Smiley Documentation

Release 0.8.1

Doug Hellmann

Jun 21, 2018

Contents

Smiley is a tool for spying on your Python programs and recording their activities. It can be used for post-mortem debugging, performance analysis, or simply understanding what parts of a complex program are actually used in different code paths.

Contents:

CHAPTER 1

Quick Start

1.1 Installing

Install with `pip`:

```
$ pip install smiley
```

1.2 Using

This quick-start is not a complete reference for the command line program and its options. Use the help subcommand for more details.

In one terminal window, run the `monitor` command:

```
$ smiley monitor
```

In a second terminal window, use smiley to run an application. This example uses `test.py` from the `test_app` directory in the smiley source tree.

```
$ smiley run ./test.py
args: ['./test.py']
input = 10
Leaving c() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Leaving b()
Leaving a()
```

The monitor session will show the execution path and local variables for the app.

```
Starting new run: ./test.py
test.py:   1: import test_funcs
test.py:   1: import test_funcs
test_funcs.py:   1: import sys
```

(continues on next page)

(continued from previous page)

```
test_funcs.py:   1: import sys
test_funcs.py:   3: def gen(m):
test_funcs.py:   8: def c(input):
test_funcs.py:  14: def b(arg):
test_funcs.py:  21: def a():
test_funcs.py:  21: return>>> None
test.py:   3: if __name__ == '__main__':
test.py:   4:     test_funcs.a()
test_funcs.py:  21: def a():
test_funcs.py:  22:     print 'args:', sys.argv
test_funcs.py:  23:     b(2)
test_funcs.py:  14: def b(arg):
                    arg = 2
test_funcs.py:  15:     val = arg * 5
                    arg = 2
test_funcs.py:  16:     c(val)
                    arg = 2
                    val = 10
test_funcs.py:   8: def c(input):
                    input = 10
test_funcs.py:   9:     print 'input =', input
                    input = 10
test_funcs.py:  10:     data = list(gen(input))
                    input = 10
test_funcs.py:   3: def gen(m):
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    m = 10
test_funcs.py:   5:         yield i
                    i = 0
                    m = 10
test_funcs.py:   5: return>>> 0
test_funcs.py:   5:         yield i
                    i = 0
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    i = 0
                    m = 10
test_funcs.py:   5:         yield i
                    i = 1
                    m = 10
test_funcs.py:   5: return>>> 1
test_funcs.py:   5:         yield i
                    i = 1
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    i = 1
                    m = 10
test_funcs.py:   5:         yield i
                    i = 2
                    m = 10
test_funcs.py:   5: return>>> 2
test_funcs.py:   5:         yield i
                    i = 2
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    i = 2
```

(continues on next page)

(continued from previous page)

```
                     m = 10
test_funcs.py:   5:          yield i
                     i = 3
                     m = 10
test_funcs.py:   5: return>>> 3
test_funcs.py:   5:          yield i
                     i = 3
                     m = 10
test_funcs.py:   4:      for i in xrange(m):
                     i = 3
                     m = 10
test_funcs.py:   5:          yield i
                     i = 4
                     m = 10
test_funcs.py:   5: return>>> 4
test_funcs.py:   5:          yield i
                     i = 4
                     m = 10
test_funcs.py:   4:      for i in xrange(m):
                     i = 4
                     m = 10
test_funcs.py:   5:          yield i
                     i = 5
                     m = 10
test_funcs.py:   5: return>>> 5
test_funcs.py:   5:          yield i
                     i = 5
                     m = 10
test_funcs.py:   4:      for i in xrange(m):
                     i = 5
                     m = 10
test_funcs.py:   5:          yield i
                     i = 6
                     m = 10
test_funcs.py:   5: return>>> 6
test_funcs.py:   5:          yield i
                     i = 6
                     m = 10
test_funcs.py:   4:      for i in xrange(m):
                     i = 6
                     m = 10
test_funcs.py:   5:          yield i
                     i = 7
                     m = 10
test_funcs.py:   5: return>>> 7
test_funcs.py:   5:          yield i
                     i = 7
                     m = 10
test_funcs.py:   4:      for i in xrange(m):
                     i = 7
                     m = 10
test_funcs.py:   5:          yield i
                     i = 8
                     m = 10
test_funcs.py:   5: return>>> 8
test_funcs.py:   5:          yield i
                     i = 8
```

(continues on next page)

(continued from previous page)

```
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    i = 8
                    m = 10
test_funcs.py:   5:         yield i
                    i = 9
                    m = 10
test_funcs.py:   5: return>>> 9
test_funcs.py:   5:         yield i
                    i = 9
                    m = 10
test_funcs.py:   4:     for i in xrange(m):
                    i = 9
                    m = 10
test_funcs.py:   4: return>>> None
test_funcs.py:  11:     print 'Leaving c()', data
                    data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
                    input = 10
test_funcs.py:  11: return>>> None
test_funcs.py:  17:     print 'Leaving b()'
                    arg = 2
                    val = 10
test_funcs.py:  18:     return val
                    arg = 2
                    val = 10
test_funcs.py:  18: return>>> 10
test_funcs.py:  24:     print 'Leaving a()'
test_funcs.py:  24: return>>> None
test.py:   4: return>>> None
Finished run
```

1.3 Passing Arguments to Traced Programs

The arguments to `run` are interpreted as a new command to be executed as though it was run directly, but with tracing enabled.

