Opentea Documentation

Release 3.6.0

Team COOP

Jul 21, 2023

CONTENTS

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

1.1 Installation

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

```
>pip install opentea
```

then test your installation with

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

1.2 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 :

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

```python
"""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
    elif operation == "-":
        res = nb1 - nb2
    elif operation == "*":
        res = nb1 * nb2
    elif operation == "/":
        res = nb1 / nb2
```

(continues on next page)

```python
    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.*

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

```yaml
first_tab:
  first_block:
    number_1: 32.0
    number_2: 10.0
    operand: +
    result: 42.0
```

TWO

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.
```

THREE

ACKNOWLEDGMENTS

FOUR

BUILDING THE GUI

4.1 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
```
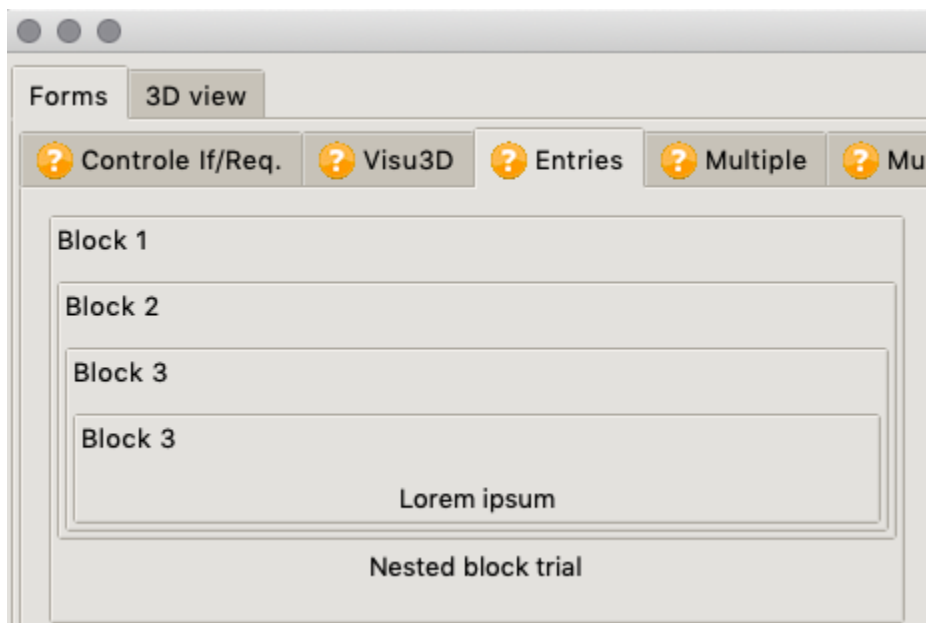
 Root is the Major "Forms" tab. Tabs are shown in the second line with oragne icons. Then four nested blocks are shown

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.

###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.

4.1.1 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:

```python
"""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 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*

4.1.2 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:
      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
```

(continues on next page)

```
        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.
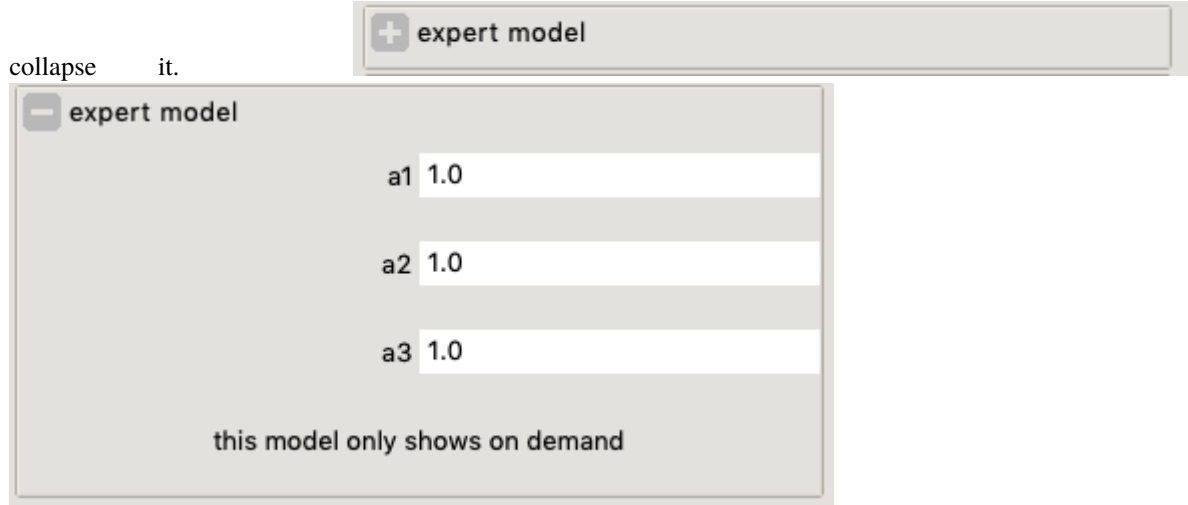
4.1.3 Leaf level , or Parameters

