gocept.selenium

Release 1.0

October 08, 2015

Contents

gocept.selenium provides an API for the Selenium remote control that is suited for writing tests and integrates this with your test suite for any WSGI, Plone, Zope 2, ZTK, or Grok application.

While the testing API could be used independently, the integration is done using test layers, which are a feature of zope.testrunner.

Contents:

CHAPTER 1

Installation

Download the Selenium Remote Control JAR from seleniumhq.org and run:

```
$ java -jar /path/to/selenium-server.jar
```

This starts the server process that your tests will connect to to spawn and control the browser.

Choose the appropriate test layer (see Integration) and create a test case:

```
import gocept.selenium.wsgi
from mypackage import App

test_layer = gocept.selenium.wsgi.Layer(App())


class TestWSGITestCase(gocept.selenium.wsgi.TestCase):

    layer = test_layer

    def test_something(self):
        self.selenium.open('http://%s/foo.html' % self.selenium.server)
        self.selenium.assertBodyText('foo')
```

1.1 Environment variables

You can configure the selenium server that gocept.selenium connects to from the command line. Selenium RC defaults to localhost:4444, but you can also connect to a selenium grid in your organization by using the following environment variables:

```
GOCEPT_SELENIUM_SERVER_HOST=selenium.mycompany.com
GOCEPT_SELENIUM_SERVER_PORT=8888
```

If multiple browsers are connected to your selenium grid, you can choose the browser to run the tests with as such:

```
GOCEPT_SELENIUM_BROWSER=*iexplore
```

When you are running your selenium tests on a selenium grid, you need to instruct the browser which host and port to connect to:

```
GOCEPT_SELENIUM_APP_HOST=10.0.0.15
GOCEPT_SELENIUM_APP_PORT=8001
```

The default for the port to bind is 0 which let the kernel choose a random, free port.

When you are testing an application on one machine, you can access the running application from another machine if you set `GOCEPT_SELENIUM_APP_HOST = 0.0.0.0` instead of the default `localhost`.

You can set the speed with which the tests are run through an environment variable:

```
GOCEPT_SELENIUM_SPEED=500
```

This example will introduce a 500 millisecond pause between tests.

1.2 Tips & Tricks

1.2.1 Using a custom Firefox profile

For debugging purposes it's helpful to have the Firebug debugger available in the Selenium-controlled browser. To do that, create a new Firefox profile and install Firebug into it. Then you can tell Selenium to use this profile for running Firefox:

```
$ java -jar /path/to/selenium-server.jar -firefoxProfileTemplate ~/.mozilla/firefox/<PROFILE_FOLDER>
```

1.2.2 Using a nested X Server

Under Linux, the Selenium-controlled browser tends to steal the mouse focus, which makes it impossible to do anything else while a Selenium test is running. To prevent this, use Xephyr (successor of Xnest) to start an X server contained in a window, for example:

```
#!/bin/sh
display=:1
Xephyr -host-cursor -dpi 100 -wr -screen 1400x900 $display &
export DISPLAY=$display
sleep 2
metacity &  # or any other window manager
x-terminal-emulator -e java -jar /path/to/selenium-server.jar
```

CHAPTER 2

Integration

gocept.selenium provides integration with several web frameworks. Different frameworks require different dependencies; this is handled via setuptools extras of gocept.selenium (e.g. for Grok integration you need to require `gocept.selenium[grok]`).

Generally, you need a test layer that handles the setup, and then have your tests inherit from the appropriate `TestCase`.

2.1 WSGI

No extra requirements (simply `gocept.selenium`).

This test layer takes a WSGI callable and runs it in a temporary HTTP server:

```
import gocept.selenium.wsgi
from mypackage import App

test_layer = gocept.selenium.wsgi.Layer(App())

class WSGIExample(gocept.selenium.wsgi.TestCase):

    layer = test_layer

    def test_something(self):
        self.selenium.open('http://%s/foo.html' % self.selenium.server)
        self.selenium.assertBodyText('Hello world!')
```

2.2 Static files

No extra requirements (simply `gocept.selenium`).

This test case provides a temporary directory (as `self.documentroot`) that is served via HTTP where tests can put HTML files to examine:

```
import gocept.selenium.static

class StaticFilesExample(gocept.selenium.static.TestCase):

    def test_something(self):
        open(os.path.join(self.documentroot, 'foo.html'), 'w').write(
```

```
            'Hello World!')
            self.selenium.open('http://%s/foo.html' % self.selenium.server)
            self.selenium.assertBodyText('Hello world!')
```

2.3 Zope3 / ZTK (zope.app.testing)

Requires `gocept.selenium[ztk]`.