A simple command without options can be run directly:

```
$ smiley run ./test.py
```

If the command takes options, the argument parser for `run` needs to be told to ignore them by using `--` to separate the command sequence from the options for `run`:

```
$ smiley run -- ./test.py -e
```

CHAPTER 2

Command Reference

The main program for Smiley is `smiley`. It includes several sub-commands.

2.1 run

Run an application and trace its execution.

2.2 monitor

Listen for trace data from an application running under the `run` command.

2.3 record

Listen for trace data from an application running under the `run` command and write it to a database for later analysis.

2.4 list

Show the runs previously recorded in the database.

2.5 replay

Given a single run id, dump the data from that run in the same format as the `monitor` command.

2.6 server

Run a web server for browsing the pre-recorded run data collected by the `record` command.

2.7 report

Export a set of HTML files describing the pre-recorded run data collected by the `record` command.

See also:

- sample report output

2.8 stats show

Show the profiling data from a run.

2.9 stats export

Dump the profiling data from a run to a local file.

2.10 help

Get help for the `smiley` command or a subcommand.

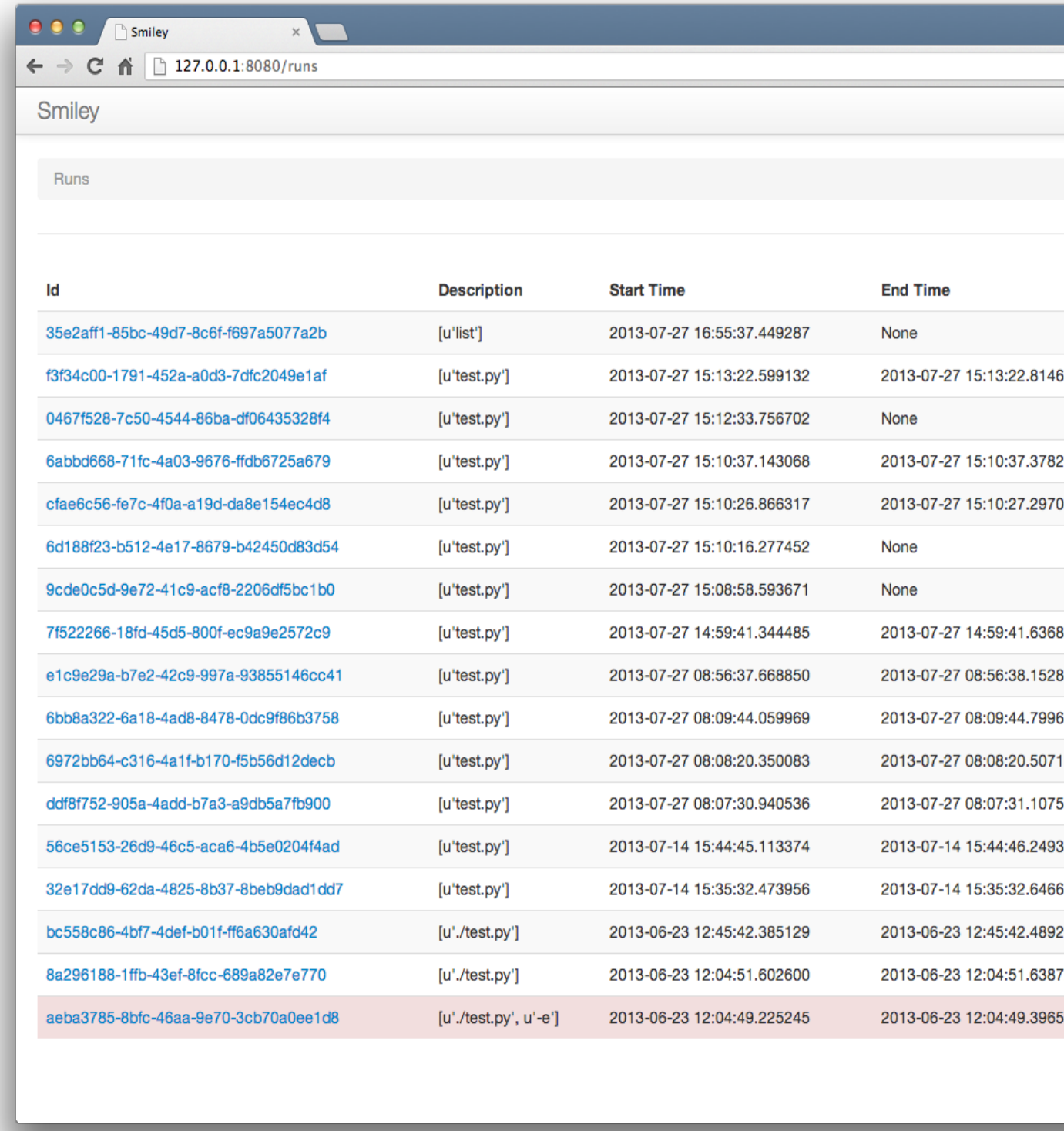
CHAPTER 3

Server Mode

The *server* command starts a web server on a local port to provide a user interface for browsing through the run data captured by *record* and *run*. It connects to the same database, so as new runs are captured they appear in the user interface.