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

(continues on next page)

```
                      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
```

4.1.4 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
            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.
```

4.1.5 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
```
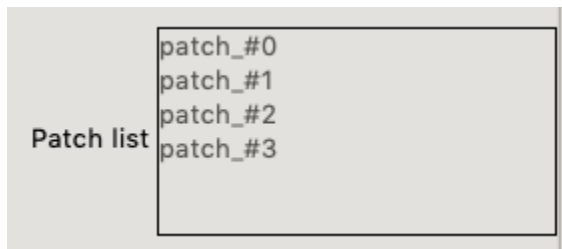

Choices

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

```
(block)
ndim_choice:
  default: two
  enum:
  - two
  - three
  enum_titles:
  - 2-D
  - 3-D
  title: Dimensions
  type: string
```

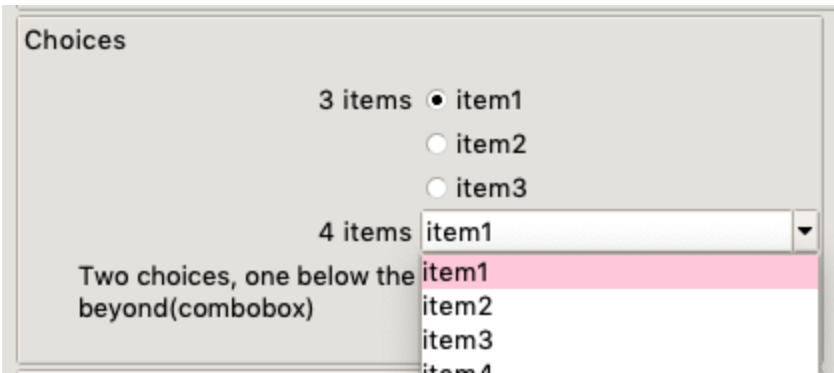Note the attribute `enum_titles` specific to opentea, to override the Titles shown in the GUI.

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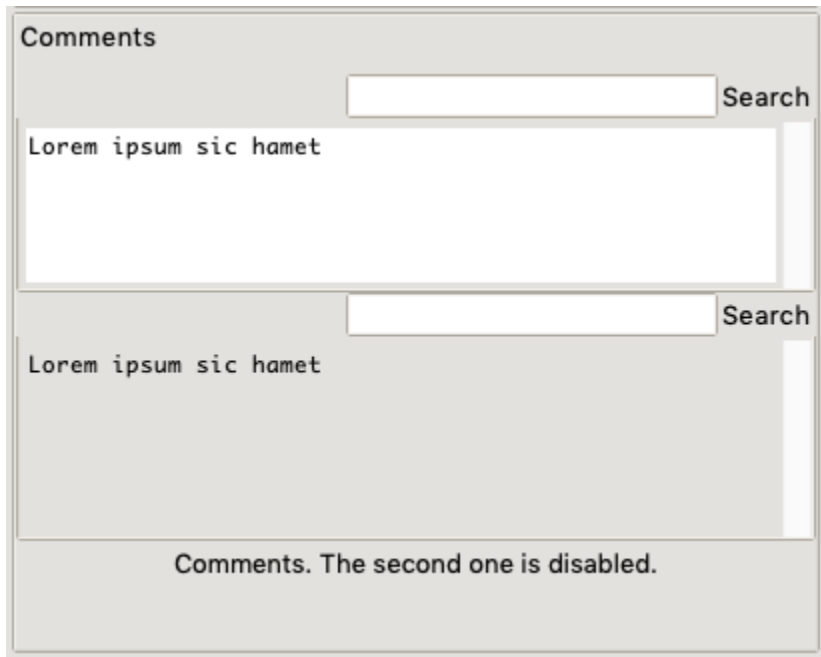
The `height` attibute allow to increase the size of the widget. Its default appearance is on 6 lines. The `state=disabled`allow to deactivate the user-interaction. The widget content can only be updated by a callback, which is usefull to present logfiles, informations or input_files generated by the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