This test layer wraps your usual ZCMLLayer that is used for typical ZTK functional tests, and provides an HTTP server for testing:

```
import gocept.selenium.ztk
import zope.app.testing.functional

zcml_layer = zope.app.testing.functional.ZCMLLayer(
    'ftesting.zcml', __name__, __name__, allow_teardown=True)
selenium_layer = gocept.selenium.ztk.Layer(zcml_layer)

class ZTKExample(gocept.selenium.ztk.TestCase):

    layer = selenium_layer

    def test(self):
        self.selenium.open('http://%s/foo.html' % self.selenium.server)
        self.selenium.assertBodyText('Hello world!')
```

2.4 Zope3 / ZTK (zope.app.wsgi)

If your ZTK application uses `zope.app.wsgi.testlayer`, see *Grok* for integrating `gocept.selenium`.

2.5 Grok

Requires `gocept.selenium[grok]`.

This test layer groks your package and sets everything up so Selenium can access the application. You will probably want to setup your app in your test setup:

```
import gocept.selenium.grok
import transaction

selenium_layer = gocept.selenium.grok.Layer(my.package)

class GrokExample(gocept.selenium.grok.TestCase):

    layer = selenium_layer

     def setUp(self):
         super(MyTest, self).setUp()
         root = self.getRootFolder()
         root['app'] = mypackage.App()
         transaction.commit()
```

```
    def test(self):
        self.selenium.open('/app')
        self.selenium.assertBodyText('Hello world!')
```

2.6 Zope 2

Requires `gocept.selenium[zope2]`

This test layer requires `Testing.ZopeTestCase.layer.ZopeLiteLayer` and provides an HTTP server for the tests. See `gocept.selenium.zope2.tests.test_zope212` for details how to set this up.

2.7 Zope 2 via WSGI

If your Zope 2 setup supports it, you can use the WSGI integration instead of a specialised Zope 2 integration to run your tests.

You might see the following exception when running tests:

```
File ".../repoze.retry-1.0-py2.7.egg/repoze/retry/__init__.py", line 55, in __call__
  cl = int(cl)
 ValueError: invalid literal for int() with base 10: ''
```

To fix it you can use an additional middleware around your WSGI application: `gocept.selenium.wsgi.CleanerMiddleware`. It also fixes an issue with `wsgiref`. See comments in the code for more information.

2.8 Plone

Requires `gocept.selenium[plone]`.

This test layer requires `Products.PloneTestCase.laye.PloneSiteLayer` and provides an HTTP server for the tests. See `gocept.selenium.plone.tests.test_plone{3,4}` for details how to set this up.

2.9 Zope 2 / Plone with plone.testing

Requires `gocept.selenium[plonetesting]`.

`gocept.selenium` provides a `plone.testing.Layer` at `gocept.selenium.plonetesting.SELENIUM` that you can mix and match with your other layers, see `gocept.selenium.plonetesting.testing` with `gocept.selenium.plonetesting.tests.zope2`, and `gocept.selenium.plonetesting.testing_plone` with `gocept.selenium.plonetesting.tests.plone{3,4}` for details how to set this up.

2.10 Converting Selenese HTML files

Selenium tests can be written in HTML tables.

Their syntax is a bit clunky. But their development and debugging is eased a lot by using Selenium IDE Firefox extension. Selenium IDE provides both initial recording of tests and stepping through those tests. However, HTML tests have a main drawback: they are hard to include in a continuous integration system.

`gocept.selenium` provides a script that converts a set of Selenium HTML tests into a Python module with a `TestCase` (based on `gocept.selenium` and `plone.testing`).

Using the `converthtmltests` script, the developer can use HTML tests – written, debugged and maintained with the Selenium tools – while being able to easily include those Selenium tests in a continuous integration system.

2.10.1 Usage

```
converthtmltests -l LAYER [options] directory

options:
  -f FILE, --file=FILE  write tests to FILE
  -l LAYER, --layer=LAYER
                        full python import path to layer instance
```

The script gathers and converts all Selenium HTML tests found in the mentioned directory.

The user must refer to a `plone.testing` layer by specifying its Python import path. That layer is set on the test case generated in the Python module.

An output file can be specified. In case no output file name is specified, the module produced is named `tests_all_selenium.py`.

On Python-2.4, converthtmltests requires `gocept.selenium[script]`.

CHAPTER 3

API reference

3.1 Selenese API

The `Selenese` object available as `self.selenium` for each TestCase provides methods to control the browser, and to make assertions about things the browser sees.