3.1 Run List

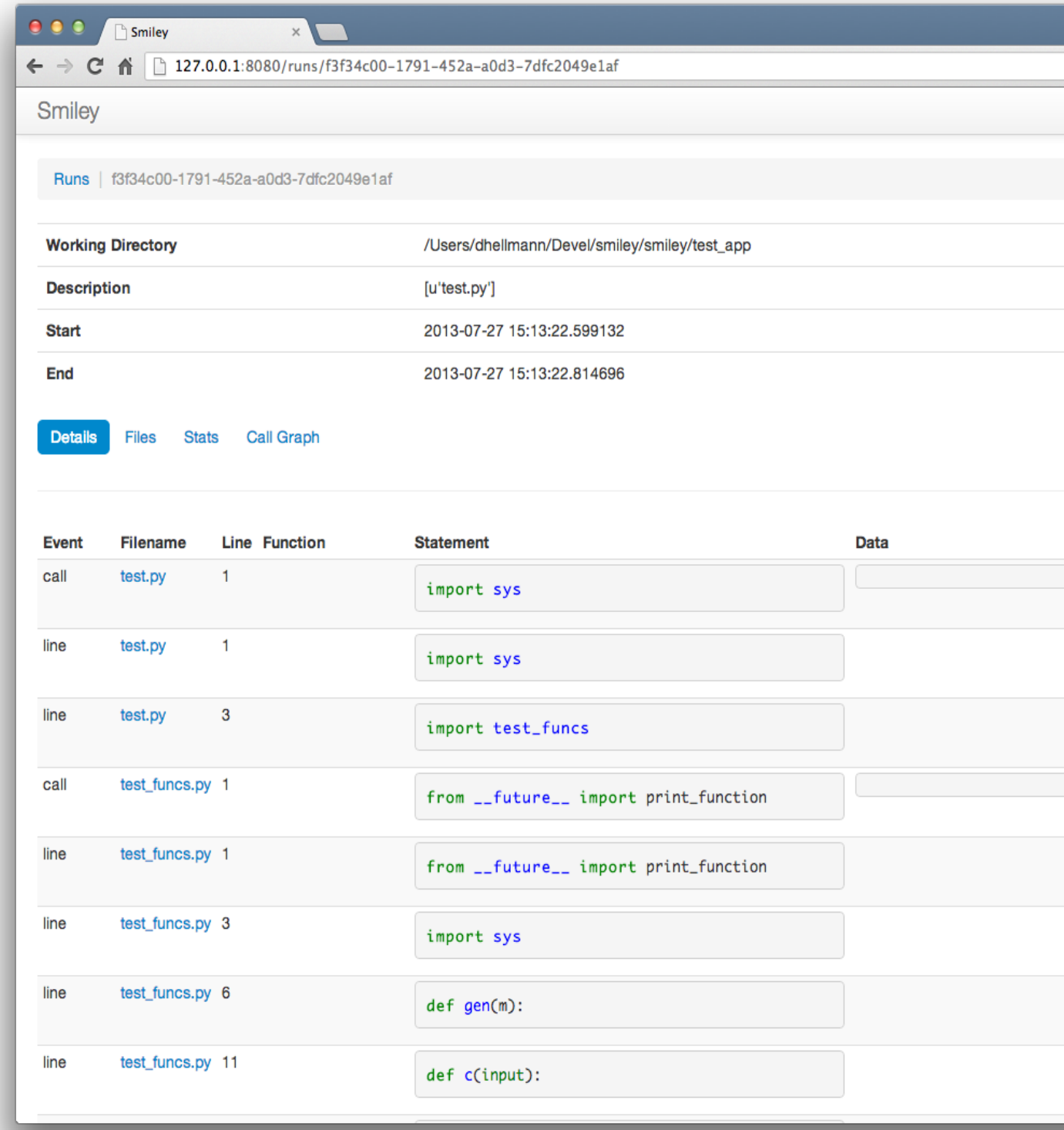
The server listens on `http://127.0.0.1:8080` by default. Visiting that page in a browser causes the server to return a list of the runs found in the database in reverse chronological order. For each run the list shows its id, "description", start and end times, and any final error message.



