mutaprops Documentation

Release 0.6.6

Josef Nevrly

Jan 31, 2019

Contents

Contents:

CHAPTER 1

mutaprops

Mutated Properties - a simple HTML5 property-configuration UI, autogenerated from your classes.

It's great if you need a quick'n'dirty UI with minimal effort and without changing much of the existing codebase. Mutaprops also thrive on headless systems when usual UI solutions like tkinter doesn't make sense.

However, the customization possibilities are limited, so if you are looking for some framework for building a full-fledged attractive GUI, better look elsewhere.

1.1 Features

- Generate a self-documented web UI directly from your objects with simple decorators
- UI state automatically updated with object state changes (through websockets)
- Supports multiple UI sessions on the same object, synchronized through websockets
- Supports clustering of UI's from multiple machines
- UI look and feel can be customized with your own stylesheet
- Add any widget you like with direct HTML support
- HTML5 log console capturing all your Python logging
- Asyncio support (and also a requirement ;))

1.2 The simplest example

Imagine a normal Python class:

```
class Hoovercraft:

    MAX_EELS = 40
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, number_of_eels=20, speed=0, direction='North'):
        self._eel_count = number_of_eels
        self._speed = speed
        self._direction = direction
        self._engine_running = False
        self._steering_locked = True

    @property
    def eels(self):
        return self._eel_count

    @eels.setter
    def eels(self, value):
        self._eel_count = value
        if self._eel_count >= self.MAX_EELS:
            logger.warning("The hoovercraft is full of eels!")

    def drop_all_eels(self):
        self.eels = 0
        logger.info("Eels are goooone!")
```

Now, to turn this into an UI, one just has to decorate it like this:

```
from mutaprops import *

@mutaprop_class("Hoovercraft UI")
class Hoovercraft:

    MAX_EELS = 40

    def __init__(self, number_of_eels=20, speed=0, direction='North'):
        self._eel_count = number_of_eels
        self._speed = speed
        self._direction = direction
        self._engine_running = False
        self._steering_locked = True

    @mutaproperty("Number of eels", MutaTypes.INT, min_val=0,
                  max_val=MAX_EELS)
    def eels(self):
        return self._eel_count

    @eels.setter
    def eels(self, value):
        self._eel_count = value
        if self._eel_count >= self.MAX_EELS:
            logger.warning("The hoovercraft is full of eels!")

    @mutaprop_action("Drop all eels!")
    def drop_all_eels(self):
        self.eels = 0
        logger.info("Eels are goooone!")
```

And then run it like this:

```
if __name__ == '__main__':

    test = Hoovercraft()
    test.muta_init("Hoovercraft instance #1")
    man = HttpMutaManager("Hoovercraft manager", proxy_log=logger)
    man.add_object(test)
    man.run(port=9000)
```

Et voila, here's the UI:

1.3 Other examples

The `examples/` folder contains several other examples:

- simple_example.py is the extension of the example above, including more data types and also shows how to work with docstrings and `mutasources`
- advanced_example.py demonstrates grouping of parameters, style customizations, raw HTML features and asyncio integration.

1.4 Full documentation

The complete documentation is available at https://mutaprops.readthedocs.io

1.5 Using the UI

Simple explanation how to use the UI is here.

1.6 Credits

The default logo created with the Chlorinar font.

The JavaScript frontend created with the fantastic Vue.js.

The widgets and styling are based on Bootstrap 3.

The toggle widget is the Bootstrap toggle.

Hoovercraft logo used in advanced_example.py was created by Theresa Stoodley from the Noun Project. Licensed under Creative Commons 3.0 license.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install mutaprops, run this command in your terminal:

```
$ pip install mutaprops
```

This is the preferred method to install mutaprops, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for mutaprops can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/calcite/mutaprops
```

Or download the tarball:

```
$ curl  -OL https://github.com/calcite/mutaprops/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use `mutaprops` in a project:

```
from mutaprops import *
```

3.1 Decorating classes

`mutaprops` are created by decorating classes with some custom decorators, much like the usage of the @property decorator. Those decorators enable the property manager to see the properties and track their state.

