# Pylons Project Documentation

***Release 0.0***

**Pylons Developers**

June 29, 2015

# Contents


**1 Pylons Project Documentation** **3**
- 1.1 Pyramid Documentation . . . 3
- 1.2 Pylons (the web framework) Documentation . . . 7
- 1.3 Pylons Project Library Package Documentation . . . 7

**2 Support** **9**

**3 FAQ** **11**
- 3.1 Pylons Project FAQ . . . 11

**4 Podcasts** **15**
- 4.1 Podcasts . . . 15

**5 Promote** **17**
- 5.1 Official Pylons Project Logos . . . 17
- 5.2 Pyramid Badges . . . 18
- 5.3 Pylons Project Desktops . . . 19

**6 Contributing** **23**
- 6.1 Community Code of Conduct . . . 23
- 6.2 Contributing Source Code and Documentation . . . 24
- 6.3 Coding Style and Standards . . . 27
- 6.4 Unit Testing Guidelines . . . 30
- 6.5 Adding Features and Dealing with Bugs . . . 37
- 6.6 Add-on and Development Environment Guidelines . . . 38

**7 Denials** **41**
- 7.1 Pyramid Denials . . . 41
- 7.2 Glossary . . . 41
- 7.3 Pylons RTD . . . 42

The Pylons Project maintains the Pyramid web framework as well as additional packages intended for use with Pyramid.

# CHAPTER 1

# Pylons Project Documentation

The following are documentation of projects under the Pylons Project.

***Pyramid Documentation*** Includes the official narrative documentation, API, and tutorials, as well as links to unofficial or community contributions to cookbook recipes, and sample Pyramid applications.

***Pyramid Add-ons*** A list of over a dozen Add-ons which extends Pyramid's functionality and whose names begin with `pyramid_`. Examples of Add-ons include templating, the debug toolbar, mailer, and transaction manager.

***Pylons Project Library Package Documentation*** A list of packages under the Pylons Project that are useful outside of Pyramid.

***Pylons (the web framework) Documentation*** Documentation of Pylons (the web framework).

For the most current list of all repositories and projects under the Pylons Project, visit the Pylons Project organization page on GitHub.

## 1.1 Pyramid Documentation

### 1.1.1 Getting Started

If you are new to Pyramid, we have a few resources that can help you get up to speed right away:

- Installing Pyramid.
- To see a minimal Pyramid web application, check out Creating Your First Pyramid Application.
- Pyramid Frequently Asked Questions.
- To get a detailed experience of creating a Pyramid web application, try the Quick Tutorial for Pyramid.
- Like learning by example? Check out to the official Pyramid Tutorials as well as the community-contributed Pyramid Tutorials and Pyramid Cookbook Recipes.
- Need help? See *Support*.

### 1.1.2 Main Documentation

#### Stable branch : version 1.5 (latest)

- Pyramid latest documentation (latest PDF) (latest Epub) - narrative and API documentation for Pyramid's latest version, 1.5, the current stable release.

#### Development branch : upcoming 1.6 (1.6-branch)

- Pyramid development documentation - narrative and API documentation for Pyramid's in-development version.

#### Tutorials and Cookbook Recipes

- A list of all official and community-contributed Pyramid Tutorials. You may submit your tutorial for inclusion by submitting a pull request through the pyramid_tutorials repository.
- The Pyramid Cookbook presents topical, practical uses of Pyramid. The Pyramid Cookbook is not a source of official Pyramid documentation, but is a collection of contributions from the community. You may submit your Pyramid recipes for inclusion in the Cookbook through the pyramid_cookbook repository.

#### Previous versions

- Pyramid documentation 1.4 (1.4 PDF) (1.4 Epub) - narrative and API documentation for Pyramid's 1.4 version.
- Pyramid documentation 1.3 (1.3 PDF) (1.3 Epub) - narrative and API documentation for Pyramid's 1.3 version.
- Pyramid documentation 1.2 (1.2 PDF) (1.2 Epub) - narrative and API documentation for Pyramid's 1.2 version.
- Pyramid documentation 1.1 (1.1 PDF) (1.1 Epub) - narrative and API documentation for Pyramid's 1.1 version.
- Pyramid documentation 1.0 (1.0 PDF) (1.0 Epub) - narrative and API documentation for Pyramid's 1.0 version.

### 1.1.3 Pyramid Add-ons

#### Supported Add-ons

Pyramid supports extensibility through add-ons. The following add-ons are officially endorsed by the Pylons Project. Documentation for each add-on is hosted at its respective name under the Pylons Project.