Id	Description	Start Time	End Time
35e2aff1-85bc-49d7-8c6f-f697a5077a2b	[u'list']	2013-07-27 16:55:37.449287	None
f3f34c00-1791-452a-a0d3-7dfc2049e1af	[u'test.py']	2013-07-27 15:13:22.599132	2013-07-27 15:13:22.8146
0467f528-7c50-4544-86ba-df06435328f4	[u'test.py']	2013-07-27 15:12:33.756702	None
6abbd668-71fc-4a03-9676-ffdb6725a679	[u'test.py']	2013-07-27 15:10:37.143068	2013-07-27 15:10:37.3782
cfae6c56-fe7c-4f0a-a19d-da8e154ec4d8	[u'test.py']	2013-07-27 15:10:26.866317	2013-07-27 15:10:27.2970
6d188f23-b512-4e17-8679-b42450d83d54	[u'test.py']	2013-07-27 15:10:16.277452	None
9cde0c5d-9e72-41c9-acf8-2206df5bc1b0	[u'test.py']	2013-07-27 15:08:58.593671	None
7f522266-18fd-45d5-800f-ec9a9e2572c9	[u'test.py']	2013-07-27 14:59:41.344485	2013-07-27 14:59:41.6368
e1c9e29a-b7e2-42c9-997a-93855146cc41	[u'test.py']	2013-07-27 08:56:37.668850	2013-07-27 08:56:38.1528
6bb8a322-6a18-4ad8-8478-0dc9f86b3758	[u'test.py']	2013-07-27 08:09:44.059969	2013-07-27 08:09:44.7996
6972bb64-c316-4a1f-b170-f5b56d12decb	[u'test.py']	2013-07-27 08:08:20.350083	2013-07-27 08:08:20.5071
ddf8f752-905a-4add-b7a3-a9db5a7fb900	[u'test.py']	2013-07-27 08:07:30.940536	2013-07-27 08:07:31.1075
56ce5153-26d9-46c5-aca6-4b5e0204f4ad	[u'test.py']	2013-07-14 15:44:45.113374	2013-07-14 15:44:46.2493
32e17dd9-62da-4825-8b37-8beb9dad1dd7	[u'test.py']	2013-07-14 15:35:32.473956	2013-07-14 15:35:32.6466
bc558c86-4bf7-4def-b01f-ff6a630afd42	[u'./test.py']	2013-06-23 12:45:42.385129	2013-06-23 12:45:42.4892
8a296188-1ffb-43ef-8fcc-689a82e7e770	[u'./test.py']	2013-06-23 12:04:51.602600	2013-06-23 12:04:51.6387
aeba3785-8bfc-46aa-9e70-3cb70a0ee1d8	[u'./test.py', u'-e']	2013-06-23 12:04:49.225245	2013-06-23 12:04:49.3965