@mutaprop_class [*`mutaprop_class()`*] is the mandatory class decorator which enable objects instantiated from the decorated classes to be tracked by the property manager.

@mutaproperty [*`mutaproperty()`*] is the basic decorator. Any attribute decorated with `@mutaproperty` becomes visible in the UI. The arguments of `@mutaproperty` allow to define the type of the parameter (that defines the UI widget), it's display name, numerical ranges, the position/category in the UI etc.

The docstring of the attribute is displayed as help in the UI. Any change to this attribute during run-time is reflected in the UI. Any change in the UI causes change of the attribute's value.

Same as with usual properties, `@mutaproperty` can be read only, either by not defining a setter function, or by explicitly setting it to `read_only`.

Most parameters of the `@mutaproperty` can be assigned as a constant, or as a reference to another `@mutaproperty` or `@mutasource`. Change of a referred `@mutaproperty` then causes UI update of the referee.

All optional parameters can be specified either in getter or in setter decorator, it doesn't matter which.

@mutaprop_action [*`mutaprop_action()`*] is basically a simplified version of `@mutaproperty`. It's represented as a button on UI level, and causes direct function call of the decorated function when the button is pressed.

@mutasource [*mutasource()*] is best described as a "hidden mutaproperty". `@mutasource` is not directly visible, but it's changes are propagated to the UI. It can be used to implement some controller (as in MVC) functionality which would be hard to do with `@mutaproperty` alone.

All above-mentioned decorators wrap the original class implementation and add some extra functionality. None of it is used, however, if the instantiated objects are not registered with the UI manager. One therefore doesn't need to be concerned about the functionality/performance impact of the decorators on the original class.

3.2 Setting up an UI manager

HttpMutaManager does all the heavy-lifting, providing data from registered object to the UI front-end and tracking the object changes and user actions. It runs on aiohttp.

UI manager is basically a REST service (the UI front-end is a HTML5 app) with additional websocket component.

Apart of the bi-directional object data update, the manager is also responsible for forwarding the registered logger messages to the front-end, and also serving static files (the HTML5 app blob and custom user data).

3.2.1 Using external Asyncio loop

By default, the manager would create a new loop upon start. If the application is using `asyncio` as well, manager can be specified to use the application's loop.

```
import asyncio
from mutaprops import *

loop = asyncio.get_event_loop()

# Some nice async code

man = HttpMutaManager("Some manager", loop=loop)
```

3.2.2 Custom static files

UI manager can serve a custom static files as well. This can be utilized in several ways:

- providing image files for the auto-documentation
- providing logos and custom stylesheets for the UI customization

All custom files must be placed in one directory structure, the path to this directory can be specified as the `local_dir` argument.

```
from mutaprops import *

man = HttpMutaManager("Some manager", loop=loop,
                      local_dir="path/to/custom/dir")
```

The contents of the `local_dir` will be served in the path `/local`.

3.2.3 Custom style modification

To modify the UI look-and feel, one can produce a custom stylesheet and override the stylesheet in `mutaprops/web_ui/dist/base.css`.

This stylesheet must be called `custom.css` and must be placed in the `local_dir`. Any additional asset files (logos, images etc.) must be placed there as well.

3.2.4 UI self-documentation and help files

Each parameter is documented with its own docstring (ReST can be used for formatting).

On top of that, an additional help text can be displayed in the help window (activated by the help link in the menu bar). This text is specified as the `help_doc` argument, the content must be a string containing HTML code.

To translate a ReST text into suitable `help_doc`, the *`mutaprops.utils.rest_to_html()`* can be used.

```
from mutaprops import *
from mutaprops.utils import rest_to_html

help = """
Help-less
---------

This help is of *NO USE* at all!
"""

man = HttpMutaManager("Some manager", loop=loop,
                      local_dir="path/to/custom/dir",
                      help_doc=rest_to_html(help))
```

3.2.5 Log forwarding

UI manager is capable of forwarding the `logging` messages to the HTML UI. In order to do so, a logger has to be registered with the manager using the `proxy_log` argument.

```
from mutaprops import *
import logging

logger = logging.getLogger(__name__)

man = HttpMutaManager("Some manager", loop=loop,
                      local_dir="path/to/custom/dir",
                      proxy_log=logger)
```

By default, all log messages are forwarded. The log level can be further specified by the `log_level` argument.

