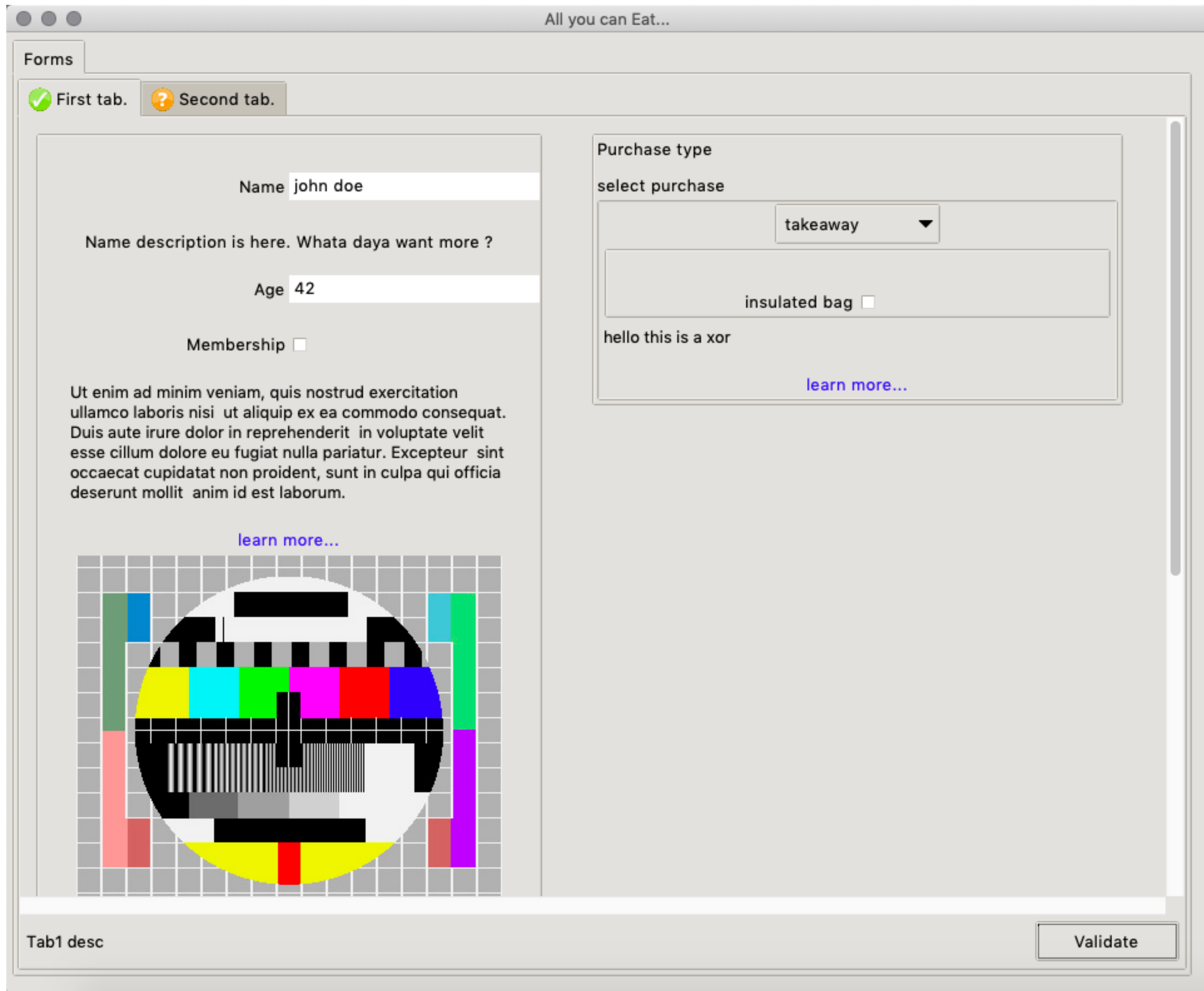
CHAPTER 1

Quick introduction

CHAPTER 2

OpenTEA

OpenTEA is a graphical user interface engine. It convert a set of degrees of freedom, expressed in SCHEMA, into graphical forms.