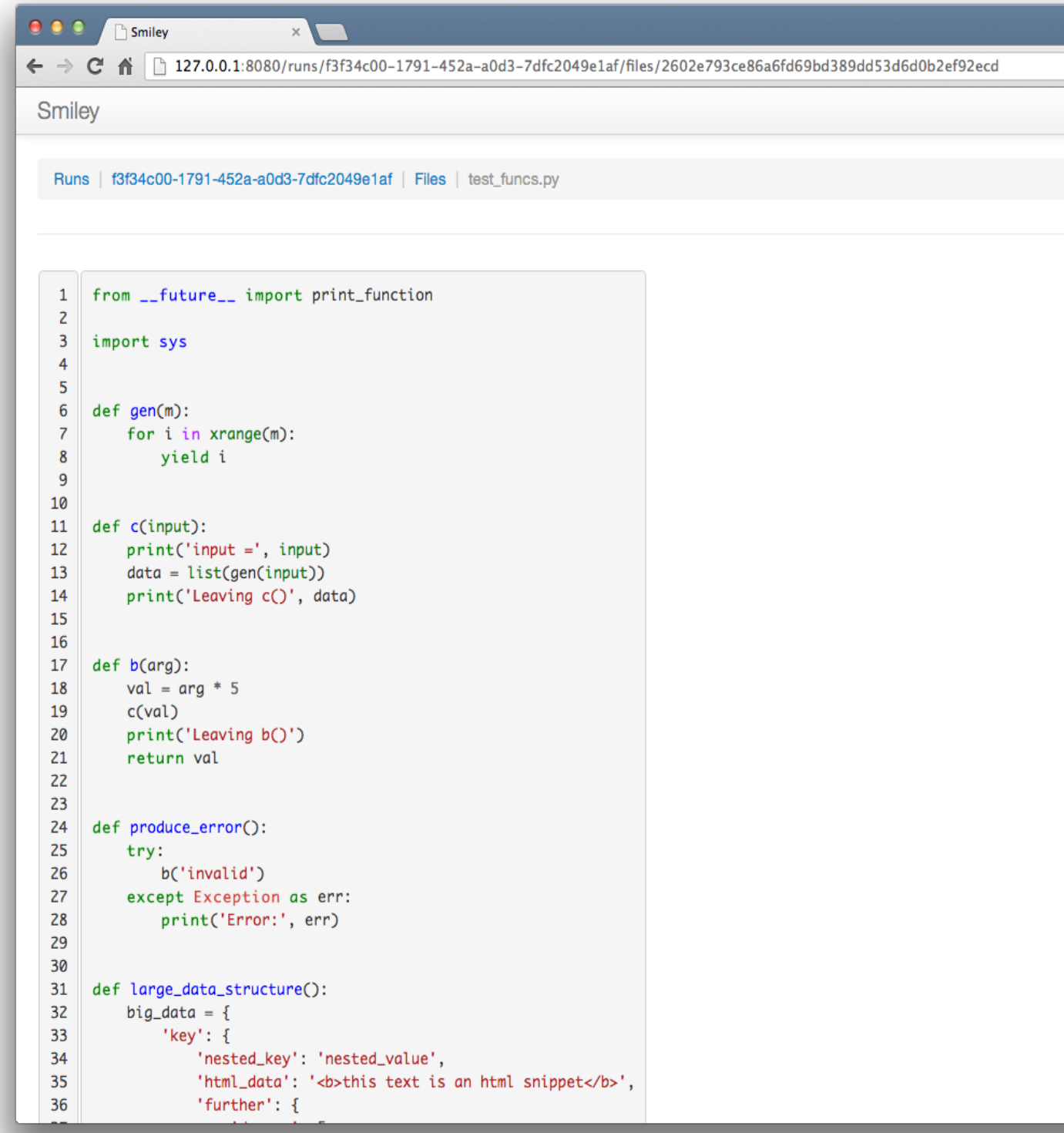
3.2 Run Details

Clicking on one of the run id values opens the detailed information recorded for that run. The details page shows the state of the program line-by-line as it runs, including where the program control is (filename, line number, and source line) as well as local variables and the return values from functions. This is the same information reported by *monitor* and *replay*, in a format that is easier to read.


Smiley
127.0.0.1:8080/runs/f3f34c00-1791-452a-a0d3-7dfc2049e1af
Smiley
Runs | f3f34c00-1791-452a-a0d3-7dfc2049e1af
Working Directory /Users/dhellmann/Devel/smiley/smiley/test_app
Description [u'test.py']
Start 2013-07-27 15:13:22.599132
End 2013-07-27 15:13:22.814696
Details Files Stats Call Graph
Event Filename Line Function Statement Data
call test.py 1 import sys
line test.py 1 import sys
line test.py 3 import test_funcs
call test_funcs.py 1 from __future__ import print_function
line test_funcs.py 1 from __future__ import print_function
line test_funcs.py 3 import sys
line test_funcs.py 6 def gen(m):
line test_funcs.py 11 def c(input):