3.2.6 Manager clustering and chaining

Several running UI manager instances can be connected into one *master* manager. This feature can be used for example to create a joint UI console for several headless machines (e.g. RPi's) that are used in a cluster, each running it's mutaprops UI.

Furthermore, such clusters can be chained (a master manager connects to another, higher-level master manager). There is no fixed limit for such chains, however in practice the latencies will increase with each chain link.

```
# Running on machine 192.168.1.1
from mutaprops import *

```

(continues on next page)

(continued from previous page)

```
man = HttpMutaManager("Some master manager")
dead_parrot = SomeDecoratedClass()
dead_parrot.muta_init("Parrot #1")
man.add_object(dead_parrot)
man.run(port=9000)
```

```
# Running on machine 192.168.1.2
from mutaprops import *

man = HttpMutaManager("Some slave manager",
                      master='http://192.168.1.1:9000')
dead_parrot = SomeDecoratedClass()
dead_parrot.muta_init("Parrot #2")
man.add_object(dead_parrot)
man.run(port=9000)
```

In the above example, the `Parrot #2` will appear in the UI served at `http://192.168.1.1:9000`, alongside the `Parrot #1`

At the same time, it will be also accessible from it's own UI served at `http://192.168.1.2:9000`. Any change on any of those UI's will be reflected on the other UI as well.

3.3 Registering and unregistering objects with UI manager

Once an object is instantiated from the muta-decorated class, there is nothing special going on with it until the mutaproperties are initialized and registered with the UI manager.

To initialize the mutaproperties, the *muta_init()* must be called. During this initialization process, an ID (UI-visible name) is assigned to a given object.

```
from mutaprops import *

dead_parrot = SomeDecoratedClass()
dead_parrot.muta_init("Parrot #1")  # ID is assigined to the dead_parrot instance.
```

After mutaproperties are initialized, the object can be added to an existing UI manager using *add_object()*.

```
man = HttpMutaManager("Some master manager")
man.add_object(dead_parrot)
```

The whole process can be simplified by initializing the mutaproperties when adding the object to the manager

```
dead_parrot = SomeDecoratedClass()
man = HttpMutaManager("Some master manager")
man.add_object(dead_parrot, "Parrot #1")  # Mutaproperties initialized while adding
```

Objects that were once added to the manager can be removed in the similar fashion using the *add_object()*.

```
man.remove_object(dead_parrot)
```

3.4 Running the manager

UI manager is basically just a server, and need to be run. In case of asyncio-based application, this is very simple, just as starting any sort of server.

```
# Continuing the example above

man.run(port=9000)
```

In case the rest of your application is not asyncio based and there are other tasks which has to be run alongside the user-initiated actions, it's possible to run the UI manager in a separate thread.

```
# Continuing the example above

man.run_in_thread(port=9000)
```

This feature is not very well tested, and by definition opens all sort of synchronization problems which needs to be dealt with by the implementator. *Use at your own risk!*.

3.4.1 Run parameters

UI manager is built on the top of the aiohttp server, it's therefore possible to use any parameters used in the aiohttp.run_app() method.

3.5 Using the UI

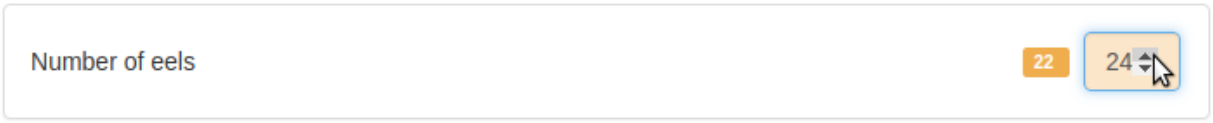
The UI itself is just a simple HTML5 application. Change of any property causes immediate change of the corresponding attribute of the underlying object.

Conversely, any change of underlying object's state causes immediate update of the UI state.