4.1.6 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
    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
```

4.2 Special blocks

Special blocks are structures allowing more complexity in the nested object

4.2.1 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, 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:
          - 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.

4.2.2 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:


```
... (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, documentationor 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.

4.3 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
```

4.4 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:

4.4.1 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()`.

4.4.2 Images

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

4.4.3 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
```

FIVE

CHANGELOG

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

5.1 [3.6.0] 2023 / 07 / 02

5.2 Added

- Verbose mode on the terminal with loguru output

5.3 Changed

- move from package logging to loguru

5.4 [3.5.0] 2023 / 02 / 02

5.5 Added

- Support to `tiny_2d_engine` for enmbedding 2D interactive canvas.

5.6 Changed

- Removal of the aggressive splashscreen when starting a new project
- XOr menus now feature an automatic submenus creation is several prefixes are repeated
- switch to combo boxes is moved from 3+ to 5+ choices.
- radiobuttons are sunken to ease options clustering.

5.7 Fixed

- Display labels in containers corrected
- Spurious status changes fixed
- Simplification of coding

5.8 [3.4.4] 2022 / 09 / 12

5.9 Fixed

- `validate_light` extended to allow : {"void": None} for string validation, because opentea moved void container description from : None to : {"void": None}.

5.10 [3.4.3] 2022 / 07 / 13

5.11 Changed

- entries of widget file path are scrollable

5.12 Fixed

- reload of dependant boundaries
- bug on disabled lists

5.13 [3.4.2] 2022 / 07 / 04

5.13.1 Changed

- `mem_change` and `mem_check` events removed: goal is to be more explicit
- tree (or graph, for exactness) structure well defined for widgets: it is very easy to fully traverse the tree
    - the tree can be trasversed by assuming all children are dicts
- `status` plays a central role now:
    - controls style of widgets
    - controls update of tab icon
- `status` computation:
    - to increase performance, traverse of full tree is avoided: status computed bottom-up (everytime a leaf status changes, statuses are updated until the top (but only following that path)
- `tree` -> `children`
- `tab` not passed as input to widgets, as it can quickly be retrieved

- `previous_value` comparison for coloring done against saved value

- new objects to handle creation of related objects (e.g. `OTChoice`)

- improved naming regarding public and private objects

- default is set when creating object, instead of using `nob_complete`

- improved `TextConsole`

- more abstraction

- handle of dependents: instead of a global event everytime there's a change in a leaf, leaves now "know" which nodes/leaves depend on them and update them accordingly

- menus are now individual objects: building menus by composition should be trivial (specially useful for connection to external apps, such as `neverd`)

- connection to external apps simplified:

    - `tab_3d` key is still supported, but external apps can now be connect via their own `yaml`

5.13.2 Fixed

- `smartpacker` non-visible widgets influence

5.14 [3.4.1] 2021 / 10 / 04

5.14.1 Added

- popup menu with copy, paste and move for `OTMultipleWidget`

- new bindings in `OTMultipleWidget` (move and deselect)

- move up and down buttons in `OTMultipleWidget`

- highlight widgets when they are changed (by changing color)

- add scrollbar to disabled listboxes (`OTListStatic`)