- pyramid_chameleon: Chameleon templating bindings for Pyramid.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_chameleon
- pyramid_debugtoolbar, an interactive HTML debug toolbar for Pyramid.
  - Maintained by: Chris McDonough, Blaise Laflamme
  - Version Control: https://github.com/Pylons/pyramid_debugtoolbar
- pyramid_exclog, a package which logs exceptions from Pyramid applications.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_exclog
- pyramid_handlers: analogue of Pylons-style "controllers" for Pyramid.
  - Maintained by: Ben Bangert, Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_handlers
- pyramid_jinja2: Jinja2 template renderer for Pyramid
  - Maintained by: Domen Kožar
  - Version Control: https://github.com/Pylons/pyramid_jinja2

- pyramid_jqm, scaffolding for developing jQuery Mobile apps with Pyramid.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_jqm
- pyramid_layout: Pyramid add-on for managing UI layouts.
  - Maintained by: Chris Rossi, Paul Everitt, Blaise Laflamme
  - Version Control: https://github.com/Pylons/pyramid_layout
- pyramid_ldap, an LDAP authentication policy for Pyramid.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_ldap
- pyramid_mailer: a package for the Pyramid framework to take the pain out of sending emails.
  - Maintained by: Dan Jacobs, Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_mailer
- pyramid_mako: Mako templating bindings for Pyramid.
  - Maintained by: Bert JW Regeer
  - Version Control: https://github.com/Pylons/pyramid_mako
- pyramid_rpc: RPC service add-on for Pyramid, supports XML-RPC in a more extensible manner than *pyramid_xmlrpc* with support for JSON-RPC and AMF.
  - Maintained by: Michael Merickel, Ben Bangert
  - Version Control: https://github.com/Pylons/pyramid_rpc
- pyramid_tm: Centralized transaction management for Pyramid applications (without middleware).
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_tm
- pyramid_who: Authentication policy for pyramid using repoze.who 2.0 API.
  - Maintained by: Chris McDonough, Tres Seaver
  - Version Control: https://github.com/Pylons/pyramid_who
- pyramid_xmlrpc: XML-RPC add-on for Pyramid
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_xmlrpc
- pyramid_zcml: Zope Configuration Markup Language configuration support for Pyramid.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_zcml
- pyramid_zodbconn: ZODB Database connection management for Pyramid.
  - Mantained by: Chris McDonough, Chris Rossi
  - Version Control: https://github.com/Pylons/pyramid_zodbconn

#### Unsupported Add-Ons

These are libraries which used to be officially supported by the Pylons Project, but have since become unsupported.

- pyramid_beaker: Beaker session backend plug-in.
  - Maintained by: Ben Bangert, Chris McDonough
  - Version Control: https://github.com/Pylons/pyramid_beaker
  - Became unsupported October 2013 because Beaker itself is no longer maintained.

### 1.1.4 Sample Pyramid Applications

**cluegun** A simple pastebin application based on Rocky Burt's ClueBin. It demonstrates form processing, security, and the use of ZODB within a *Pyramid* application.

- Version Control: https://github.com/Pylons/cluegun

**KARL** A moderately-sized application (roughly 80K lines of Python code) built on top of *Pyramid*. It is an open source web system for collaboration, organizational intranets, and knowledge management. It provides facilities for wikis, calendars, manuals, searching, tagging, commenting, and file uploads. See the KARL site for download and installation details.

**shootout** An example "idea competition" application by Carlos de la Guardia and Lukasz Fidosz. It demonstrates URL dispatch, simple authentication, integration with SQLAlchemy and `pyramid_simpleform`.

- Version Control: https://github.com/Pylons/shootout.git

**virginia** A very simple dynamic file rendering application. It is willing to render structured text documents, HTML documents, and images from a filesystem directory. It's also a good example of traversal. An earlier version of this application runs the repoze.org website.

- Version Control: https://github.com/Pylons/virginia.git

### 1.1.5 Sample Pyramid Development Environments

"Development environments" are packages which use Pyramid as a core, but offer alternate services and scaffolding. Each development environment presents a set of opinions and a "personality" to its users. Although users of a development environment can still use all of the services offered by the Pyramid core, they are usually guided to a more focused set of opinions offered by the development environment itself. Development environments often have dependencies beyond those of the Pyramid core.

**Akhet** A Pyramid library and demo application with a Pylons-like feel. Its most known for its former application scaffold, which helped users transition from Pylons and those preferring a more Pylons-like API. The scaffold has been retired but the demo plays a similar role.

**Khufu Project** Khufu is an application scaffolding for Pyramid that provides an environment to work with Jinja2 and SQLAlchemy.

- Maintained by: Rocky Burt
- Version Control: https://github.com/khufuproject

**Kotti** Kotti is a high-level, Pythonic web application framework. It includes an extensible Content Management System called the Kotti CMS, offering all the features you would expect from a modern CMS.