The documentation is currently available in ReadtheDocs

2.1 Installation

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

```
>pip install opentea
```

then test your installation with

```
>opentea3 test-gui trivial
```

2.2 Basic Usage

OpenTEA is a GUI engine, based on the json-SCHEMA description. For example, assume a nested information conforming to the following SCHEMA :

```yaml
---
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 will show as :

Trivial form...

Forms

Only tab.

Custom Block

Number 1 32.0

Operation + - * /

Number 2 10.0

result 42.0

Done in 0.177s, successfull Process

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.process_utils import process_tab

def custom_fun(nob):
    """Update the result."""

    operation = nob["first_tab"]["first_block"]["operand"]
    nb1 = nob["first_tab"]["first_block"]["number_1"]
    nb2 = nob["first_tab"]["first_block"]["number_2"]

    res = None
    if operation == "+":
        res = nb1 + nb2
```

(continues on next page)

(continued from previous page)

```
    elif operation == "-":
        res = nb1 - nb2
    elif operation == "*":
        res = nb1 * nb2
    elif operation == "/":
        res = nb1 / nb2
    else:
        res = None

    nob["first_tab"]["first_block"]["result"] = res
    return nob

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

Note that OpenTEA meomory is a classical nested object named here `nob`. The memory I/O can be done the usual Python way : `nob["first_tab"]["first_block"]["result"] = res`. *We however encourage the use our nested object helper , available on PyPI, which gives a faster -an still pythonic- access to the nested object. The name of the package is, unsurprisigly 'nob <https://pypi.org/project/nob/>'_.*

Finally, the data recorded 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
```

CHAPTER 3

Command line

A small CLI makes available small tools for developpers. Only two tools are present now. Call the CLI using `opentea3`:

```
Usage: opentea3 [OPTIONS] COMMAND [ARGS]...

  ---------------    O P E N T E A  III  --------------------

  You are now using the Command line interface of Opentea 3, a Python3
  Tkinter GUI engine based on SCHEMA specifications, created at CERFACS
  (https://cerfacs.fr).

  This is a python package currently installed in your python environement.
  See the full documentation at : https://opentea.readthedocs.io/en/latest/.

Options:
  --help  Show this message and exit.

Commands:
  test-gui     Examples of OpenTEA GUIs
  test-schema  Test if a yaml SCHEMA_FILE is valid for an opentea GUI.
```

CHAPTER 4

Acknowledgments

This work was funded, among many sources, by the CoE Excellerat and the National project ICARUS. Many thanks to the people from SAFRAN group for their feedback.