- `MouseScrollFrame`: scrollable frame scrollable via mouse wheel

- add copy-paste popup menu to `OTList` and `OTListStatic`

- add copy-paint binding to `OTMultipleWidget`

- add row number to multiple treeview

- `OTHidden` leaf to make it easier to have nodes that can change value, but are not visible

- add several warning message boxes in `OTMultipleWidget`

5.14.2 Fixed

- remove double description when `ot_type` in schema
- bindings in `OTMultipleWidget`
- no underscores required as prefix in `OTMultipleItem` names
- `OTMultipleItem` load button behavior
- multiple changes that were not triggering tab icon change
- checkboxes, menus and radiobuttons do not trigger any action if current value is chosen again
- `OTFileBrowser` does not raise error when user cancel path search
- `OTList` behavior: validation of input and error message
- theme selection
- empty titles are not shown in `OTContainer`

5.14.3 Changed

- `OTMultipleWidget.tree` is now a `dict` (instead of `list`)
- split `OTList` and `OTChoice` in smaller objects for easier mantainability
- replaced several `tk.Label` by `ttk.Label` in order to make the most out of styles
- multiple treeview is now its own object (`MultipleTreeview`)
- name only appears in switchform title in `OTMultipleItem`
- entries with no validation requirements (strings) do not have extra space for status label anymore

5.15 [3.3.1] 2021 / 05 / 07

5.15.1 Fixed

- Fix a pb of repeated items for generation of Hybrid GUIs
- Fix the Import Tk issues for usage of process_utils on non graphical environments.
- Add a .yml filter on file dialogs

5.16 [3.3.0] 2021 / 03 / 09

5.16.1 Added

- adding `opentea.__version__` attribute though VERSION file
- adding a new project management. A project name is asked if missing
- nobvisual inspection of projects
- recursive fusion of SCHEMA, for composite GUIs

5.16.2 Changed

- the project is saved at each "Validate/Process"
- cursor switch to waiting mode during "Validate/Process"
- the temp. files like `dataset_to_gui.yml` are now hidden as `.dataset_to*`
- help windows are now rendered using the tkhtmlview package as a top-level window.
- File dialogs store relative paths, not absolute paths

5.17 [3.2.3] 2020 / 11 / 09

5.17.1 Fixed

- the widget comment is no more recursively adding blank lines.

5.18 [3.2.2] 2020 / 11 / 09

5.18.1 Changed

- nob_complete can now keep the input data that was not in the SCHEMA

5.18.2 Fixed

- some loop holes in validate_light

5.18.3 Deprecated

- H5proxy is deprecated, should be replaced by hdfdict

5.19 [3.2.1] 2020 / 06 / 04

5.19.1 Changed

- Statuses of tabs are recovered when reading a project
- search disabled in consoles by default
- CLI improvements

5.19.2 Fixed

- spurious dependency on python 3.7 for subprocess, now 3.6 is fine too
- no more deprecated calls to 3D engine
- for documentation : use recomonmark instead of m2r (deprecated)

5.20 [3.2] 2020 / 03 / 13

5.20.1 Added

- expert dialogs
- dynamic choices
- support of 3D viewer with tiny_3d_engine

5.20.2 Changed

- red output if an error is onprocess
- comments with clear enable/disable mode
- description can tag style tags for itlaliv, bold, small or tiny texts.

5.20.3 Fixed

- bug on multiple with depencencies if the list was going to zero

5.21 [3.1.1] 2020 / 01 / 15

5.21.1 Fixed

- deprecation warning from **h5proxy** removed (consider using hdfdict or H5wrapper instead..)

5.22 [3.1.0] 2019 / 12 / 05

5.22.1 Added

- **h5proxy** to read quickly in an H5file (consider using hdfdict or H5wrapper instead..)
- **schema2md** convert schema file into HTML tables

5.23 [3.0.0] 2019 / 03 / 26

5.23.1 Added

- Tkinter Graphical engine
- noob library
- nob_complete
- nob_validate

5.23.2 Deprecated

- All Tcl and Python for Version 2

5.24 [2.3] 2018 / 10 / 17

5.24.1 Changed

- Documentation using sphinx
- widget info can display icon depending on content

INDICES AND TABLES

- genindex
- modindex
- search