Reactor Track Documentation

Release 0.0.1

Martin Gergov

August 22, 2013

CONTENTS

A simple hack that can turn out to be useful someday. (For use with python twisted)

It strives to make twisted debugging easy.

For now it can:

1. track deferred creation
2. track when call/errback is attached to the deferred
3. track when call/errback has executed or failed to execute
4. track when a deferred is chained to another deferred

Documentation index:

CHAPTER

ONE

INTRODUCTION

Note: If you have no knowedge of what twisted is I recommend you start here http://twistedmatrix.com/documents/current/core/howto/index.html

Hello there ! This is the introduction page for Reactor Track - a very new and very experimental project intended to drive away the falsely accusations from Twisted for being hard to understand and debug. Currently a great deal of effort on all abstaction layers is made to make the learning curve smooth. Either way this tool might help you visualize how deferreds work (and other objects coming soon).

Note: You are very welcome to send suggestions for what the tool/s should do more or what it should do better by submitting issues to https://bitbucket.org/marto1/reactortrack/issuse !

Reactor track works as a chain of scripts that need to be executed(or not) so that integration with your source remains easy while allowing it to be used for other stuff too !

So firstly we need to choose which objects we will track. Tracked objects produce events like executing a callback which we could use later. We can look at example.py:

```
from twisted.internet import defer, reactor
import json, sys
from time import sleep

from reactortrack.monitor import JSONMonitor

def simpleCallback(ignored):
    sleep(1)
    return 1

def followup(ignored):
    return 2

def problem(ignored):
    raise Exception("oh no")

def simpleErrback(failure):
    return None

def finished(ignored):
    print "FINISHED!"


monitor = JSONMonitor(True)
```

```
d = defer.Deferred()
d2 = defer.Deferred()
monitor.track(d)
d.addCallback(simpleCallback)
d.addCallbacks(followup)
d.addCallbacks(problem)
d.addErrback(simpleErrback)

d.chainDeferred(d2)
d2.addCallback(simpleCallback)

d.addCallback(finished)

reactor.callLater(0.1, d.callback, None)
reactor.run()

f = open("mon.out", "w")

for line in monitor.buffer:
    f.write("{0}\n".format(line))

f.close()
```

The script does several things:

```
from twisted.internet import defer, reactor
import json, sys
from time import sleep

from reactortrack.monitor import JSONMonitor

def simpleCallback(ignored):
    sleep(1)
    return 1

def followup(ignored):
    return 2

def problem(ignored):
    raise Exception("oh no")

def simpleErrback(failure):
    return None

def finished(ignored):
    print "FINISHED!"
```

Define some functions, some of them intentionally sleep so that we can simulate something that executes longer.

```
monitor = JSONMonitor(True)
d = defer.Deferred()
d2 = defer.Deferred()
monitor.track(d)
d.addCallback(simpleCallback)
d.addCallbacks(followup)
d.addCallbacks(problem)
d.addErrback(simpleErrback)

d.chainDeferred(d2)
```

```
d2.addCallback(simpleCallback)

d.addCallback(finished)
```

Here we have four things to look at:

1. `monitor = JSONMonitor(True)` We create an instance of a so called Monitor(not to be confused with Per Brinch Hansen's monitor) which is an object that takes care of tracking objects. The `True` parameter is telling the monitor to track chained deferreds too so you don't have to tell him explicitly. Another thing to note here is that it will log events in json format in an internal buffer which we read at the final step.
2. `d = defer.Deferred(); d2 = defer.Deferred()` We create several deferreds to track. Nothing special here.
3. `monitor.track(d)` We tell the monitor that it will track this deferred.
4. We add some call and errbacks as well as a chained deferred. These are events which will be logged.

After that we set a callLater for firing d and run the reactor.

The following simply gets every line from the monitor buffer and saves it to a file.

```
f = open("mon.out", "w")

for line in monitor.buffer:
    f.write("{0}\n".format(line))

f.close()
```

And that's it ! Following this idea we can track any deferred we like.

Now just run example.py, wait for it to print FINISHED!, CTRL+C the reactor and it will create the file mon.out .

Next is visualising mon.out.

