zope.event Documentation

Release 4.2.1.dev0

Zope Foundation and Contributors

April 20, 2016

Contents

This package provides a simple event system on which application-specific event systems can be built.

Application code can generate events without being concerned about the event-processing frameworks that might handle the events.

Events are objects that represent something happening in a system. They are used to extend processing by providing processing plug points.

Contents:

CHAPTER 1

Using zope.event

The zope.event package has a list of subscribers. Application code can manage subscriptions by manipulating this list. For the examples here, we'll save the current contents away and empty the list. We'll restore the contents when we're done with our examples.

```
>>> import zope.event
>>> old_subscribers = zope.event.subscribers[:]
>>> del zope.event.subscribers[:]
```

The package provides a notify() function, which is used to notify subscribers that something has happened:

```
>>> class MyEvent:
...     pass

>>> event = MyEvent()
>>> zope.event.notify(event)
```

The notify function is called with a single object, which we call an event. Any object will do:

```
>>> zope.event.notify(None)
>>> zope.event.notify(42)
```

An extremely trivial subscription mechanism is provided. Subscribers are simply callback functions:

```
>>> def f(event):
...     print 'got:', event
```

that are put into the subscriptions list:

```
>>> zope.event.subscribers.append(f)

>>> zope.event.notify(42)
got: 42

>>> def g(event):
...     print 'also got:', event

>>> zope.event.subscribers.append(g)

>>> zope.event.notify(42)
got: 42
also got: 42
```

To unsubscribe, simply remove a subscriber from the list:

```
>>> zope.event.subscribers.remove(f)
>>> zope.event.notify(42)
also got: 42
```

Generally, application frameworks will provide more sophisticated subscription mechanisms that build on this simple mechanism. The frameworks will install subscribers that then dispatch to other subscribers based on event types or data.

We're done, so we'll restore the subscribers:

```
>>> zope.event.subscribers[:] = old_subscribers
```

CHAPTER 2

Theory of Operation

Note: This section explains both why an application or framework might publish events, and various ways the integrator might configure the subscribers to achieve different goals.

2.1 Outline

- Events as decoupling mechanism.
  - Injecting policy into reusable applications.
  - Extending frameworks.
- Event dispatch strategies
  - Type-based dispatch, as used in ZCA
  - Attribute-based / key-based dispatch

CHAPTER 3

zope.event API Reference

The package exports the following API symbols.

3.1 Data

zope.event.**subscribers** = []
: Applications may register for notification of events by appending a callable to the `subscribers` list.

  Each subscriber takes a single argument, which is the event object being published.

  Exceptions raised by subscribers will be propagated.

3.2 Functions

zope.event.**notify**(*event*)
: Notify all subscribers of `event`.

CHAPTER 4

Class-based event handlers

A light-weight event-handler framework based on event classes is provided by the `zope.event.classhandler` module.

Handlers are registered for event classes:

```
>>> import zope.event.classhandler
```

```
>>> class MyEvent(object):
...     def __repr__(self):
...         return self.__class__.__name__
```

```
>>> def handler1(event):
...     print("handler1 %r" % event)
```

```
>>> zope.event.classhandler.handler(MyEvent, handler1)
```

Descriptor syntax:

```
>>> @zope.event.classhandler.handler(MyEvent)
... def handler2(event):
...     print("handler2 %r" % event)
```

```
>>> class MySubEvent(MyEvent):
...     pass
```

```
>>> @zope.event.classhandler.handler(MySubEvent)
... def handler3(event):
...     print("handler3 %r" % event)
```

Subscribers are called in class method-resolution order, so only new-style event classes are supported, and then by order of registry.

```
>>> import zope.event
>>> zope.event.notify(MySubEvent())
handler3 MySubEvent
handler1 MySubEvent
handler2 MySubEvent
```

CHAPTER 5

Hacking on `zope.event`

5.1 Getting the Code

The main repository for `zope.event` is in the Zope Foundation Github repository:

https://github.com/zopefoundation/zope.event

You can get a read-only checkout from there:

```
$ git clone https://github.com/zopefoundation/zope.event.git
```

or fork it and get a writeable checkout of your fork:

```
$ git clone git@github.com/jrandom/zope.event.git
```

The project also mirrors the trunk from the Github repository as a Bazaar branch on Launchpad:

https://code.launchpad.net/zope.event

You can branch the trunk from there using Bazaar:

```
$ bzr branch lp:zope.event
```

5.2 Working in a `virtualenv`

5.2.1 Installing

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a virtualenv. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/hack-zope.event
```

Next, get this package registered as a "development egg" in the environment:

```
$ /tmp/hack-zope.event/bin/python setup.py develop
```

5.2.2 Running the tests

Then, you canrun the tests using the build-in `setuptools` testrunner:

```
$ /tmp/hack-zope.event/bin/python setup.py test
running test
...
test_empty (zope.event.tests.Test_notify) ... ok
test_not_empty (zope.event.tests.Test_notify) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