For a detailed list of commands and assertions please consult the Selenium Reference.

Assertions come in several flavours:

- Return the value `self.selenium.getText('id=foo')`
- Assert `self.selenium.assertText('id=foo', 'blabla')`
- Negated Assert `self.selenium.assertNotText('id=foo', 'blabla')`
- Wait `self.selenium.waitForElementPresent('id=foo')`
- Negated Wait `self.selenium.waitForNotElementPresent('id=foo')`

3.2 Test helpers

3.2.1 Skipping tests for certain browsers

There are cases when a test should does not pass on certain browsers. This is either due to the application using browser features which are not supported by the browser, or due to selenium not working well with the browser. To aid in skipping tests in these cases, there is a test decorator `gocept.selenium.skipUnlessBrowser(name, version=None)`:

```
>>> class TestClass(...):
...
... @gocept.selenium.skipUnlessBrowser('Firefox', '>=16.0')
... def test_fancy_things(self):
...     ...
```

Note: `skipUnlessBrowser` *only* supports skipping test methods. It cannot be used as class decorator.

Warning: The version test is only supported for Python >= 2.5. For Python < 2.5 *only* a name check can be performed. Giving a version number will skip the test unconditionally.

CHAPTER 4

Developing gocept.selenium

Author gocept <mail@gocept.com>

Online documentation http://packages.python.org/gocept.selenium/

PyPI page http://pypi.python.org/pypi/gocept.selenium/

Issue tracker https://projects.gocept.com/projects/gocept-selenium/

Source code https://bitbucket.org/gocept/gocept.selenium/

Current change log https://bitbucket.org/gocept/gocept.selenium/raw/tip/CHANGES.txt

4.1 buildout configuration

gocept.selenium integrates with quite a lot of different testing approaches and needs to work across a wide spectrum of software versions, e. g. Zope2 before and after eggification (2.10/2.12), ZTK-KGS, Grok-KGS, Plone3, Plone4 etc.

This has two consequences, one is that we use different extras_require for the different flavours, so clients will need to specify that, e. g. gocept.selenium[ztk] or gocept.selenium[grok].

The second is that there is no single buildout configuration for this package, but rather quite a lot of them, so we are able to run our tests against all the different software versions we integrate with.

The base package itself is tested with `selenium.cfg`, this has no further dependencies except the `selenium` package. The various flavours have their own cfg file, in some cases in several versions (e.g. Plone3/Plone4, Zope2 pre/post eggs etc.). This means that in order to set up the buildout, you'll need to specify the configuration you want to test, like this:

```
$ python bootstrap.py -c ztk.cfg
$ bin/buildout -c ztk.cfg
```

Note that the zope210 and plone3 configurations require Python-2.4, while the others should work at least up to Python-2.6.

CHAPTER 5

Changelog

5.1 1.1.2 (2012-12-21)

- Fix: Initialise the WSGI layer in the correct order to actually allow the configured WSGI app to be remembered.
- Fix: updated some imports after the extraction of gocept.httpserverlayer.

5.2 1.1.1 (2012-12-19)

- Update StaticFilesLayer to the new httpserverlayer API.

5.3 1.1 (2012-12-19)

- Extract HTTP server integration into separate package, gocept.httpserverlayer

5.4 1.0 (2012-11-03)

- Marking the API as stable.

5.5 0.17 (2012-11-01)

- Added `gocept.selenium.skipUnlessBrowser` decorator to skip tests unless ceratins browser requirements are met.
- Fix: The static test server did not shutdown in some situations.

5.6 0.16 (2012-10-10)

- Fixed selenese popup tests.
- Open a random port for the server process by default: When the environment variable *GOCEPT_SELENIUM_APP_PORT* is not set, a random free port is bound. This allows parallel testing, for instance (#11323).

5.7 0.15 (2012-09-14)

- WSGI-Layer is comptabile with Python 2.5.
- Encoding support in converthtmltests (Patch by Tom Gross <tom@toms-projekte.de>).
- XHTML support for selenium tables (Patch by Tom Gross <tom@toms-projekte.de>).

5.8 0.14 (2012-06-06)

- API expansion: Added `assertCssCount`. Thus requiring selenium >= 2.0.
- Added Trove classifiers to package metadata.
- Moved code to Mercurial.

5.9 0.13.2 (2012-03-15)

- Fixed WSGI flavor: There was a `RuntimeError` in tear down if the WSGI server was shut down correctly.