CHAPTER 5

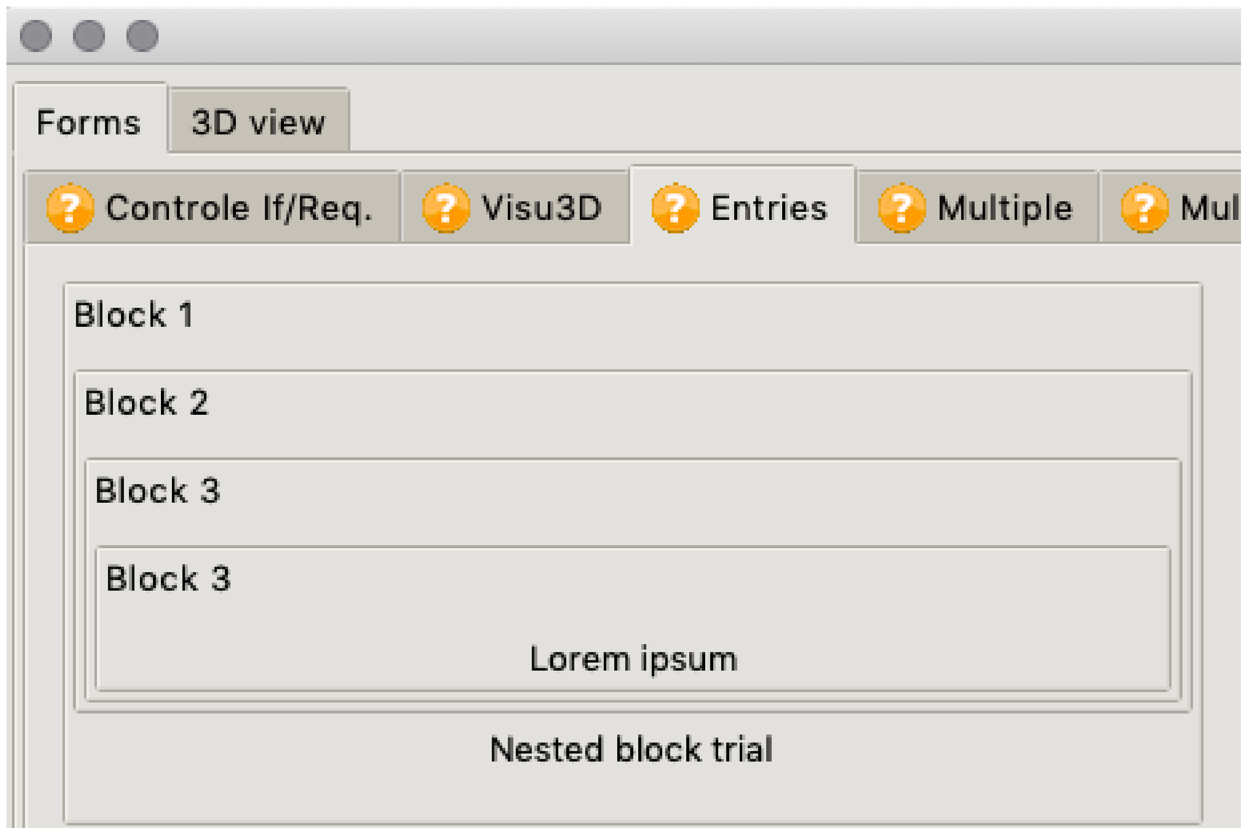
GUI Building

5.1 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 is the Major "Forms" tab. Tabs are shown in the second line with oragne icons. Then four nested blocks are shown

Root stands for the top-level of the whole form.

Tabs are the nodes grouping parameters of a similar family, such as "numerics", "boundary conditions", "meshes". One should design the interface with the general idea of a Left to Right filling of the forms. Therefore there is a tacit order of resolutions between tabs. **The last tab** must be reserved for the final execution of the action.

Blocks are grouping visually parameters in columns. The packing algorithm is filling the GUI screen in columns. The width of the general window controls the number of columns packed.

Please, stick this 3-level structure for your GUIs. The packing is optimized for this usage. Less than 3 levels will probably fail at startup. More than 3 levels (blocks in blocks) will limit the fluidity of the repacking when readjusting the window.

5.1.1 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:
        ...
```

Root nodes, in yaml are litterally sticking to left margin of your YAML file.

5.1.2 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.
     process: custom_callback.py
     description: "tab description"
     order: 20
     properties:
       first_block:
         ...
       second_block:
         ...
```

A quick tip : Tabs nodes, in yaml are found after 1 indentation (2 spaces, providing you use the standard 2 spaces chars indentation). The content is found after 2 indentations / 4 chars.

As for now, tabs a displayed in the order of the schema. The Tab level is interpreting the following additionnal attributes:

- `description`. This attribute is the SCHEMA official attibute. It takes a string. The string will be shown in the GUI at thebottom left of the Tab.
- `process`. This attribute is special for opentea. The string refer to the name of a python script to be called when pressing the "Process" button (a.k.a callback). See section Tabs callbacks for further information…
- `order` This attribute is special for opentea, but have no effect for the moment. It will force the order of Tabs when the functunality "Hide this tab" will be implemented.

Tab DOES NOT support the attributes `image`or `documentation`. It however supports the attribute `description`, providing the string is sufficently short. Indeed, there is no huge room for display at the bottom of the tabs.

Tabs callbacks

Without callbacks, OpenTEA is simply some forms allowing you to fill a nested object (a YAML file on the disc) according to a SCHEMA specification. Tabs callbacks are the way to add interactivity to your forms. The data passed from the GUI to the callback is the GUI memory itself, dumped as the file `dataset_from_gui.yml`. The data passed from the callback to the GUI is dumped as the file `dataset_to_gui.yml`

In the end, the signature of the function is `callback(nob_in) > nob_out`, with nob a python nested object (e.g. dicts of dicts of lists of anything you can serialize in YAML…). YOu can refer to PyYAML documentation for practical examples.

A typical callback is the following:

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

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

```

(continues on next page)

(continued from previous page)

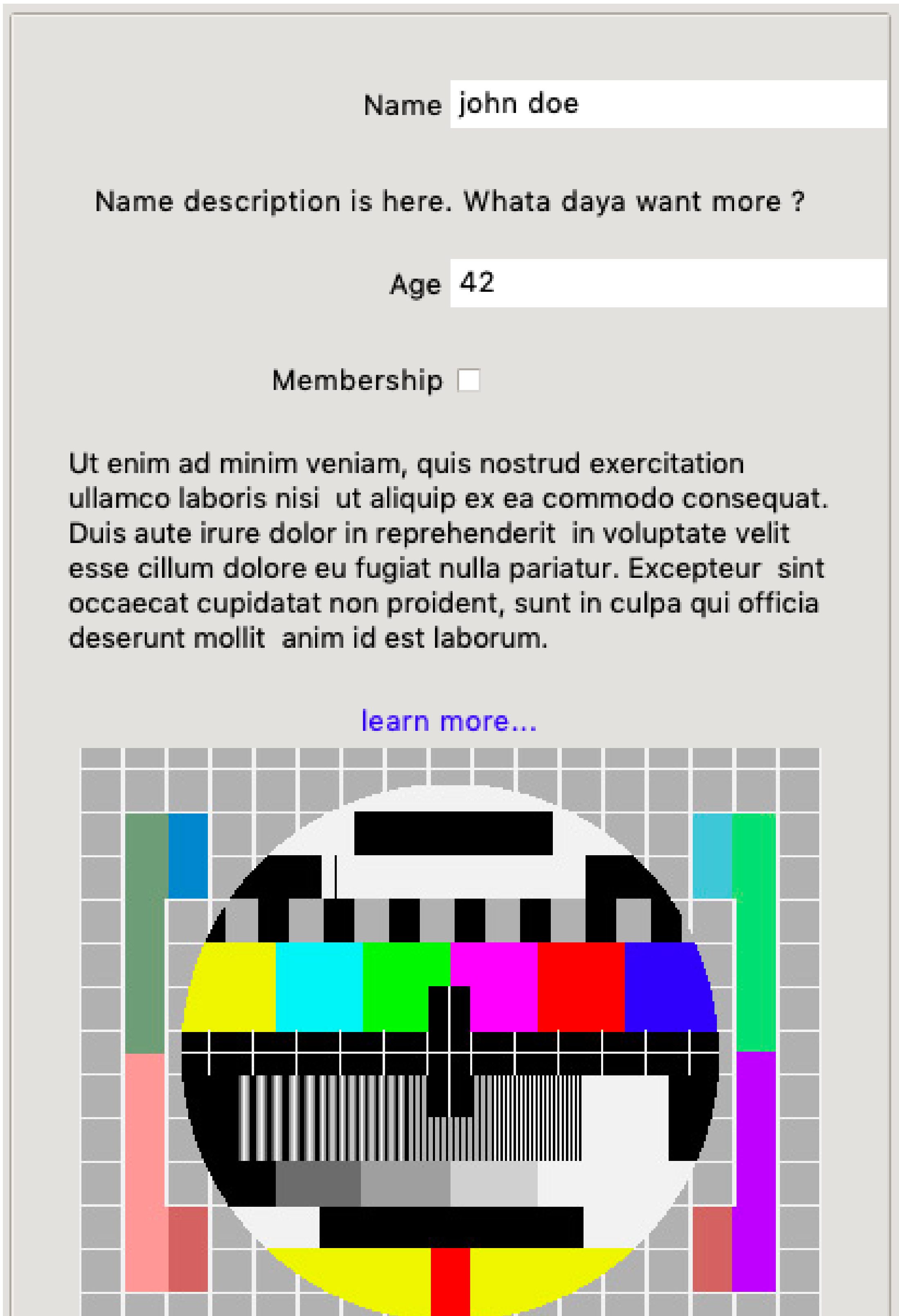
```
def custom_fun(nob_in):
    """Update the result."""
    nob_out = nob_in.copy()
    operation = nob_get(nob_in, "operand")
    nb1 = nob_get(nob_in, "number_1")
    nb2 = nob_get(nob_in, "number_2")

    res = None
    if operation == "+":
        res = nb1 + nb2
    elif operation == "-":
        res = nb1 - nb2
    elif operation == "*":
        res = nb1 * nb2
    elif operation == "/":
        res = nb1 / nb2
    else:
        res = None

    # raise RuntimeError("Tahiti a plante ce processus")

    nob_set(nob_out, res, "result")
    return nob_out


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

Concerning **Error Handling**. OpenTEA calls a Sub process of a python script. Therefore, a failue in the script will not freeze the application. The current Tab becomes red, with the message typically *Failed after 0.16s*. You can customize : if the script raises a `RunTimeError("foobar")`, the error string (here `"foobar"`) will be copied to the button status, typically *Failed after 0.16s, RunTimeError: foobar*

5.1.3 Third Node level, the block

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

It looks like the following:




```
......(tab)
      first_block:
```

(continues on next page)

(continued from previous page)

```
      type: object
      title: Customer Info
      description: >

        Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
        ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
        in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
        sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt␣
↪mollit
        anim id est laborum.

      documentation: >

          # title

          ## subtitle

          Lorem ipsum dolor sit amet, consectetur adipiscing elit,
          sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
          Ut eru[avbp website](http://www.cerfacs.fr/avbp7x/)  nisi
          ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
          in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
          sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt␣
↪mollit

          ## subtitle

          Lorem ipsum dolor sit amet, consectetur adipiscing elit,
          sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

          ![image](test-pattern-tv.jpg)

          | add  | bdfsf | vxgc | sds | vwv  |
          |---|---|---|---|---|
          | 11  | 12  | 13  | 14  | 15  |
          | 21  | 22  | 23  | 24  | 25  |

      image: test-pattern-tv.jpg
      properties:
        name:
          ...
        age:
          ...
        membership:
          ...
```