If you have the `nose` package installed in the virtualenv, you can use its testrunner too:

```
$ /tmp/hack-zope.event/bin/easy_install nose
...
$ /tmp/hack-zope.event/bin/python setup.py nosetests
running nosetests
...
----------------------------------------------------------------------
Ran 3 tests in 0.011s

OK
```

or:

```
$ /tmp/hack-zope.event/bin/nosetests
...
----------------------------------------------------------------------
Ran 3 tests in 0.011s

OK
```

If you have the `coverage` pacakge installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/hack-zope.event/bin/easy_install nose coverage
...
$ /tmp/hack-zope.event/bin/python setup.py nosetests \
    --with coverage --cover-package=zope.event
running nosetests
...
Name         Stmts   Exec  Cover   Missing
----------------------------------------
zope.event       5      5   100%
----------------------------------------------------------------------
Ran 3 tests in 0.019s

OK
```

5.2.3 Building the documentation

`zope.event` uses the nifty `Sphinx` documentation system for building its docs. Using the same virtualenv you set up to run the tests, you can build the docs:

```
$ /tmp/hack-zope.event/bin/easy_install Sphinx
...
$ cd docs
$ PATH=/tmp/hack-zope.event/bin:$PATH make html
sphinx-build -b html -d _build/doctrees   . _build/html
```

```
...
build succeeded.

Build finished. The HTML pages are in _build/html.
```

You can also test the code snippets in the documentation:

```
$ PATH=/tmp/hack-zope.event/bin:$PATH make doctest
sphinx-build -b doctest -d _build/doctrees   . _build/doctest
...
running tests...

Document: index
---------------
1 items passed all tests:
  17 tests in default
17 tests in 1 items.
17 passed and 0 failed.
Test passed.

Doctest summary
===============
   17 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the  \
    results in _build/doctest/output.txt.
```

5.3 Using `zc.buildout`

5.3.1 Setting up the buildout

`zope.event` ships with its own `buildout.cfg` file and `bootstrap.py` for setting up a development buildout:

```
$ /path/to/python2.6 bootstrap.py
...
Generated script '.../bin/buildout'
$ bin/buildout
Develop: '/home/jrandom/projects/Zope/zope.event/.'
...
Generated script '.../bin/sphinx-quickstart'.
Generated script '.../bin/sphinx-build'.
```

5.3.2 Running the tests

You can now run the tests:

```
$ bin/test --all
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Ran 2 tests with 0 failures and 0 errors in 0.000 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

5.4 Using tox

5.4.1 Running Tests on Multiple Python Versions

tox is a Python-based test automation tool designed to run tests against multiple Python versions. It creates a virtualenv for each configured version, installs the current package and configured dependencies into each virtualenv, and then runs the configured commands.

zope.event configures the following tox environments via its tox.ini file:

- The py26, py27, py33, py34, and pypy environments builds a virtualenv with the corresponding interpreter, installs zope.event and dependencies, and runs the tests via python setup.py -q test -q.
- The coverage environment builds a virtualenv with python2.6, installs zope.event, installs nose and coverage, and runs nosetests with statement and branch coverage.
- The docs environment builds a virtualenv with python2.6, installs zope.event, installs Sphinx and dependencies, and then builds the docs and exercises the doctest snippets.

This example requires that you have a working python2.6 on your path, as well as installing tox:

```
$ tox -e py26
GLOB sdist-make: .../zope.event/setup.py
py26 sdist-reinst: .../zope.event/.tox/dist/zope.event-4.0.2dev.zip
py26 runtests: commands[0]
...
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
_____________________________________ summary ______________________________________
py26: commands succeeded
congratulations :)
```

Running tox with no arguments runs all the configured environments, including building the docs and testing their snippets:

```
$ tox
GLOB sdist-make: .../zope.event/setup.py
py26 sdist-reinst: .../zope.event/.tox/dist/zope.event-4.0.2dev.zip
py26 runtests: commands[0]
...
Doctest summary
===============
 17 tests
   0 failures in tests
   0 failures in setup code
   0 failures in cleanup code
build succeeded.
_____________________________________ summary ______________________________________
py26: commands succeeded
py27: commands succeeded
py32: commands succeeded
pypy: commands succeeded
coverage: commands succeeded
docs: commands succeeded
congratulations :)
```

5.5 Contributing to zope.event

5.5.1 Submitting a Bug Report

zope.event tracks its bugs on Github:

https://github.com/zopefoundation/zope.event/issues

Please submit bug reports and feature requests there.

5.5.2 Sharing Your Changes

Note: Please ensure that all tests are passing before you submit your code. If possible, your submission should include new tests for new features or bug fixes, although it is possible that you may have tested your new code by updating existing tests.

If have made a change you would like to share, the best route is to fork the Githb repository, check out your fork, make your changes on a branch in your fork, and push it. You can then submit a pull request from your branch:

https://github.com/zopefoundation/zope.event/pulls

If you branched the code from Launchpad using Bazaar, you have another option: you can "push" your branch to Launchpad:

```
$ bzr push lp:~jrandom/zope.event/cool_feature
```

After pushing your branch, you can link it to a bug report on Github, or request that the maintainers merge your branch using the Launchpad "merge request" feature.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

N

S