There is some color coding to help making sense of those transitions:

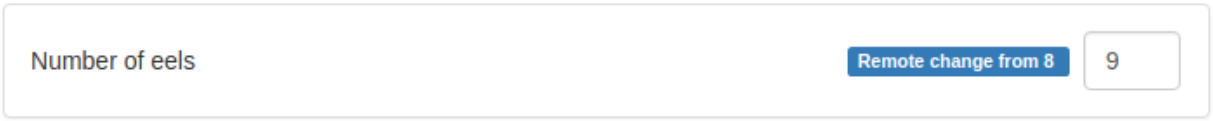
***Orange code* - value is being changed by the user** Orange label/background of the widget singalizes that the UI is currently being changed by the user and it's state does not correspond with the underlying object's state. The value on the label shows the value which is currently set on the object.

As soon as user stops changing the property value (widget looses it's focus), the value is set to the underlying object and the orange code disappears.

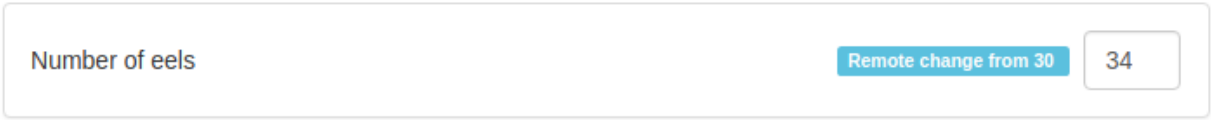


***Blue code* - value changed by the underlying object** Blue code signalizes user that the underlying object's state has changed without user's interaction. Blue label shows the last value before this change.

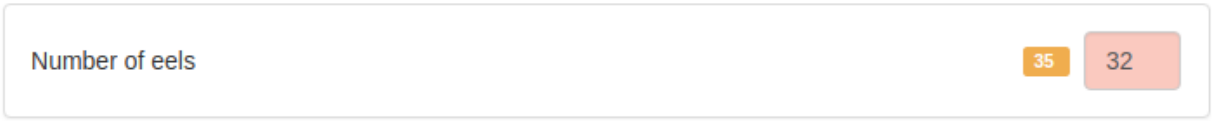
Blue label dissapears when the value is changed by the user.



***Azure code* - value changed by another user session** Since UI manager supports multiple parallel sessions (=multi-user), this mechanism signalizes the changes caused by another users. Azure label shows the last value before this change.



***Red code* - value could not be updated** Red background on the widget signalizes that the user change on the UI could not be set to the underlying object. Usually because the connection to the UI manager was lost etc.



CHAPTER 4

mutaprops

4.1 mutaprops package

4.1.1 Submodules

4.1.2 mutaprops.decorators module

mutaprops.decorators.**mutaprop_action**(*display_name*, ***kwargs*)

Make decorated method accessible in the UI.

Parameters

- **display_name** – A name to be displayed at the UI level.
- **read_only** – bool, *dynamic* Sets GUI element to read-only state. Automatically set to true when setter is not defined.
- **priority** – int Display priority, higher numbers are displayed first
- **hierarchy** – string Hierarchy path in the GUI. MutaProps with the same hierarchy patch are grouped together.
- **deford** – int Modifies definition order. This is normally defined automatically based on the decorator calls. If priority is not used, the order at which MutaProps are listed in GUI is defined by deford.

mutaprops.decorators.**mutaprop_class**(*display_name*, *gui_id=None*, *gui_major_version=0*, *gui_minor_version=0*)

Class-level decorator. It is required for classes whose instances should be visible for the Mutaprop UI manager.

Parameters

- **display_name** – Class name/description to be accessible by the UI API. In most cases, it's not really important.
- **gui_id** – API-level identifier of a class definition. In most cases, it's not really important.

- **gui_major_version** – Reserved for future use.
- **gui_minor_version** – Reserved for future use.

mutaprops.decorators.**mutaproperty**(*display_name*, *value_type*, ***kwargs*)

Create a UI property out of existing attribute.

Parameters

- **display_name** – A name to be displayed at the UI level.
- **value_type** – [*MutaTypes*]

Optional arguments - general

Parameters

- **read_only** – bool, *dynamic* Sets GUI element to read-only state. Automatically set to true when setter is not defined.
- **select** – `list` or `dict`, *dynamic* Set of allowed values. GUI will offer just this set.