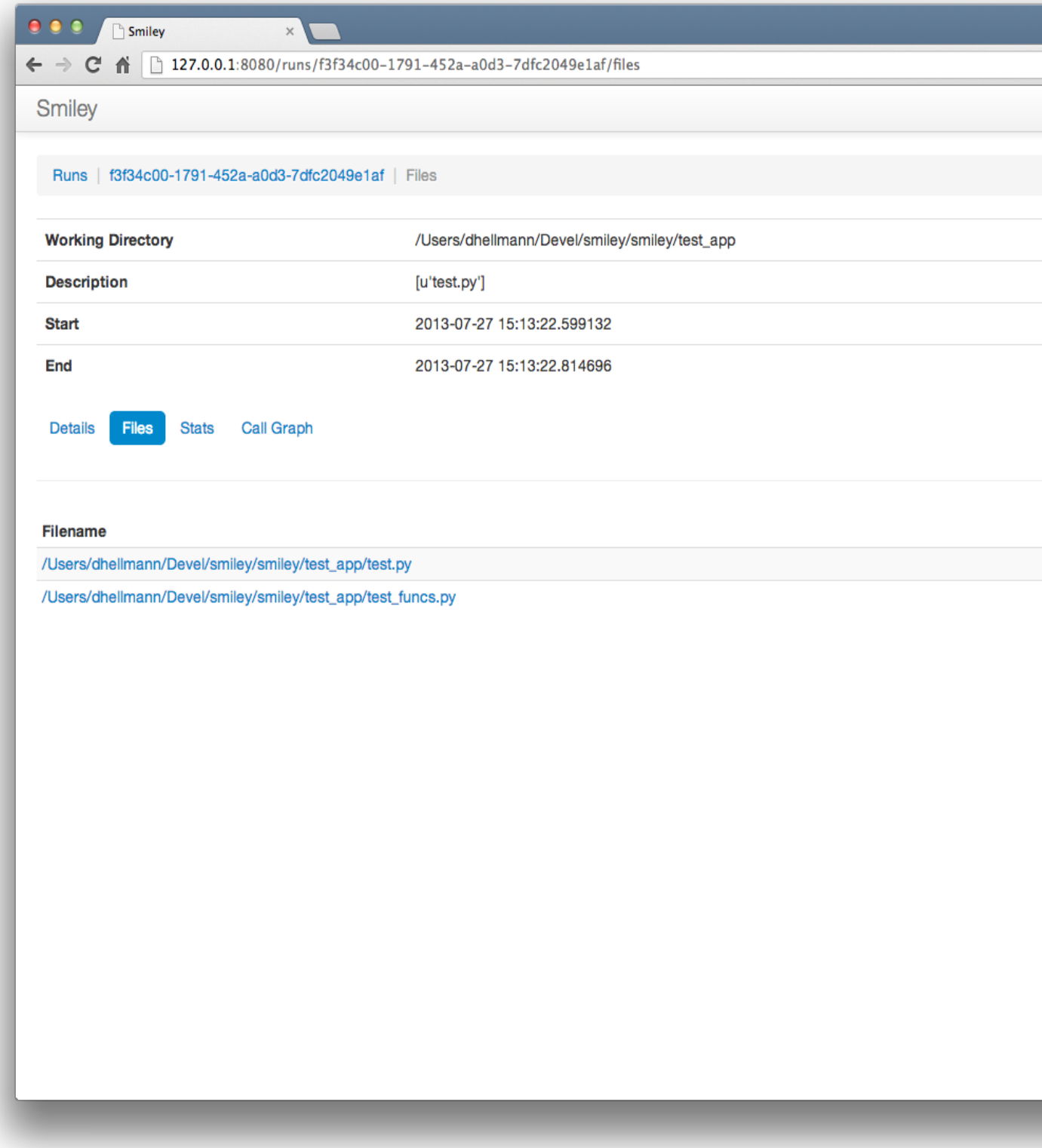
3.3 Source Files

Each of the filenames in the run details view links to a page showing the full source of the Python file as it was at the time of the program's execution.


Smiley
127.0.0.1:8080/runs/f3f34c00-1791-452a-a0d3-7dfc2049e1af/files/2602e793ce86a6fd69bd389dd53d6d0b2ef92ecd
Smiley
Runs | f3f34c00-1791-452a-a0d3-7dfc2049e1af | Files | test_funcs.py
from __future__ import print_function
import sys
def gen(m):
for i in xrange(m):
yield i
def c(input):
print('input =', input)
data = list(gen(input))
print('Leaving c()', data)
def b(arg):
val = arg * 5
c(val)
print('Leaving b()')
return val
def produce_error():
try:
b('invalid')
except Exception as err:
print('Error:', err)
def large_data_structure():
big_data = {
'key': {
'nested_key': 'nested_value',
'html_data': 'this text is an html snippet',
'further': {