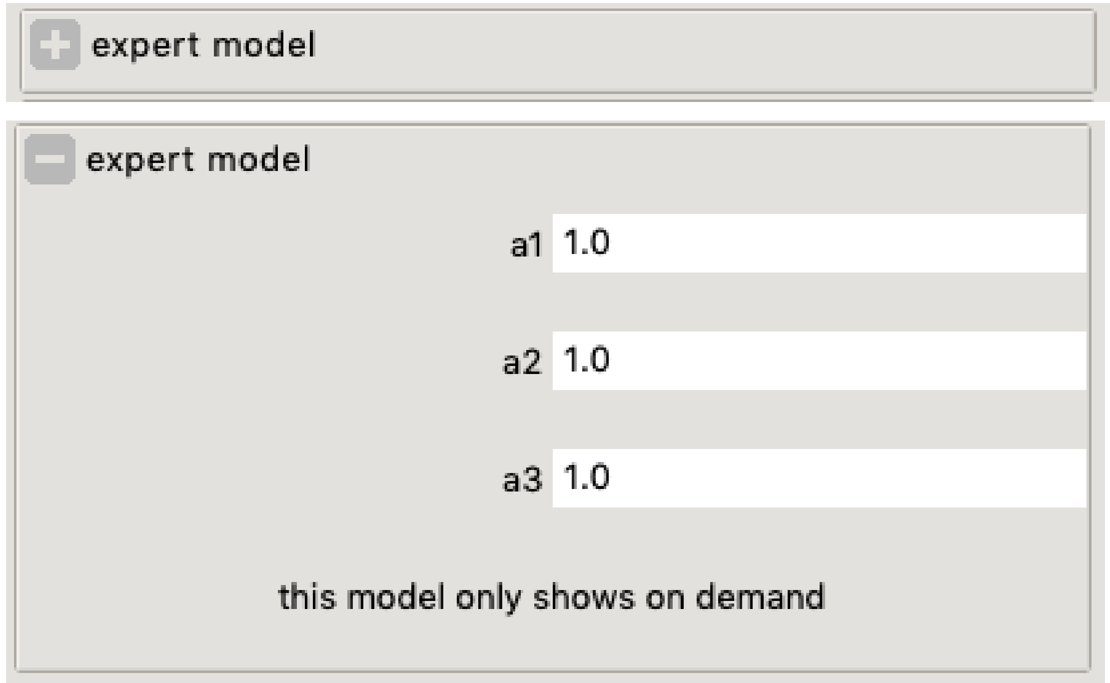
Block nodes, in yaml, are found after three indentation (6 spaces, providing you use the standard 2 spaces chars indentation). The content is found after four indentations / 8 chars.

You can nest more blocks under blocks if needed.

The Block level is accepting the following additionnal attributes:

- `description`. This attributes is the SCHEMA official attibute. It takes a string. The string will be shown in the GUI at the bottom left of the block.
- `image`. This attribute is specific to OpenTEA, and does not belong to the SCHEMA standard. The image must be stored in the folder of the main schema file. It will be shown, without scaling, at the bottom of the block.

- `documentation`. This attribute is specific to OpenTEA, and does not belong to the SCHEMA standard. It takes a string using Markdown syntax. This wil add at the bottom of the block the blue label "learn more…". On click this label trigger the opening of the browser, loading the HTML rendering of the Markdown content. Al features of Markdown are supoorted. Images must be stored at the root of tge GUI, where the schema is.
- `expert`.This attribute is specific to OpenTEA. It make the block collapsable. If `expert`is set to True, the block is initially collapsed. A click on the + / - witl expand-collapse it.



5.1.4 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)
  name:
        title: "Name"
        type: string
        default: "john doe"
   age:
        title: "Age"
        type: integer
        default: 42
   age:
```

(continues on next page)

(continued from previous page)

```
        title: "Weight"
        type: number
        default: 13.2
    membership:
        title: "Membership"
        type: boolean
        default: False
```

The appearance is the following:

The gui will check the type of the entry, and refuse invalid inputs.

You can add further validation rules, to prevent non-acceptable values righ from the form, using the SCHEMA validators. In the following example, the user cannot enter a number outside ofthe range]1, 2[. :

```
(block)
ent1:
   default: 1.3
   exclusiveMaximum: true
   exclusiveMinimum: true
   maximum: 2
   minimum: 1
   title: Essai double_gt1_lt2
   type: number
```

5.1.5 simple arrays (lists)

The SCHEMA arrays, for the simplest ones, are equivalent to Python's lists. In the following example, the list is modifiable by the user, from 0 to 999 elements.