  If it's `list`, the list items must conform to the *value_type*:

```
@mutaproperty("Cheese", MutaTypes.STRING,
              select=['Stilton', 'Gruyere', 'Liptauer'])
def cheese(self):
    return self._cheese
```

  If it's `dict`, the keys will be displayed as a selector view, and the values will be set:

```
@mutaproperty("Cheese", MutaTypes.INT,
              select={'Stilton': 1, 'Gruyere': 2,
                      'Liptauer': 3})
def cheese(self):
    return self._cheese
```

- **priority** – int Display priority, higher numbers are displayed first
- **hierarchy** – string Hierarchy path in the GUI. MutaProps with the same hierarchy patch are grouped together.
- **deford** – int Modifies definition order. This is normally defined automatically based on the decorator calls. If priority is not used, the order at which MutaProps are listed in GUI is defined by deford.

Optional arguments - numerical type (*MutaTypes.INT*, *MutaTypes.REAL*)

Parameters

- **min_val** – int, *dynamic* Minimum possible value,
- **max_val** – int, *dynamic* Maximum possible value,
- **step** – int, *dynamic* For numerical type, step interval between values, for other types ignored

Optional arguments - *MutaTypes.STRING*

Parameters

- **min_val** – int, *dynamic* Minimal length of string
- **max_val** – int, *dynamic* Maximal length of string

Optional arguments - *MutaTypes.BOOL*

Parameters **toggle** – Dict of format:

```
{'on': 'Some-on-label',
 'off': 'Some-off-label'}
```

If set, a toggle-switch like control will be used as GUI instead of a simple checkbox.

Dynamic arguments

Where dynamic argument is supported, the value doesn't have to be only a class definition constant, but also another *MutaProp* or *MutaSource*.

Dynamic arguments can be referred only directly and cannot be in any sort of expression, as this expression is evaluated only once during the class definition time:

```
@mutaproperty("Some useful flag", MutaTypes.BOOL)
def some_flag(self):
    return self._some_flag

@some_flag.setter
def some_flag(self, value):
    self._some_flag = value

@mutaproperty("Ham", MutaTypes.STRING,
              read_only=some_flag)  # direct referral, will work
def ham(self):
    return self._spam

@mutaproperty("Spam", MutaTypes.STRING,
              read_only=not some_flag)  #expression, will not work
def spam(self):
    return self._spam
```

mutaprops.decorators.**mutasource**(*func=None*, *class_scope=False*)

Decorated attribute's changes will be notified to the UI layer, but will not be displayed.

MutaSource allows to implement some additional controller (as in MVC) logic in addition to the usual mutaproperties. Usually used to enable or disable (read_only) some parts of UI depending on other parameters, or change the select values.

Parameters **class_scope** – [True, False] Set to reflect class-level attribute.

4.1.3 mutaprops.managers module

class mutaprops.managers.**HttpManagerProxy**(*address*)

Bases: object

Utility class representing a remote (slave) HTTP manager.

address

attach(*host_manager*)

Attaches itself to the host manager, by making it's own remote MutaObjects part of the host managers object list. Also manages WebSocket connection to the remote manager and relays the messagest to the host manager.

Parameters **host_manager** – A host HttpMutaManager object.

Returns

detach()

is_attached

is_being_removed
Host manager checks the flag to know if it shall try to reconnect with the remote manager.

session

ws_manager()
Coroutine implementing the Websocket communication task.

class mutaprops.managers.**HttpMutaManager**(*name*, *loop=None*, *master=None*, *local_dir=None*, *help_doc=None*, *proxy_log=None*, *log_level=0*)

Bases: object

Manages HTML5 gateway for controlling the MutaObjects. Each MutaObject is made accessible as REST API with websocket (SockJS) downstream channel for notifications about model update.

Managers can be chained by specifying a "master manager" upon initialization. Master manager can see and manipulate the slave manager's MutaObjects. One Manager can have only one master, but can be master of many slaves. Chains can be of any length, but in practice the feasibility of long chains will be limited by the HTTP response times etc.