CHAPTER

TWO

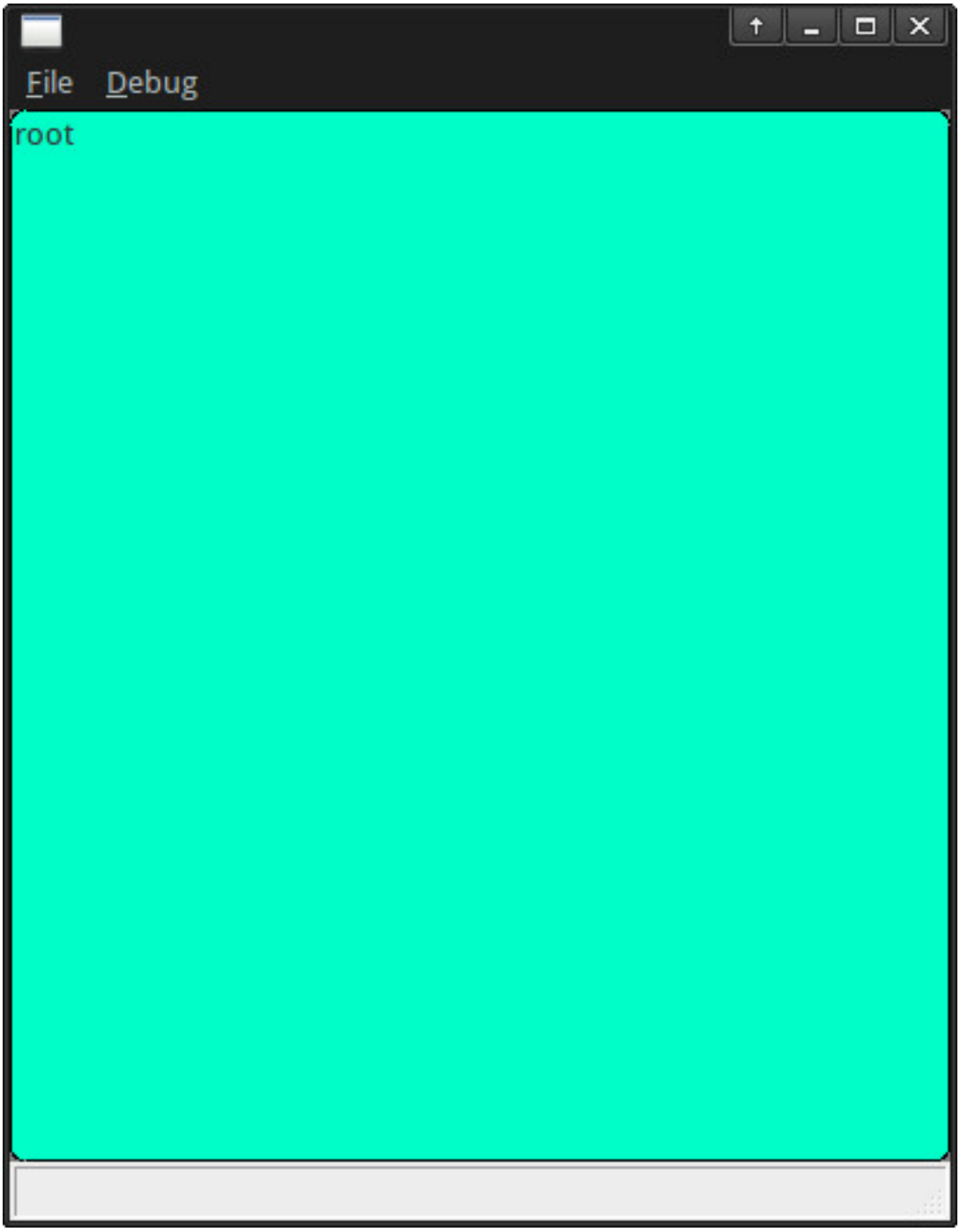
USING MODEL2MAP

You've got your json event log which looks something like:

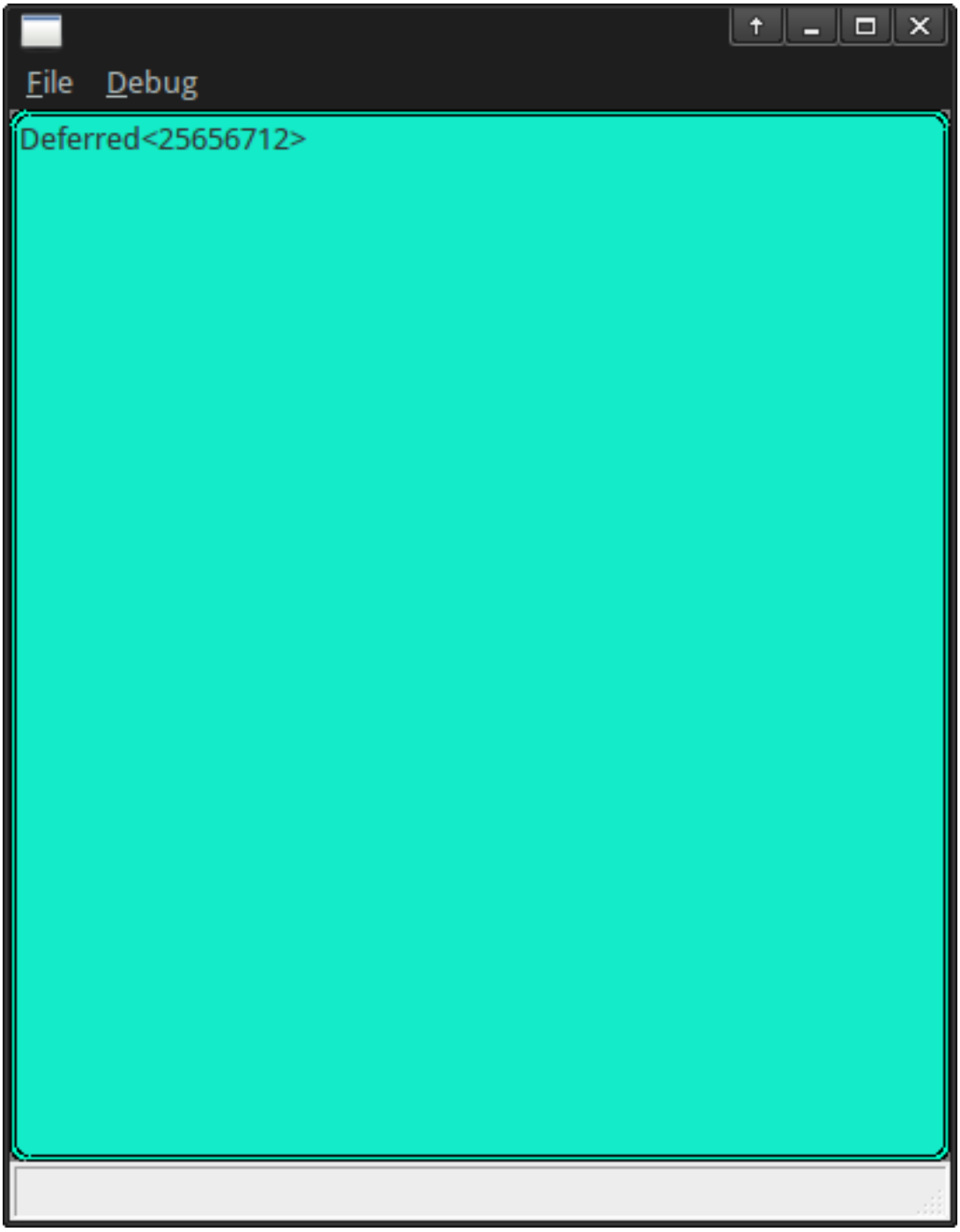
```
{"action": "attached", "deferred": 25657144, "errback": null, "callback": "__main__.simpleCallback"}
{"action": "attached", "deferred": 25656712, "errback": null, "callback": "__main__.finished"}
{"action": "executed", "deferred": 25656712, "time": 1.001, "callback?": true, "callback": "__main__
```

This could be helpful, but does not allow you to look at everything as a whole. model2map is a tool which builds a SquareMap from the deferred's callback chain and visualizes its execution.