- Version Control: https://github.com/Kotti/Kotti

**Nefertari** Nefertari is a REST API framework for Pyramid that uses ElasticSearch for reads and either MongoDB or Postgres for writes. It provides an interface to ElasticSearch's Query String DSL for full text search.

- Version Control: https://github.com/brandicted/nefertari

**Ptah** Ptah is a fast, fun, open source high-level Python web development environment.

- Version Control: https://github.com/ptahproject/ptah

**Ramses** Ramses is a library that generates a RESTful API using *RAML <http://raml.org>*. It uses Pyramid and *Nefertari <https://nefertari.readthedocs.org/>* which provides ElasticSearch-powered views.

- Version Control: https://github.com/brandicted/ramses

**Ringo** Ringo is an extensible high-level web application framework with strength in building form based management or administration software, providing ready to use components often needed in web applications.

- Version Control: https://bitbucket.org/ti/ringo

**Substance-D** An application server built upon the Pyramid web framework. It provides a user interface for managing content as well as libraries and utilities which make it easy to create applications.

- Version Control: https://github.com/Pylons/substanced

**Ziggurat** A bundled application framework for data driven Pyramid project development.

- Version Control: https://github.com/sernst/Ziggurat

## 1.2 Pylons (the web framework) Documentation

### 1.2.1 Main Documentation Sources

- Pylons documentation 1.0: official Pylons (the web framework) documentation.
- Definitive Guide to Pylons: Pylons book published by Apress, written by James Gardner, free HTML rendering.

## 1.3 Pylons Project Library Package Documentation

The following packages are hosted within the Pylons Project's GitHub repository structure, and are officially endorsed, but are not Pyramid add-ons.

- WebOb, WSGI request/response library
  - Maintained by: Bert W Regeer
  - Version Control: https://github.com/Pylons/webob
- colander, serialization/deserialization/validation library.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/colander
- deform: an HTML form library.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/deform
- peppercorn, a library for converting a token stream into a data structure comprised of sequences, mappings, and scalars, developed primarily for converting HTTP form post data into a richer data structure.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/peppercorn

- translationstring, an internationalization library used by various Pylons Project software.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/translationstring
- venusian, a library useful for framework authors to defer decorator actions.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/venusian
- webhelpers, a library for python web helpers.
  - Maintained by: Ben Bangert, Mike Orr
  - Version Control: https://bitbucket.org/bbangert/webhelpers
- Waitress, WSGI server that runs under Python 2 and Python 3 on UNIX and Windows.
  - Maintained by: Chris McDonough
  - Version Control: https://github.com/Pylons/waitress

CHAPTER 2

# Support

Development questions related to projects under the Pylons Project can be discussed on the pylons-discuss mail list.

On IRC, developers are generally available on the `#pylons` channel on the Freenode IRC Network. Each project may have its own channel, too.

**Using Support Wisely**

Before asking a technical question on the mail list(s) or in IRC, please make sure to try the following things (paraphrased from Before You Ask):

- Try to find an answer by reading the manual.
- Try to find an answer by searching the mail list archives.
- Try to find an answer by searching the Web.
- Try to find an answer by inspection or experimentation.
- If you're a programmer, try to find an answer by reading the source code.

After exhausting these avenues, it's completely appropriate to ask a question on the Pylons-discuss mail list or #pylons IRC channel. When you ask your question, please describe what you've learned from the efforts above, as it will help the developers focus on answering your question quickly. It also helps tremendously if you are able to provide a code or configuration snippet that makes the problem easily repeatable.

CHAPTER 3

# FAQ

For Pylons users coming from Pylons 1 or repoze.bfg, the change to a new core package might raise some questions regarding how to proceed and what it means for existing applications.

## 3.1 Pylons Project FAQ

### 3.1.1 How does "The Pylons Project" relate to Pylons-the-web-framework?

The Pylons Project was founded by the people behind the Pylons web framework to develop web application framework technology in Python. Rather than focusing on a single web framework, the Pylons Project develops a collection of related technologies.

The first package from the Pylons Project is the Pyramid web framework. Other packages will be added to the collection over time, including higher-level components, applications, and other frameworks which rely on a particular persistence mechanism (Pyramid does not). Ben Bangert, the Pylons web framework project lead, is already aiming to develop layers above the core web framework.

"The Pylons Project" was chosen to reflect the shared core ethos with the Pylons web framework: an overall effort combining the best parts from different projects.

### 3.1.2 Why not just continue developing the Pylons 1.0 code-base?

Unfortunately, the Pylons 1.0 code-base has hit a point of diminishing returns in flexibility and extendability. Due to the use of sub-classing, extensive and sometimes confusing use of Stacked Object Proxy globals, and issues with configuration organization, attempts to refactor or redesign the existing `pylons` core weren't working out.