EVENT_SOURCE_MASTER = 'master'

EVENT_SOURCE_OBJECT = 'object'

EVENT_SOURCE_USER = 'user'

HEADER_SUPERVISOR = 'muta-supervisor'

INDEX_FILE = b'<!DOCTYPE html>\n<html lang="en">\n <head>\n <meta charset="utf-8">\n <

NOTIFICATION_EXTERNAL_CHANGE = 'external_change'

NOTIFICATION_LOG_MESSAGE = 'log'

NOTIFICATION_OBJECTS_CHANGE = 'objects_change'

NOTIFICATION_PROPERTY_CHANGE = 'property_change'

NOTIFICATION_TERMINATION = 'terminated'

WEB_ASSETS = '/home/docs/checkouts/readthedocs.org/user_builds/mutaprops/checkouts/lat

class WsHandler(*msg_callback*, *level=0*)
Bases: logging.Handler

Handler to forward logging messages over websocket.

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

add_object(*muta_object*, *obj_id=None*)
Add decorated object to the UI manager.

Parameters **muta_object** – An instance of a class decorated with *mutaprop_class()*.

Optional:

Parameters **obj_id** – 'Id to be used for the added *muta_object*.

register_on_master(*master_addr*)

remove_object(*muta_object*)

Remove object from the UI manager. (Causes object to disappear from the UI).

Parameters **muta_object** – Object to be removed.

run(***aiohttp_kwargs*)
Run the manager.

Parameters **aiohttp_kwargs** – HTTP server parameters as defined for aiohttp web.run_app

run_in_thread(***aiohttp_kwargs*)
Run the UI manager in a separate thread.

Theoretically this allows to run the UI for code which is otherwise incompatible with Asyncio. In practice, this is a minefield and it was never properly tested.

Parameters **aiohttp_kwargs** – HTTP server parameters as defined for aiohttp web.run_app

shutdown()

class mutaprops.managers.**HttpMutaObjectProxy**(*manager_proxy*, *obj_id*)
Bases: object

Utility class proxying remote MutaObjects through REST calls.

get_object()

get_prop(*prop_id*)

get_prop_value(*prop_id*)

get_props()

is_muta_ready()

muta_id

muta_init(*object_id*, *change_callback=None*)

muta_unregister()

set_prop_action(*prop_id*)

set_prop_value(*prop_id*, *value*)

exception mutaprops.managers.**MutaManagerError**
Bases: *mutaprops.utils.MutaPropError*

4.1.4 mutaprops.mutaprops module

class mutaprops.mutaprops.**MutaAction**(*pid*, *display_name*, *callback*, ***kwargs*)
Bases: *mutaprops.mutaprops.MutaProp*

MP_CLASS_TYPE = 'action'

MP_READ_ONLY = 'read_only'

muta_call(*obj*)

class mutaprops.mutaprops.**MutaProp**(*pid*, *display_name*, ***kwargs*)
Bases: object

Abstract class defining a generic MutaProp object. Such object holds basic information about a "property" of a MutaProp-accessible class (such as ID and human-readable name) as well as position of such property in the hierarchy of all properties.

Each MutaProp implementation is expected to overload following:

- **`_allowed_kwargs()` class method,** defining kwargs which are used/allowed in the constructor.
- **`_exported_kwargs()` class method, i** defining which parameters are exported during serialization of the MutaProp.
- **`MP_CLASS_TYPE` constant defining** the MutaProp class for GUI use (the utilization of this parameter is GUI-implementation-dependent).

MP_CLASS_TYPE = 'abstract'

MP_DEFINITION_ORDER = 'deford'

MP_DOC = 'doc'

MP_HIERARCHY = 'hierarchy'

MP_ID = 'id'

MP_NAME = 'name'

MP_PRIORITY = 'priority'

MP_TYPE = 'type'

MP_VIEW = 'view'

definition_order

display_name

display_priority

hierarchy

prop_id

to_dict(*obj=None*)

view

class mutaprops.mutaprops.**MutaPropClass**

Bases: object

MP_CLASS_ID = 'class_id'

MP_DOC = 'doc'

MP_GUI_ID = 'gui_id'

MP_GUI_MAJOR_VERSION = 'gui_major_version'