```
(block)
list_patches:
   type: array
   title : Liste des patches
   items :
     type : string
     default: single_patch
```

The list entries look like:

this is given by the SCHEMA:

```
entlist:
       title: List entries
       type: object
       description: >
           List entries
       properties:
         entl1:
           title: String - dynamic
           type: array
           items:
             type: string
```

(continues on next page)

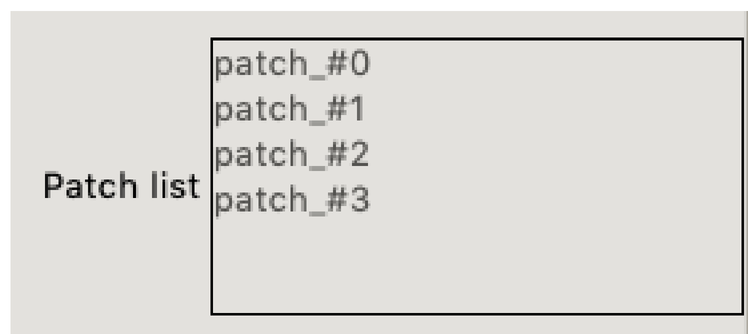
(continued from previous page)

```
            default: Catch a tiger
        entl2:
          title: Integer - X4
          type: array
          minItems: 4
          maxItems: 4
          items:
            type: integer
            default: 42
        entl3:
          title: Number - 3X
          type: array
          minItems: 3
          maxItems: 3
          items:
            type: number
            default: 666.
```

5.1.6 Disabled state

You can set an entry in *disabled* state when you set the opentea-specific attribute `state= disabled`. The user wil not be able to act directly on the value. You can howver promatically modify the value by changing the memory in the callback. In the following example, a list of string that will be modified by setting the node `list_patches` to a list of strings.

```
(block)
list_patches:
   type: array
   title : Patch list
   state : disabled
   items :
     type : string
     default: single_patch
```



Choices

The SCHEMA notation for used-defined entries options is the `enum` attribute :

```
(block)
ndim_choice:
  default: two
```

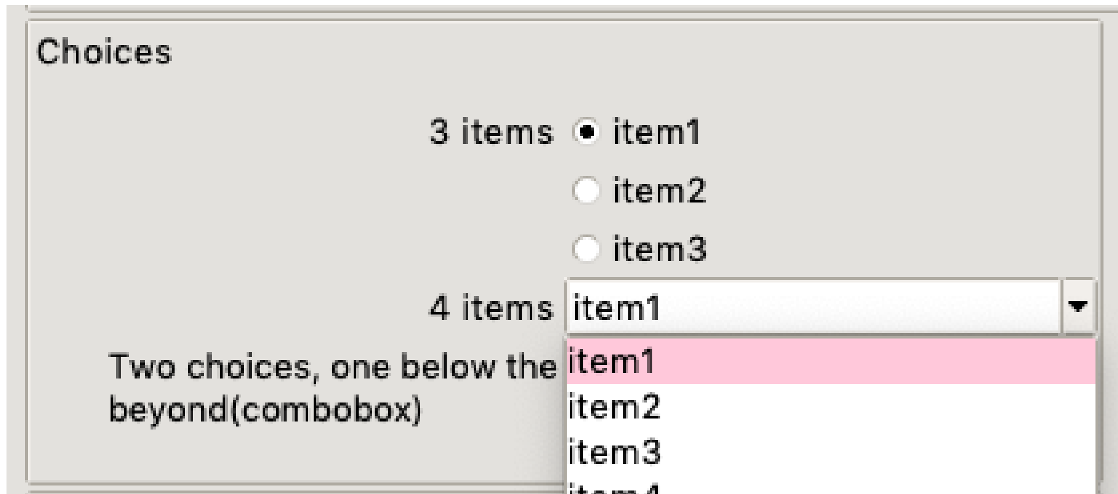
(continues on next page)

(continued from previous page)

```
  enum:
  - two
  - three
  enum_titles:
  - 2-D
  - 3-D
  title: Dimensions
  type: string
```

Note the attribute `enum_titles` specific to opentea, to override the Titles shown in the GUI.

The choice is initially a radiobutton, but with switch to a combobox beyond 3 items:



In some cases you want to create a choice between options that will be known only at run-time. This is a **dynamic choice**. In the following example, the list `list_patches` is updated by some callbacks. The choice `choice_patches` will have its options list updated when `list_patches` changes.

