bbrecorder Documentation

Release 1.0.1

Laurent Pointal

May 13, 2016

Contents

A Python log handler to keep latest logs in memory and only dump them when needed.

CHAPTER 1

About bbrecorder

author Laurent Pointal <laurent.pointal@limsi.fr> <laurent.pointal@laposte.net>

organization CNRS - LIMSI

version 1.0.2

This system is an intermediate spool keeping a limited set of last log records, and calling other handlers with these log records in case of problem. It allows to not fill log files, to not spend too much time in logging... but to be able to retrieve complete up-to-date information in case of exception raising.

It is composed of a main *BlackBoxHandler* logging handler class, managing the spool, which must be associated to normal logging handlers using its *add_sub_handler()* method. The box size can be modified via *set_size()* method (it is initially set to store 200 log records).

When a situation require the dump of black box logging content, a call to the *crisis()* function (module level, proceed with all black boxes) or to a specific box *crisis_emit()* method make the box(es) dump their stored logs using associated standard log handlers.

Usage example:

```
import bbrecorder
import logging

# Create the logger.
logger = logging.getLogger("bbtest")
logger.setLevel(logging.DEBUG)

# Create BlackBoxHandler and relative logging stuff.
box = bbrecorder.BlackBoxHandler()
box.set_size(10)
fh = logging.FileHandler('_testresult.log')
ch = logging.StreamHandler()
box.add_sub_handler(fh)
box.add_sub_handler(ch)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
box.setFormatter(formatter)

# Associate BlackBoxHandler with our logger.
logger.addHandler(box)

lst = []
```

```
for i in range(1000):
    lst.append(i)
    logger.debug('This %d message should go to the log file with %s', i, lst)
    if len(lst)>=7:
        lst = []

# Here, we should only get the last 10 logs, with the value of lst corresponding to the
# time logs ware created. This is typically called within an exception handler.
bbrecorder.crisis()
```

The module level *full_install()* function setup en environment where it logs Python crashes and allows user to trig black box log emitting via the Ctrl-C key combination and other signals.

Important: For applications which may hardly crash within low level C Python code, inside a compiled extension module, when calling a native function via ctypes... with invalid args, etc. You must look at faulthandler standard module and use it to retrieve tracebacks for threads:

```
import faulthandler
# Create a special file.
outfile = open("python_faulthandler.txt", "w")
# Enable the fault handler for the SIGSEGV, SIGFPE, SIGABRT, SIGBUS and SIGILL
faulthandler.enable(outfile)
```

Because generating logs via a BlackBoxHandler may make them loosed if a hard Python crash occur.

Note that this Python faulthandler tool module appear with Python 3.3, and has not been backported (yet) to Python 2.7.

Note: The module has been tested with Python 3.4 and 3.5, and made compatible with Python 2 and tested on Python 2.7 for version 1.0.2.

---

class bbrecorder.**BlackBoxHandler**(***args*)

Specific logging handler to store in memory a limited (rotation of) logs.

By default the black box handler will keep last INITIAL_BOX_SIZE (200) log records. You can change this value using *set_size()* method.

Once you created a *BlackBoxHandler*, you must associate some logging handlers to it with its *add_sub_handler()* method. Unless you specify a formatter for these sub handlers, they will use same formatter as the black box one.

When you encounter a situation where you want to store/display recent stored log records, you can call general *crisis()* function, to extract logs from all black boxes, or call a specific black box *crisis_emit()* method.

Warning: By default log records msg is immediately formatted with its args, which are then dismissed. Two benefits:

- mutable args are formatted with the value they have at log record creation time.
- args are not keep during all life time of the record in the blackbox.

You may delay this formatting calling *set_dismiss_record_args()* method with False... but care with delayed logs generation consequences vs mutable formatting parameters.

Variables

- **_boxsize** (*int*) – maximum count of last logs stored, default to 200.

- **_dismiss_record_args** (*bool*) – indicator to format log msg with args immediately then dismiss them, default to True.
- **_records** (*list*) – storage of most recent log records.
- **_sub_handlers** (*list*) – handlers to use in case of crisis_emit. All these sub handlers will use the black box filters and formatter, they are directly used to emit the log records.

add_sub_handler(*handler*)
Add a logging handler to be used to transmit / store / display log records in crisis_emit situation.

Parameters **handler** (*logging.Handler*) –

Returns

crisis_emit()
Ensure transmission / storage / display of recent log records via the sub handlers installed by *add_sub_handler()* method calls.

Once done, log records storage is cleared (so logs won't be written multiple times by the same box).

set_dismiss_record_args(*dismiss=True*)
Modify the dismiss record args flag to do immediate log record msg format and dimiss args.

Parameters **dismiss** (*bool*) – new flag value.

set_size(*size*)
Modify the count of log records stored by the black box handler.

Parameters **size** (*int*) – new maximum count of log records to keep.

bbrecorder.**crisis**()
Dump black box log records, called in crisis situation (exception...).

You must call this function when you are interested to emit currently stored log records to normal handlers (file, syslog, etc). The function loop over created *BlackBoxHandler* objects and call their respective *crisis_emit()* methods.

The *crisis_signal()* function provides same service callable as a signal handler .

bbrecorder.**crisis_signal**(*signum*, *stackframe*)
Callback function to be installed as signal handler (see Python signal module).

bbrecorder.**signal_install**(*signum*, *chain=True*)
Install a *crisis_signal()* handler for signum.

Parameters

- **signum** (*int*) – signal number to handle.
- **chain** (*bool*) – flag to require call of existing signal handler after crisis one.

bbrecorder.**full_install**()
Install services to catch / trace hard errors from the Python application.

Intercept keyboard breaks to dump black boxes log records.

b

A

B

C

F

S