MP_GUI_MINOR_VERSION = 'gui_minor_version'

MP_NAME = 'name'

MP_OBJ_ID = 'obj_id'

MP_PROPS = 'props'

classmethod get_class_name()

classmethod get_gui_id()

classmethod get_gui_version()

is_muta_ready()

classmethod muta_attr(*attr*)

muta_id

muta_init(*object_id*, *change_callback=None*)

muta_unregister()

props

to_dict()

update_props(*change_callback=None*)

Because this is potentially heavy operation and property definitions are not likely to be changed during objects lifetime, it's easier to cache it.

class mutaprops.mutaprops.**MutaProperty**(*pid*, *display_name*, *value_type*, ***kwargs*)

Bases: *mutaprops.mutaprops.MutaProp*

Emulate PyProperty_Type() in Objects/descrobject.c

MP_CHANGE_CALLBACK = 'change_callback'

MP_CLASS_TYPE = 'property'

MP_FDEL = 'fdel'

MP_FGET = 'fget'

MP_FSET = 'fset'

MP_MAXVAL = 'max_val'

MP_MINVAL = 'min_val'

MP_READ_ONLY = 'read_only'

MP_SELECT = 'select'

MP_STEP = 'step'

MP_TOGGLE = 'toggle'

MP_VALUE = 'value'

MP_VALUE_TYPE = 'value_type'

deleter(*fdel*)

getter(*fget*)

Decorator function for constructing MutaProperty on getter function. Takes all `kwargs` from `__init__()`

is_writeable()

Returns true if only getter is defined.

Warning: doesn't reflect the read_only kwarg!

muta_set(*obj*, *value*)

register_change_callback(*callback*)

setter(*func=None*, *min_val=None*, *max_val=None*, *step=None*, *select={}*)

Decorator function usable in two ways:

- decorator without arguments:

```
@some_metaprop.setter
def some_metaprop(self, value):
    pass
```

- decorator with arguments:

```
@some_metaprop.setter(min_value=1.0, max_value=2.0)
def some_metaprop(self, value):
    pass
```

Parameters

- **func** – Is only for internal decorator use, don't use it
- **min_val** – Min. value for numeric types, min. lenght for Strings
- **max_val** – Max. value for numeric types, max. length for Strings
- **step** – Step increment for numeric types
- **select** – Selector object to provide user select. A selector can be either a dict, or list of (label, value) tuples, or another MutaProperty or MutaSource which provides dict or list of tuples. In such case, the selector list will be updated during runtime.

Returns MutaProp object

to_dict(*obj=None*)

value_type

class mutaprops.mutaprops.**MutaSource**(*pid*, *display_name*, *value_type*, ***kwargs*)

Bases: *mutaprops.mutaprops.MutaProperty*

MutaSource is generalized MutaProperty, which is not visible in the GUI, however it's changes are reflected in the GUI. MutaSource does not need to define display name and value type - any serializable type goes. MutaSource can also be a class-property. MutaSources cannot be directly changed from the GUI layer, however they can be changed indirectly from the model/MutaObject itself.

Implementation Note

In theory *MutaProperty* should be child of *MutaSource*, in practice the differences are of such character it doesn't make it more convenient to implement *MutaSource* as child of *MutaProperty*.

MP_CLASS_SCOPE = 'class_scope'

MP_CLASS_TYPE = 'source'

MP_OWNER_CLASS = 'owner_class'

class_scoped

Returns True if the MutaSource is classproperty.

muta_set(*obj*, *value*)

owner_class

In case of classproperty source, returns the owner class.

classmethod serialize(*value*)

set_owner_class(*defining_class*)

setter(*func*)

Get setter method. :param func: :return: MutaSource object.

setter_classproperty(*func*)

to_dict(*obj=None*)

class mutaprops.mutaprops.**MutaTypes**
Bases: enum.Enum

Representation of allowed MutaProperty types.