```
list_patches:
    type: array
    title : Liste des patches
    state : disabled
    items :
      type : string
      default: single_patch
choice_patches:
    title: Choix patches
    type: string
    ot_dyn_choice: list_patches
```

Comments

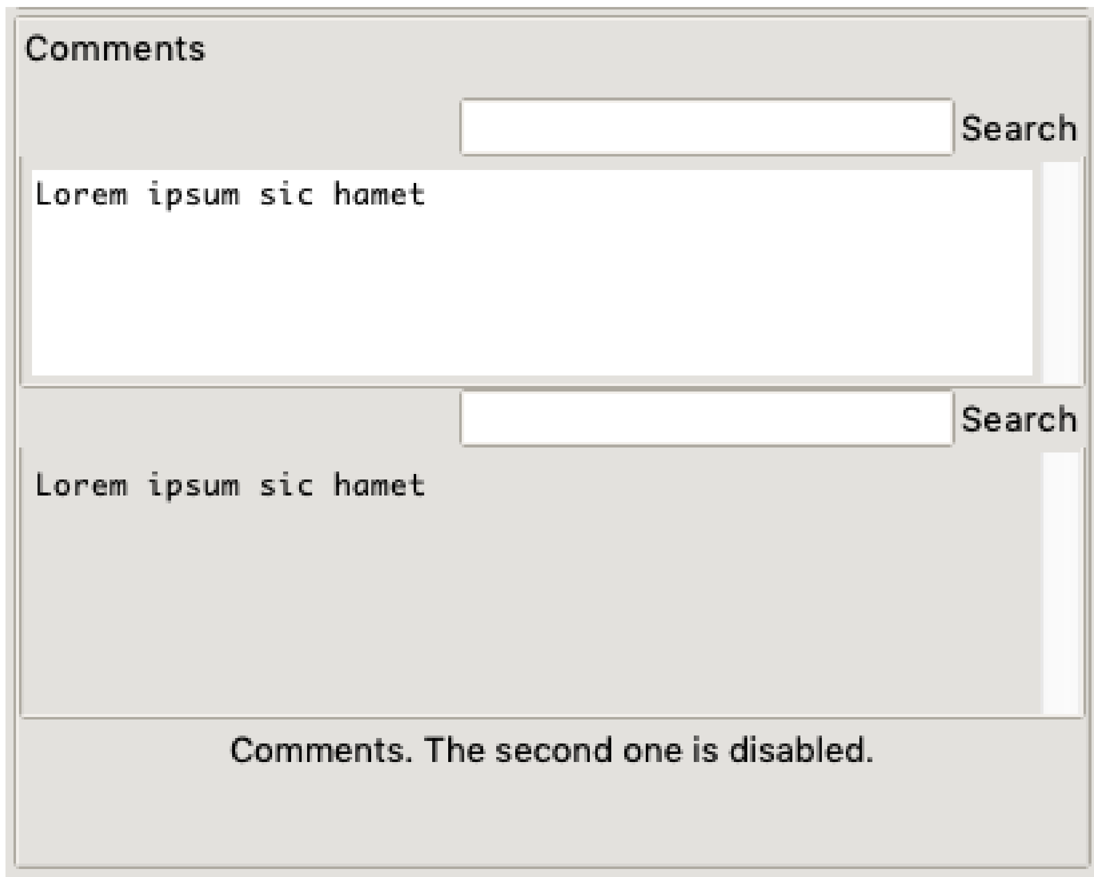
If you want to create an input on multilines, openTea offer the widget comment. It is basically a Textbox. In the yaml, add simply the decorator `ot_type: comment`to a string. In the following example, two comment entries are created.

```
 mod_comment:
  title: bossa nova
  type: string
```

(continues on next page)

(continued from previous page)

```
  default: Lorem ipsum sic hamet
  height: 20
  ot_type: comment
readonly_comment:
  title: fdo
  type: string
  default: Lorem ipsum sic hamet
  state: disabled
  ot_type: comment
```



Comments. The second one is disabled.

The `height` attibute allow to increase the size of the widget. Its default appearance is on 6 lines. The `state=disabled`allow to deactivate the user-interaction. The widget content can only be updated by a callback, which is usefull to present logfiles, informations or input_files generated by the GUI.

5.1.7 Files and folder

When you want a dialog to set a file or a folder, add the attribute `ot_type: file`. The entry will include a small button starting a "Selecting file dialog".

You can limit the search to some extentions using the atrribute "ot_filter: [h5]" (for .h5 files). You also can limit to directories using the attribute "ot_filter: directory". In the following example, two widgets are created, a H5 file selector and a directory selector.

