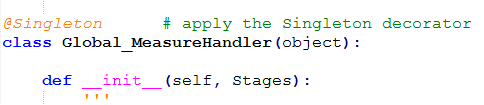
**Measure Handler**

# Overview

The measure handler is the entity that will control and distribute access of the measurement instruments, such as dc multi-meters or optical power-meters, to the procedures or objects that require them. The handler takes away from the specific functions the responsibility to know exactly what instruments they are dealing with. The functions will only need to provide the type of instrument they need and the measure handler will perform all the necessary work to find such an instrument.

# Interface

The measure handler functionality is cleanly exposed by a single class called **Global\_MeasureHandler**. The class utilizes a Singleton scheme so the same object will be accessed from anywhere in the code. The constructor only needs a reference to the Stages dictionary:



Internally the class should utilize a two level table[[1]](#footnote-1) to classify the different instruments. It should look like this:

|  |  |  |  |
| --- | --- | --- | --- |
| inst\_name1 | function | assoc. val | user |
| inst\_name2 | function | assoc. val | user |
| … |  |  |  |
| inst\_nameN | function | assoc. val | user |

|  |  |  |  |
| --- | --- | --- | --- |
| inst\_name1 | function | assoc. val | user |
| inst\_name2 | function | assoc. val | user |
| … |  |  |  |
| inst\_nameN | function | assoc. val | user |

**Figure 1. Internal handler tables**

**Instruments** (main dictionary)

|  |  |
| --- | --- |
| instr\_attr #1 |  |
| instr\_attr #2 |  |
| … | |
| instr\_attr #N |  |

The core of the measure handler’s functionality lies on the nature of the attribute type in the main dictionary. In the current implementation this attribute is the measurement type the instrument performs (ex. DC, RF, optical). The attribute type could as well be the kind of function that the specific interface of the instrument exposes (ex. Does it implement get\_current() ?). The final entries are triplets (function, threshold, user). What the user of the Global\_MeasureHandler is interested in is the measurement function and any associated values that help in the tests (ex. the power threshold for optical coupling).

A description of the class’s methods is provided below:

## \_\_init\_\_(self, Stages)

The constructor will simply get a reference to the Stages dictionary. This is how it gains access to the actual instrument objects. In the initialization stage of the prober software (ProberControl.py), all machines specified in the ProberConfig.conf file are initialized and passed in the dictionary Stages. Since all the functions after that use the equipment through that dictionary it acts as a synchronization point and thus we don’t want the measure\_handler to have a separate copy of that list. It should be examined, though, whether the measurement instruments should never make it into Stages, but only be kept in Global\_MeasureHandler.

## insert\_instr(self, stage\_name, instr\_attr)

This is the function that adds an instrument to the measure\_handler. In the current implementation we use the stage\_name as the primary id of the instrument, since we have the Stages reference from where we get the underlying object, to access its functions, attributes, etc. The instr\_attr argument is the attribute that will determine instrument’s place in the data structure. For the multilevel table and the complex classification extensions discussed in a comment earlier, the instr\_attr argument could be a list of such attributes.

## get\_instr(self, instr\_attr, pair=False)

This is one of the function that is used by the *testing* procedures to get an instrument measurement function form the handler. The user specifies the attribute they want the instrument to have, and the handler will return the first such available instrument-function. If the pair optional argument is set True, then the function returns a tuple composed of the measurement function and the associated value(s).

## get\_instr\_by\_name(self, instr\_name, pair=False)

In the case where a very specific test function requires a specific instrument, the latter can get requested by name from the handler. If that instrument is available, the method will return the wanted function (and value(s), as specified by the pair parameter, see get\_instr). Otherwise, there are two ways to go: We either find an available instrument that shares the same attributes with the one requested, or we signal the caller that the requested instrument is currently unavailable.

# Parallelism

In the method descriptions above the notion of availability is extensively referenced. The general idea behind that is that we would like in the future for the testing procedure to become pipelined so that different tests can be performed at the same time. In that case strict mechanism has to be defined to distribute the instrument resources. The Global\_MeasureHandler is also responsible for that. This is what the *user* field in the table entries are for. Also if tests are to be performed in parallel, the measure\_handler should be completely thread safe, securing the data structure with locks where necessary.

# Side Effects

## Configuration File

Because of the way we want the measure handler to work, we need to get instrument classification information from somewhere (that would go into the *insert\_instr* call). Thus, the ProberConfig.conf file and its parsing mechanism within ProberControl.py should expand to accommodate the insertion of descriptions at least for the instrument entries that are used for measurements (whose names begin with ‘M’).

## Switching

The development of Global\_MeasureHandler should also allow for an efficient way of interacting with the module that will control the large switch which will connect the different components. Some example code could look like:

def some\_test\_function(blah, blah):

g\_mh = Global\_MeasureHandler()

# ... more stuff

fun, thresh = g\_mh.get\_instr(‘DC’, True)

switch.connect(switch\_id\_of(fun), this\_fiber)

# continue with test ...

1. The design of Global\_MeasureHandler should be such that in the future it can easily allow for more complex classification of instruments with multi-level tables. Optimally the underlying data structure of the handler would be able to classify instruments in many possible ways, to accommodate what the *testing* procedures need. [↑](#footnote-ref-1)