BOOL = 3

HTML = 4

INT = 1

REAL = 2

STRING = 0

4.1.5 mutaprops.utils module

class mutaprops.utils.**BiDict**(**args*, ***kwargs*)
Bases: collections.OrderedDict

Bidirectional dictionary for handling both getters and setters of MutaProperties with selects. Copied from http://stackoverflow.com/a/21894086 and adopted for Python3.

get_map_list()

exception mutaprops.utils.**MutaPropError**
Bases: Exception

mutaprops.utils.**rest_to_html**(*docstring*)
Converts reSTructured text from docstrings to HTML.

As it uses quite strange docutils implementations, it adds some unnecessary clutter to the HTML etc.

4.1.6 Module contents

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/calcite/mutaprops/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

mutaprops could always use more documentation, whether as part of the official mutaprops docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/calcite/mutaprops/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *mutaprops* for local development.

1. Fork the *mutaprops* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mutaprops.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mutaprops
$ cd mutaprops/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. If you plan to work on the Java-Script part (UI front-end), you need to setup your npm environment as well. If you don't have already, you need to get and install NPM. Once installed, in the `mutaprops/web_ui` directory do:

```
# install dependencies
npm install

# serve with hot reload at localhost:8080
npm run dev
```

   When using the hot reload dev server, the UI manager server (the python part) must be running on the port `9000`. If you need other port numbers, adjust the `mutaprops/web_ui/webpack.config.js` file.

6. When you're done making changes, check that your changes pass flake9 and the tests, including testing other Python versions with tox:

```
$ flake8 mutaprops tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

7. As a last step, you should build the front-end minified distribution file:

```
# build for production with minification
npm run build
```

Also, if you made any changes in the `mutaprops/web_ui/webpack.config.js` file, don't include them in the commit.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4, 3.5 and 3.6. Check https://travis-ci.org/calcite/mutaprops/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_mutaprops
```

CHAPTER 6

History

6.1 Latest releases

6.1.1 0.6.6 (2019-01-31)

- Updated sockjs dependency.

6.1.2 0.6.5 (2018-08-20)

- Updated dependencies. (Still not newest, as it is required for Mutaprops to run on Python 3.4)

6.1.3 0.6.4 (2017-11-07)

- Fixed chardet dependency.

6.1.4 0.6.3 (2017-10-16)

- Fixed bug with step setting.

6.1.5 0.6.0 (2017-08-30)

- Added css separation
- Added documentation
- Minor bug fixes

6.2 Older releases

6.2.1 0.5.7 (2017-08-25)

Added the forgoten JS build...

6.2.2 0.5.6 (2017-08-25)

Fixed various UI bugs (read-only settings, responsive design, title). Actions now can have read-only setting.

6.2.3 0.5.5 (2017-04-26)

Fixed incompatibility with Python 3.4.2.

6.2.4 0.5.4 (2017-04-25)

Fixed debug print of properties.

6.2.5 0.5.3 (2017-04-21)

Fixed bug with log messages formatting on the Web UI.

6.2.6 0.5.2 (2017-04-20)

Fixed bug with Bool-type props help panels not uncollapsing.

6.2.7 0.5.1 (2017-03-06)

Fixed error message when object was not selected in an one-object list.

6.2.8 0.5.0 (2017-02-15)

- Large internal rework - introduced update-dependencies for values and selected meta-values (selects, minimums, maximums, steps etc).
- Added MutaSources as non-UI MutaProps for supporting internal dependencies
- Added HTML type of value (read-only)
- JS client now works with single state-store (Vuex)
- MutaSelects removed - this functionality is now replaced by more general update-dependencies through MutaSources. This breaks compatibility with 0.4.x

6.2.9 0.4.1 (2016-12-06)

- Fixed bug with displaying first prop in hierarchy panel.

6.2.10 0.4.0 (2016-12-06)

- One level hierarchy (panels) and experimental support of toggle buttons instead of checkboxes.

6.2.11 0.3.0 (2016-11-03)

- Allowed HTML in help blocks
- Allowed local files/local dir

6.2.12 0.2.2 (2016-11-03)

- Fixed path problem on linux

6.2.13 0.2.1 (2016-11-03)

- Added ALPS logo

6.2.14 0.2.0 (2016-11-03)

- HTTP manager chaining.
- UI bugfixes.

6.2.15 0.1.0 (2016-11-03)

- First (internal) release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

M

N

O

P

R

S

T

U

V

W