5.10 0.13.1 (2012-03-15)

- Updated URL of bug tracker.
- *script* extra no longer requires *elementtree* on Python >= 2.5.

5.11 0.13 (2012-01-30)

- Added a selenese assert type 'list' and added it to the window management query methods.
- API expansion: added `openWindow`.
- API change: filter the result of `getAllWindowNames` to ignore 'null'.
- backwards-compatible API change: `selectWindow` now selects the main window also when passed the window id `None` or no argument at all.
- pinned compatible ZTK version to 1.0.1, grok version to 1.2.1, generally pinned all software packages used to consistent versions for this package's own testing

5.12 0.12 (2011-11-29)

- API expansion: added `getAllWindow*` and `selectWindow`.

5.13 0.11 (2011-09-15)

- Added some notes how to test a Zope 2 WSGI application.
- Described how to test a Zope 2/Plone application if using *plone.testing* to set up test layers.

5.14 0.10.1 (2011-02-02)

- Improvements on the README.
- Wrote a quick start section for packages using ZTK but using `zope.app.wsgi.testlayer` instead of `zope.app.testing`.
- Allowed to use *regexp* as pattern prefix for regular expressions additionally to *regex* to be compatible with the docstring and the Selenium documentation.

5.15 0.10 (2011-01-18)

- Script that generates python tests from Selenium HTML tables. Reused from KSS project, courtesy of Jeroen Vloothuis, original author.
- Using a URL of *Selenium RC* in README where version 1.0.3 can be downloaded (instead of 1.0.1) which works fine with Firefox on Mac OS X, too.

5.16 0.9 (2010-12-28)

- Provide integration with the recent testlayer approach (zope.app.appsetup/zope.app.wsgi) used by Grok (#8260).
- Provide integration with plone.testing
- Make browser and RC server configurable (#6484).
- Show current test case in command log (#7876).
- Raise readable error when connection to RC server fails (#6489).
- Quit browser when the testrunner terminates (#6485).

5.17 0.8 (2010-10-22)

- Fixed tests for the StaticFilesLayer to pass with Python 2.4 through 2.7.
- API expansion: `getSelectOptions`

5.18 0.7 (2010-08-16)

- API expansion: `getElementHeight|Width`, `getCookie*` and a few others.
- lots of action methods (`mouse*` among others)

5.19 0.6 (2010-08-09)

- assertXpathCount now also takes ints (#7681).
- API expansion: add `isChecked` to verify checkboxes, `runScript`, `clickAt`, `getLocation`, `getSelectedValue`, `getSelectedIndex`.
- The `pause` method uses float division now. Pauses where implicitly rounded to full seconds before when an int was passed.
- The name of the factored test layer contains the module of the bases now. The name is used by zope.testrunner distinguish layers. Before this fix selenium layers factored from base layers with the same names but in different modules would be considered equal by zope.testrunner.
- The factored ZTK layer cleanly shuts down the http server in tearDown now. This allows to run different selenium layers in one go.

5.20 0.5 (2010-08-03)

- Add a static files test layer for running selenium tests against a set of static (HTML) files.
- Patterns now also work with multiline strings, i. e. 'foo*' will match 'foonbar' (#7790).

5.21 0.4.2 (2010-05-20)

- API expansion: `*keyDown`, `*keyUp`, `keyPress`.

5.22 0.4.1 (2010-04-01)

- API expansion: added `getSelectedLabel`.
- Ignore the code of a server's response when calling *open*. The default behaviour of SeleniumRC changed between 1.0.1 and 1.0.2 but we want the old behaviour by default.

5.23 0.4 (2010-03-30)

- API expansion: add `getLocation` to retrieve currently loaded URL in browser.
- API expansion: added `waitForPopUp`, `selectPopUp`, `deselectPopUp` and `close`.
- API expansion: added `verifyAlertPresent`, `verifyAlertNotPresent` and `waitForAlertPresent`.
- Usability: raise a better readable exception when an unimplemented selenese method is called.
- Usability: raise failure exceptions that convey the name of the failed assertion in spite of some lambdas wrapped around it.

5.24 0.3 (2010-01-12)

- Extracted 'host' and 'port' as class attributes of gocept.selenium.ztk.Layer so subclasses can override them; stopped hardcoding 8087 as the server port.

5.25 0.2.1 (2009-12-18)

- Fix incomplete sdist release on PyPI.

5.26 0.2 (2009-12-18)

- Make Zope 2 test server reachable from the outside.
- Implemented getTitle/assertTitle/waitForTitle/etc.

5.27 0.1 (2009-11-08)

- first release