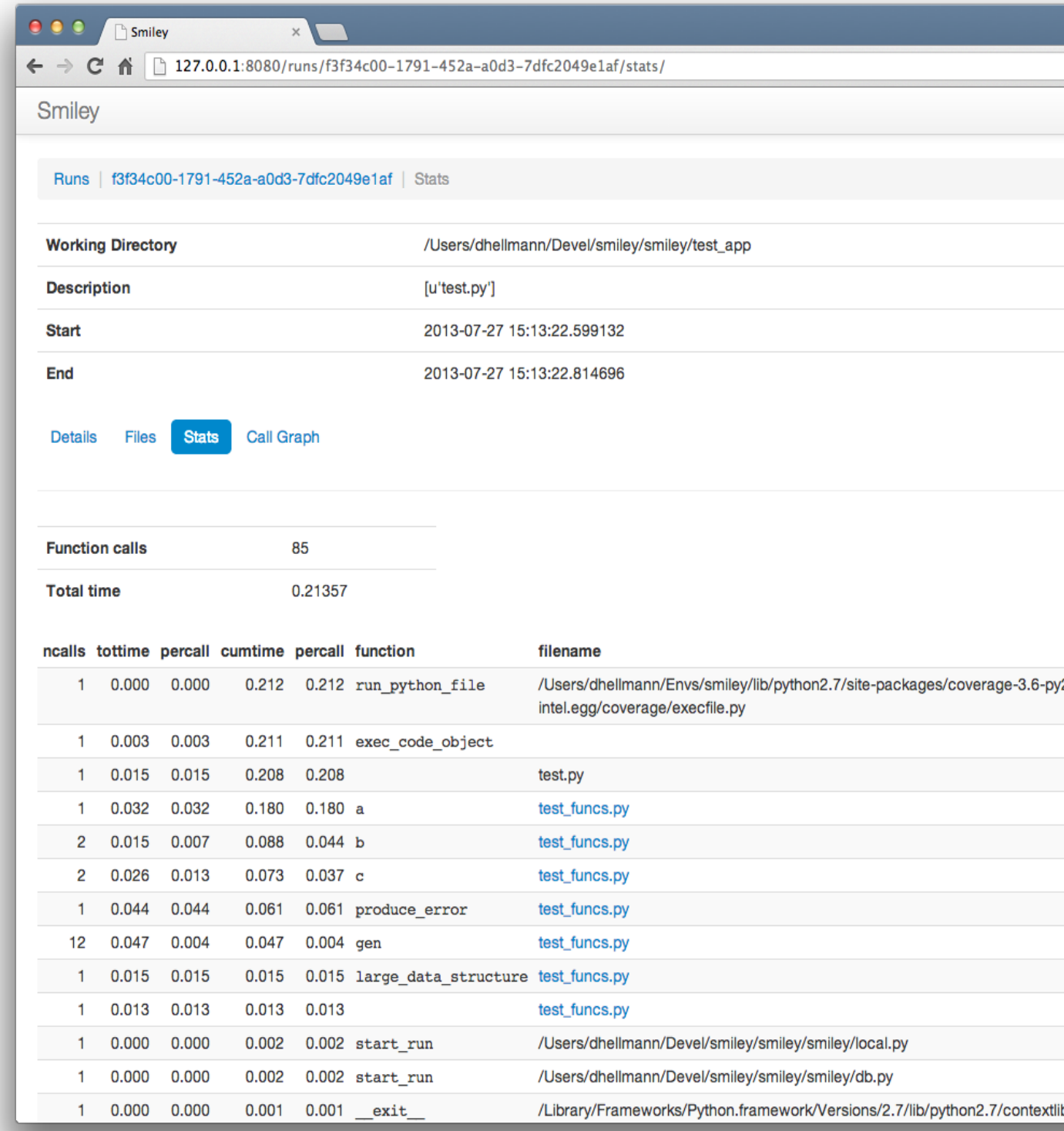
3.4 File List

For an application with many source files, it may be more convenient to examine the source by navigating to the file list view and choosing the file from the list.


Smiley
127.0.0.1:8080/runs/f3f34c00-1791-452a-a0d3-7dfc2049e1af/files
Smiley
Runs | f3f34c00-1791-452a-a0d3-7dfc2049e1af | Files
Working Directory
/Users/dhellmann/Devel/smiley/smiley/test_app
Description
[u'test.py']
Start
2013-07-27 15:13:22.599132
End
2013-07-27 15:13:22.814696
Details
Files
Stats
Call Graph
Filename
/Users/dhellmann/Devel/smiley/smiley/test_app/test.py
/Users/dhellmann/Devel/smiley/smiley/test_app/test_funcs.py