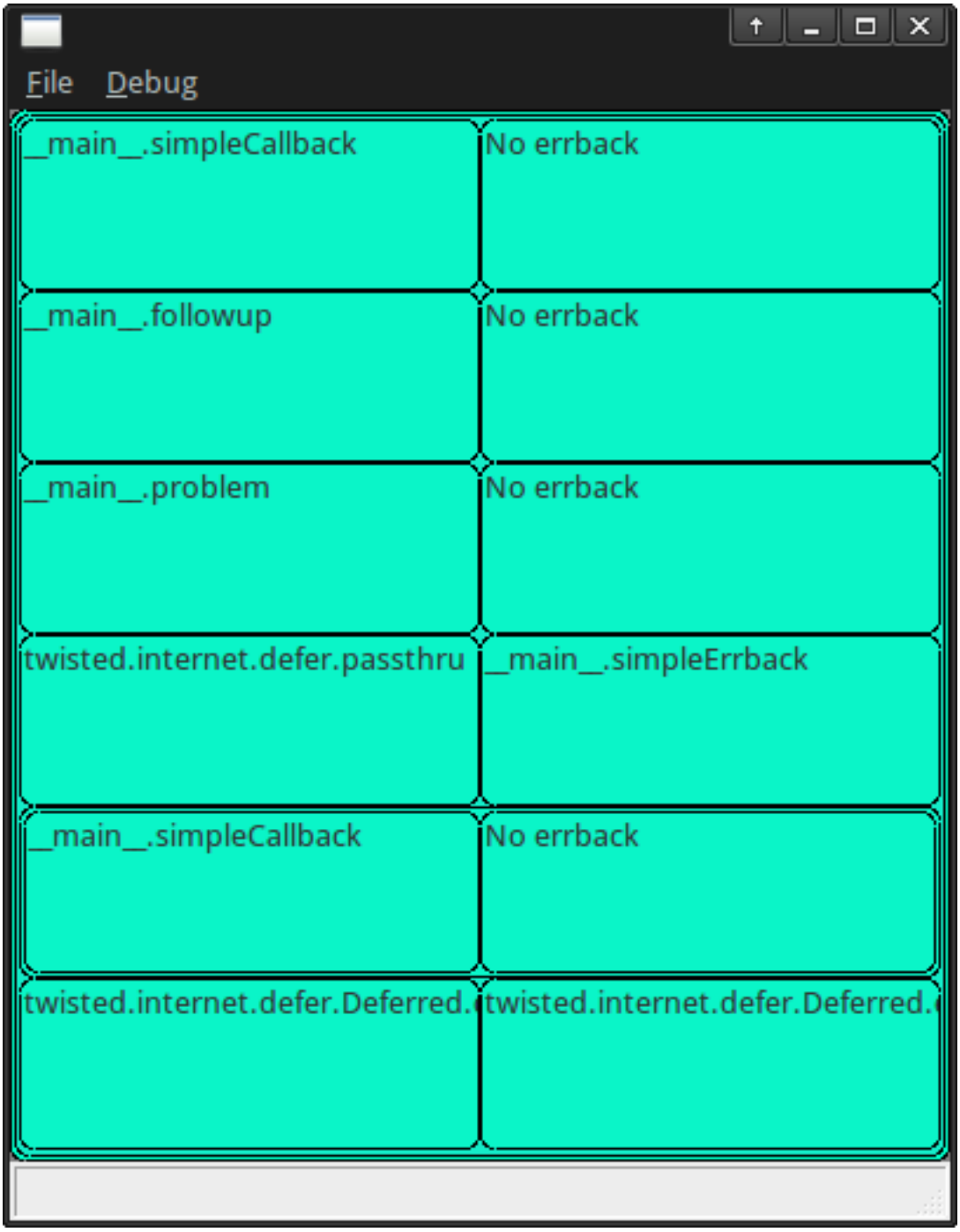
Pipe the file to model2map `cat mon.out | monitor2map`. Something like this appears:



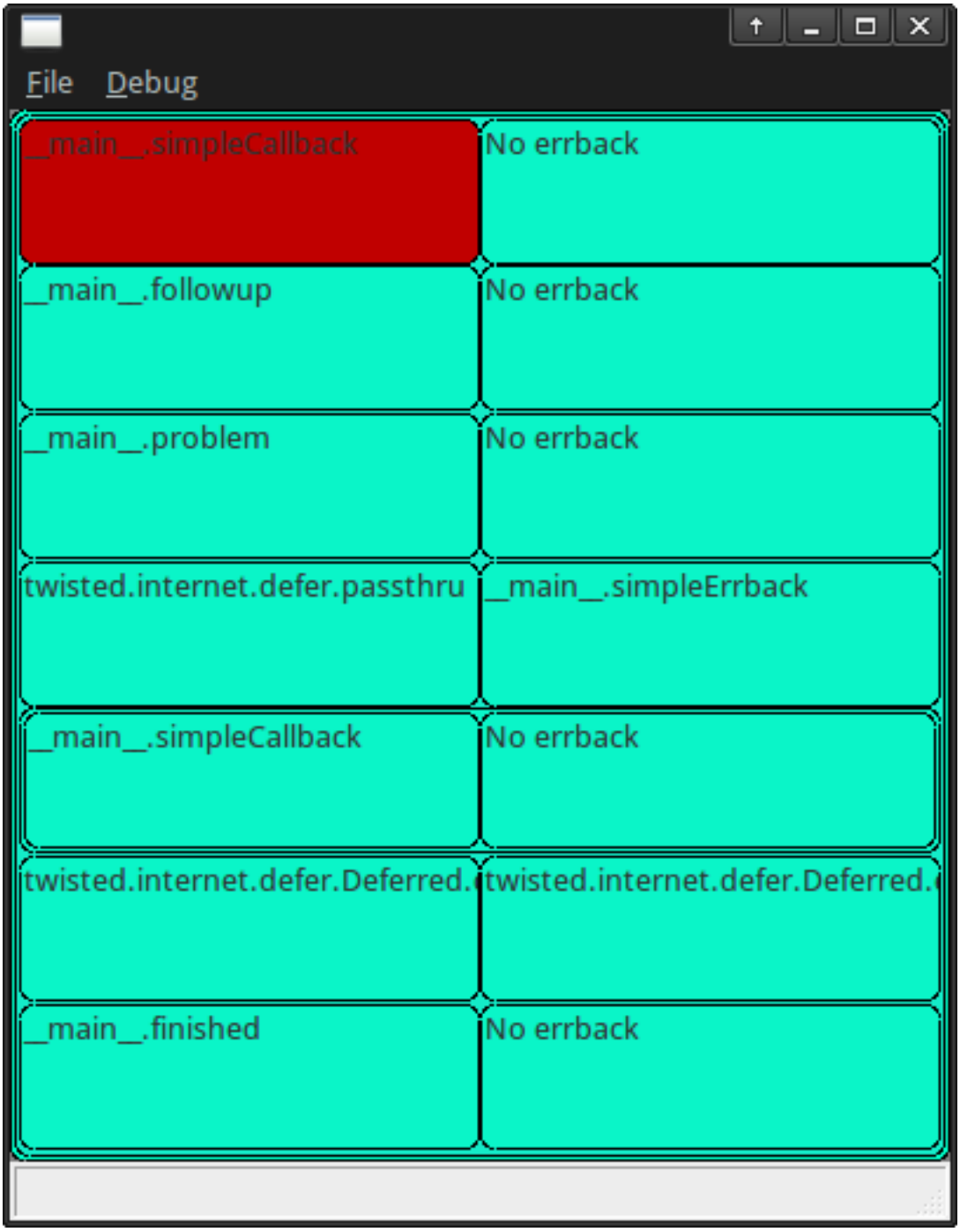
This is the root element. It is just a container for all the tracked objects. You can tell model2map to read the first line from the file from the Debug menu->Next line or CTRL+N.



Do this several times till you get something like:



This is the deferred with everything in example.py added to it (including the deferred that was chained, embedded with its callbacks drawned on a deeper depth). After this the deferred is fired and execution begins:



The tile in red is the callback currently executing. When execution finishes the "cursor" will stay at the last executed callback.

Note: For now this rule also applies to chained deferreds too.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*