zope.location Documentation

Release 4.0

Zope Foundation Contributors

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Using `zope.location`

Location

The `Location` base class is a mix-in that defines `__parent__` and `__name__` attributes.

Usage within an Object field:

```
>>> from zope.interface import implementer, Interface
>>> from zope.schema import Object
>>> from zope.schema.fieldproperty import FieldProperty
>>> from zope.location.interfaces import ILocation
>>> from zope.location.location import Location

>>> class IA(Interface):
...     location = Object(schema=ILocation, required=False, default=None)
>>> @implementer(IA)
... class A(object):
...     location = FieldProperty(IA['location'])

>>> a = A()
>>> a.location = Location()

>>> loc = Location(); loc.__name__ = u'foo'
>>> a.location = loc

>>> loc = Location(); loc.__name__ = None
>>> a.location = loc

>>> loc = Location(); loc.__name__ = b'foo'
>>> a.location = loc
Traceback (most recent call last):
...
WrongContainedType: ([WrongType('foo', <type 'unicode'>, '__name__')], 'location')
```

inside()

The `inside` function tells if l1 is inside l2. L1 is inside l2 if l2 is an ancestor of l1.

```
>>> o1 = Location()
>>> o2 = Location(); o2.__parent__ = o1
>>> o3 = Location(); o3.__parent__ = o2
>>> o4 = Location(); o4.__parent__ = o3

>>> from zope.location.location import inside

>>> inside(o1, o1)
True

>>> inside(o2, o1)
True

>>> inside(o3, o1)
True

>>> inside(o4, o1)
True

>>> inside(o1, o4)
False

>>> inside(o1, None)
False
```

LocationProxy

`LocationProxy` is a non-picklable proxy that can be put around objects that don't implement `ILocation`.

```
>>> from zope.location.location import LocationProxy
>>> l = [1, 2, 3]
>>> ILocation.providedBy(l)
False
>>> p = LocationProxy(l, "Dad", "p")
>>> p
[1, 2, 3]
>>> ILocation.providedBy(p)
True
>>> p.__parent__
'Dad'
>>> p.__name__
'p'

>>> import pickle
>>> p2 = pickle.dumps(p)
Traceback (most recent call last):
...
TypeError: Not picklable
```

Proxies should get their doc strings from the object they proxy:

```
>>> p.__doc__ == l.__doc__
True
```

If we get a "located class" somehow, its doc string well be available through proxy as well:

```
>>> class LocalClass(object):
...     """This is class that can be located"""

>>> p = LocationProxy(LocalClass)
>>> p.__doc__ == LocalClass.__doc__
True
```

LocationInterator()

This function allows us to iterate over object and all its parents.

```
>>> from zope.location.location import LocationIterator

>>> o1 = Location()
>>> o2 = Location()
>>> o3 = Location()
>>> o3.__parent__ = o2
>>> o2.__parent__ = o1

>>> iter = LocationIterator(o3)
>>> next(iter) is o3
True
>>> next(iter) is o2
True
>>> next(iter) is o1
True
>>> next(iter)
Traceback (most recent call last):
...
StopIteration
```

located()

`located` locates an object in another and returns it:

```
>>> from zope.location.location import located
>>> a = Location()
>>> parent = Location()
>>> a_located = located(a, parent, 'a')
>>> a_located is a
True
>>> a_located.__parent__ is parent
True
>>> a_located.__name__
'a'
```

If we locate the object again, nothing special happens:

```
>>> a_located_2 = located(a_located, parent, 'a')
>>> a_located_2 is a_located
True
```

If the object does not provide ILocation an adapter can be provided:

```
>>> import zope.interface
>>> import zope.component
>>> sm = zope.component.getGlobalSiteManager()
>>> sm.registerAdapter(LocationProxy, required=(zope.interface.Interface,))

>>> l = [1, 2, 3]
>>> parent = Location()
>>> l_located = located(l, parent, 'l')
>>> l_located.__parent__ is parent
True
>>> l_located.__name__
'l'
>>> l_located is l
False
>>> type(l_located)
<class 'zope.location.location.LocationProxy'>
>>> l_located_2 = located(l_located, parent, 'l')
>>> l_located_2 is l_located
True
```