```
  file4:
  ot_type: file
```

(continues on next page)

(continued from previous page)

```
  ot_filter:
  - h5
  title: Choix fichier (*.h5)
  type: string
  default: any.h5
file5:
  ot_type: file
  ot_filter: directory
  title: Choix repertoire
  type: string
  default: anyfolder
```

5.2 Special blocks

Special blocks are structures allowing more complexity in the nested object

5.2.1 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 <https://json-schema.org/understanding-json-schema/reference/combining.html#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. Erm… brace yourselves, this is the hardest SCHEMA part you will encounter in this manual:

```
... (block or tab)
        purchase:
          title: "select purchase"
          type: object
          oneOf:
```

(continues on next page)

(continued from previous page)

```
            - 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:
                      ...
```

The XOR Widget full supports the attributes `description`, `documentation`or `image`, like the other blocks.

5.2.2 Multiple objects

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

It is a big widget, with a treeview on the left, and a flipform on the right:

#mul_cont

	B.C. liquid	bnd_gas	param-order
patch_#0	inlet	inlet	42
patch_#1	outlet	outlet	1
patch_#2	outlet	outlet	0
patch_#3	outlet	outlet	0
patch_#4	outlet	outlet	0
patch_#5	outlet	outlet	0
patch_#6	outlet	outlet	0
patch_#7	outlet	outlet	0

load

patch_#0
Boundary cond.
patch name patch_#0
B.C. liquid
inlet
inlet
velocity 0.0
#bnd_gas
inlet
inlet
pressure 101325.0
tempoerature 300.0
velocity 0.0
param-order 42

```
  ... (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
            state: disabled
          veggieLike:
            type: boolean
            description: Do I like this vegetable?
            default: False

Opentea requires a **compulsory string property**\ , ``name`` that you must set as
↪read/only (see example before),
```

This will help to handle the content of the multiple. Indeed, If your multiple dialogue handle mode than 20 items, you will be happy to use names and not list index... trust me.

The Multiple Widget DOES NOT support the attributes `description`, `documentation`or `image`.

Multiple with dependency

You can link the multiple to another value using the openTEA specific `ot_require` keyword. It must refer to an existing node, preferably a list of strings, like here the `list_patches` information.

```
mul_cont:
  items:
    type: object
    title: Boundary cond.
  ot_require: list_patches
  type: array
  properties:
    (...)
```

If this list of patches is updated, the items under the multiple influence will be updated.

5.3 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
```


5.4 Style adjustments

Most of the styling in the OpenTEA GUI is automatic. The layout, the colors and the widgets cannot be overriden.

The GUI developer can however tune some aspects:

5.4.1 Theme

OpenTEA is powered by Tkinter, and rely on Tkinter themes. The default theme is `clam`, available on all platforms. You can force a Tkinter theme available on your platform (`aqua` on OSX for example), using the optional argument `theme="aqua"` on the startup function `main_otinker()`.

5.4.2 Images

The images in the GUI are introduced either with the attribute 'image' in blocks, or in the markdown documentation.

5.4.3 Block Descriptions

Block descriptions can be tuned with the following tags inserted in the text:

- `<small>` decrease the font size to 12
- `<tiny>` decrease the font size to 10
- `<bold>`chenge text to bold
- `<italic>` change text to italic

For example, the following input will create a description with italic, 12pts default font.

```
description: >
            <small> <italic> Lorem ipsum sic hamet
```

CHAPTER 6

opentea package

6.1 Subpackages

6.1.1 opentea.gui_browser package

Submodules

opentea.gui_browser.otbrowser module

6.1.2 opentea.gui_forms package

Submodules

opentea.gui_forms.constants module

opentea.gui_forms.leaf_widgets module

opentea.gui_forms.node_widgets module

opentea.gui_forms.otinker module

opentea.gui_forms.root_widget module

opentea.gui_forms.wincanvas module

6.1.3 opentea.noob package

Submodules

opentea.noob.asciigraph module

opentea.noob.check_schema module

opentea.noob.inferdefault module

opentea.noob.noob module

opentea.noob.schema module

opentea.noob.validate_light module

opentea.noob.validation module

6.1.4 opentea.tools package

Submodules

opentea.tools.proxy_h5 module

opentea.tools.schema2md module

opentea.tools.visit_h5 module

6.2 Submodules

6.3 opentea.cli module

6.4 opentea.process_utils module

CHAPTER 7

Indices and tables

- genindex
- modindex
- search