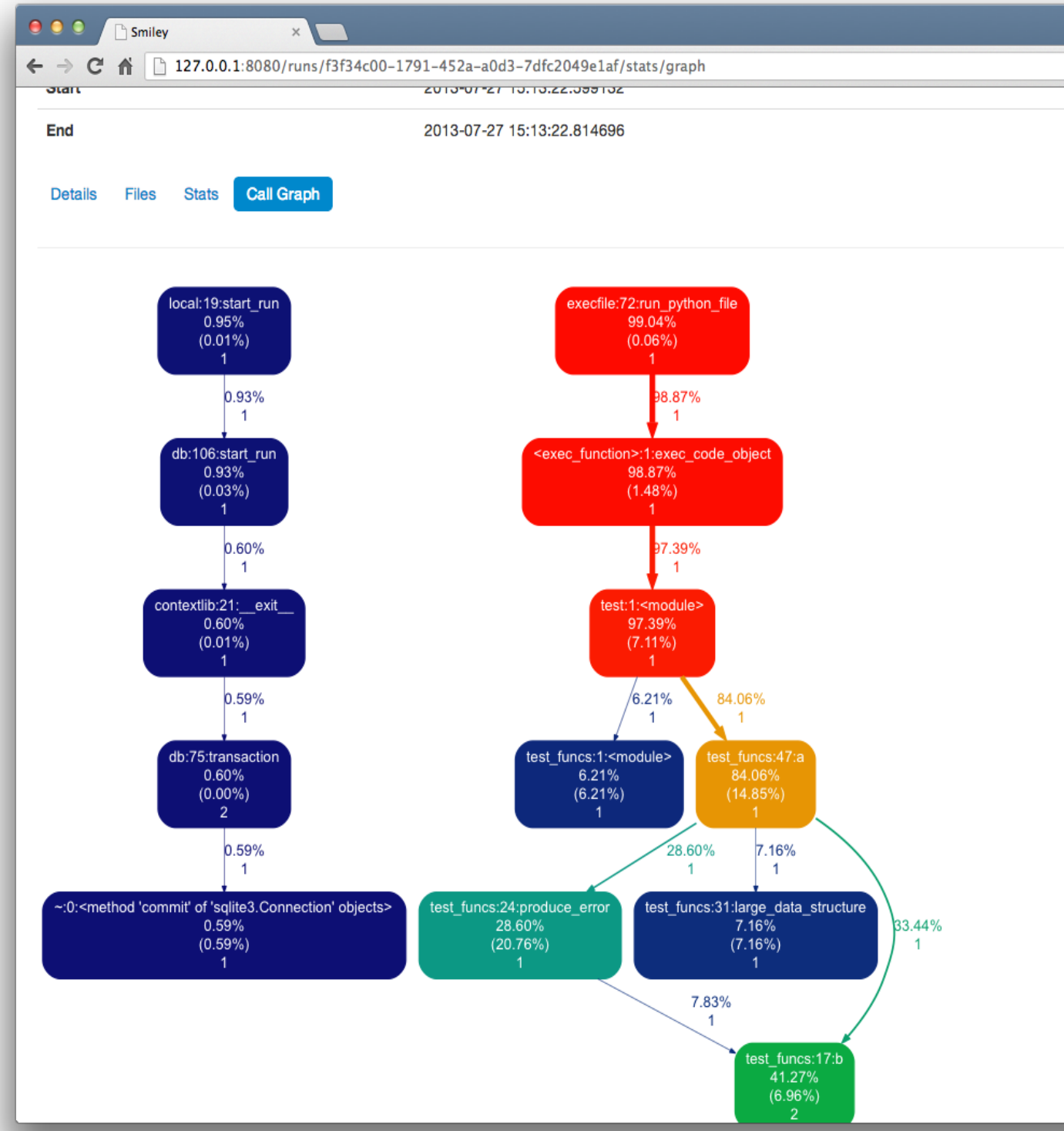
3.5 Profiler Statistics

The stats view shows the profiler output for the run, sorted by cumulative time. As with the run details, each file name links to the full source for the module.


Smiley
127.0.0.1:8080/runs/f3f34c00-1791-452a-a0d3-7dfc2049e1af/stats/
Smiley
Runs | f3f34c00-1791-452a-a0d3-7dfc2049e1af | Stats
Working Directory /Users/dhellmann/Devel/smiley/smiley/test_app
Description [u'test.py']
Start 2013-07-27 15:13:22.599132
End 2013-07-27 15:13:22.814696
Details Files Stats Call Graph
Function calls 85
Total time 0.21357
ncalls tottime percall cumtime percall function filename
1 0.000 0.000 0.212 0.212 run_python_file /Users/dhellmann/Envs/smiley/lib/python2.7/site-packages/coverage-3.6-py2 intel.egg/coverage/execfile.py
1 0.003 0.003 0.211 0.211 exec_code_object
1 0.015 0.015 0.208 0.208 test.py
1 0.032 0.032 0.180 0.180 a test_funcs.py
2 0.015 0.007 0.088 0.044 b test_funcs.py
2 0.026 0.013 0.073 0.037 c test_funcs.py
1 0.044 0.044 0.061 0.061 produce_error test_funcs.py
12 0.047 0.004 0.047 0.004 gen test_funcs.py
1 0.015 0.015 0.015 0.015 large_data_structure test_funcs.py
1 0.013 0.013 0.013 0.013 test_funcs.py
1 0.000 0.000 0.002 0.002 start_run /Users/dhellmann/Devel/smiley/smiley/smiley/local.py
1 0.000 0.000 0.002 0.002 start_run /Users/dhellmann/Devel/smiley/smiley/smiley/db.py
1 0.000 0.000 0.001 0.001 __exit__ /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/contextlib

3.6 Call Graph

The call graph view uses gprof2dot and graphviz to produce a tree diagram showing how much time is used in different parts of the program, to make it easier to focus on the areas that use the most time.



Note: In order for this page to work, you must have the `dot` command installed. Installing smiley should install

`gprof2dot` automatically.

CHAPTER 4

Frequently Asked Questions

4.1 What's with the name?

George Smiley is a character in popular spy novels by John LeCarre.

CHAPTER 5

Release History

5.1 dev

- Add *report* to produce static HTML output for a run.
- Fix a problem with the "ignore" path management under virtualenv.

5.2 0.6

- Update the web view to only show changes in variables. The calculation of changes is very rough, and just compares the current set of variables to the previous set, which might be in a completely unrelated scope.
- Update the web view to show consecutive lines executed together as a single block. A new block is started for each call into a function or when the value of a previously-seen local variable changes.
- Update the web view to show comments near the source line being executed as further context.
- Simplify calculation of local variable changes.
- Tighten up the run view output to allow for wider lines and reduce clutter.
- Make the tests pass under python 3.3. Still not doing any live testing with python3 apps, but this is a start.
- Add an option to *run* to include modules from the standard library. This is disabled by default.
- Add an option to *run* to include modules from the `site-packages` directory (for third-party installed modules). This is enabled by default.
- Add an option to *run* to include a specific package in the trace output by name on the command line.
- Updated to Bootstrap 3 CSS framework.
- Add pagination support to the detailed trace report.

5.3 0.5

- Add a call graph image, built with gprof2dot and graphviz.
- Add *Server Mode* documentation.
- Clean up template implementations.
- Clean up navigation and breadcrumbs.

5.4 0.4

- Collect profiling data along with the trace data.
- Add *stats show* command.
- Add *stats export* command.

5.5 0.3

- Add *record* command.
- Add *list* command.
- Add web ui and *server* command
- Add mode option to *run* to allow writing results directly to a database file.

5.6 0.2

Use the script runner code from coverage instead of reinventing it.

5.7 0.1

First public release. Includes basic functionality of runner and monitor.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search