When changing the name, we still do not get a different proxied object:

```
>>> l_located_3 = located(l_located, parent, 'new-name')
>>> l_located_3 is l_located_2
True

>>> sm.unregisterAdapter(LocationProxy, required=(zope.interface.Interface,))
True
```

CHAPTER 2

zope.location API

zope.location.interfaces

zope.location.location

zope.location.traversing

CHAPTER 3

Hacking on `zope.location`

Getting the Code

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

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

You can get a read-only checkout from there:

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

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

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

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

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

You can branch the trunk from there using Bazaar:

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

Working in a `virtualenv`

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.location
```

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

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

Running the tests

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

```
$ /tmp/hack-zope.location/bin/python setup.py test -q
..........................................................................
----------------------------------------------------------------------
Ran 83 tests in 0.037s

OK
```

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

```
$ /tmp/hack-zope.location/bin/nosetests
.....................................................................................
↪.
----------------------------------------------------------------------
Ran 87 tests in 0.037s

OK
```

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

```
$ /tmp/hack-zope.location/bin/easy_install nose coverage
...
$ /tmp/hack-zope.location/bin/nosetests --with coverage
.....................................................................................
↪.
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
zope.location                   5      0   100%
zope.location._compat           2      0   100%
zope.location.interfaces       23      0   100%
zope.location.location         61      0   100%
zope.location.pickling         14      0   100%
zope.location.traversing       80      0   100%
---------------------------------------------------------
TOTAL                         185      0   100%
----------------------------------------------------------------------
Ran 87 tests in 0.315s

OK
```

Building the documentation

`zope.location` 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.location/bin/easy_install \
 Sphinx repoze.sphinx.autoitnerface zope.component
...
$ cd docs
```

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

...

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

Using `zc.buildout`

Setting up the buildout

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

```
$ /path/to/python2.7 bootstrap.py
...
Generated script '.../bin/buildout'
$ bin/buildout
Develop: '/home/jrandom/projects/Zope/zope.location/.'
...
Got coverage 3.7.1
```

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 79 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.
```

Using tox

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.location configures the following tox environments via its tox.ini file:

- The py26, py27, py33, py34, and pypy environments builds a virtualenv with pypy, installs zope.location and dependencies, and runs the tests via python setup.py test -q.
- The coverage environment builds a virtualenv with python2.6, installs zope.location, installs nose and coverage, and runs nosetests with statement coverage.
- The docs environment builds a virtualenv with python2.6, installs zope.location, 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: /home/jrandom/projects/Zope/Z3/zope.location/setup.py
py26 create: /home/jrandom/projects/Zope/Z3/zope.location/.tox/py26
py26 installdeps: zope.configuration, zope.copy, zope.interface, zope.proxy, zope.
↪schema
py26 inst: /home/jrandom/projects/Zope/Z3/zope.location/.tox/dist/zope.location-4.0.4.
↪dev0.zip
py26 runtests: PYTHONHASHSEED='3489368878'
py26 runtests: commands[0] | python setup.py test -q
running test
...
.....................................................................................
----------------------------------------------------------------------
Ran 83 tests in 0.066s

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.location/setup.py
py26 sdist-reinst: .../zope.location/.tox/dist/zope.location-4.0.2dev.zip
...
Doctest summary
===============
  187 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
  py33: commands succeeded
  py34: commands succeeded
  pypy: commands succeeded
  coverage: commands succeeded
  docs: commands succeeded
  congratulations :)
```

Contributing to `zope.location`

Submitting a Bug Report

`zope.location` tracks its bugs on Github:

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

Please submit bug reports and feature requests there.

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.location/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.location/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 4

Indices and tables

- genindex
- modindex
- search