Over the course of several months, serious attempts were made to rewrite sections of the `pylons` core. After realizing that Pylons users would have to put in extensive effort to port their existing applications, and that Pylons 2 was looking more and more like `repoze.bfg`, continued development seemed a waste of development effort.

Ben Bangert started collaborating with Chris McDonough to bring the `repoze.bfg` routes functionality up to par with the stand-alone Routes project. Further development showed that the two projects had much in common, and the developers shifted from building Pylons 2 on top of BFG and towards a full merger.

### 3.1.3 What does the Pylons Project mean for Pylons-the-web-framework?

The Pylons web framework 1.x line will continue to be maintained, though not enhanced. We will provide a package that allows Pylons 1.x applications and Pyramid applications to run in the same interpreter. The future of Pylon-style web application development is Pyramid. See also the FAQ, Should I port my Pylons 1.0 project to Pyramid?.

### 3.1.4 What does the Pylons Project mean for repoze.bfg?

The Pyramid codebase is derived almost entirely from `repoze.bfg`. Some changes have been made for the sake of Pylons compatibility, but those used to development with `repoze.bfg` will find Pyramid very familiar. By merging `repoze.bfg` with the philosophically-similar Pylons framework, both gain a dramatically expanded audience.

### 3.1.5 What does this mean for the Repoze project?

The Repoze project will continue to exist. Repoze will be able to regain its original focus: bringing Zope technologies to WSGI. The popularity of BFG as its own web framework hindered this goal.

### 3.1.6 Why should I care about The Pylons Project?

This really is a good question. We hope that people are attracted at first by the spirit of the thing. It takes humility to sacrifice a little sovereignty and work together. The opposite—forking or splintering of projects—is much more common in the open source world. We feel there is a limited amount of oxygen in the space of "top-tier Python web frameworks" and we don't do the Python community a service by over-crowding.

We are a group of project leaders with experience going back to the start of Python web frameworks. We aim to bring fresh ideas to classic problems. We hope to combine a lot of hard-earned maturity into the development of a secure choice that developers and companies can bet on. Couple this with the humility and irreverence gained by surviving every stupid decision that could be imagined, and you've got a good basis for a team of developers.

### 3.1.7 Why is the Pylons Project different than other projects?

Our mantra is: "Small, Documented, Tested, Extensible, Fast, Stable, Friendly". Everything we do, from Pyramid to the batteries we want to develop for later "batteries-included" projects, should retain these qualities.

### 3.1.8 What do you mean by "Friendly"?

All of us have been around the block a few times. We've seen good communities and bad communities, effective communities and dysfunctional communities, arrogant ones and irreverent ones. We pride ourselves on constructive listening, telling the truth even when it makes us look bad, admitting when we're wrong, and attracting lots of people who actually like to help.

### 3.1.9 What does the Pylons Project mean for Zope and Plone?

The repoze.bfg people came from the world of Zope and Plone. Paul, for example, was a co-founder of Zope and was at the first Python conference at NIST. Zope was a tremendous success in the first cycle, with some truly fresh ideas and a large commercial ecosystem. Plone continued that in a second cycle, with an even larger ecosystem and an obvious, out-of-the-box value proposition.

Since then, the cycle has moved on and focus has shifted to other projects. We love our Zope roots, the experience we gained helping establish the Zope Foundation and the Plone Foundation, and consulting experience we have on very large projects. But we want to take these experiences and start fresh together with Pylons, one of the clear winners of the last cycle, to work on something for the next cycle.

If you're doing Zope and Plone and have a project that fits their bulls-eye, use them. If you have something that could use those ideas for an alternate need, keep an eye on what we're doing.

### 3.1.10 How do I participate?

Development questions related to projects under the Pylons Project can be discussed on the pylons-discuss mail list.

On IRC, developers are generally available on the `#pylons` channel on the Freenode IRC Network. Each project may have its own channel, too.

### 3.1.11 Where is the code?

https://github.com/Pylons

- Pyramid Frequently Asked Questions

CHAPTER 4

# Podcasts

Pylons Podcasts are a series of audio podcasts from developers of the Pylons Project. You can either subscribe in iTunes or Rhythmbox (or your favorite audio client) using the Feed URL below, or listen to individual podcasts.

Feed URL (RSS/iTunes)

## 4.1 Podcasts

### 4.1.1 Episode 1: Q&A with Ben and Chris

MP3 (length: 1:26:08, ~ 43MB)

In our inaugural episode, Chris McDonough and Ben Bangert talk about Pyramid, pyramid_deform, zc.buildout (aka "buildout"), Fabric, WebError, repoze.profile, Paste's errorcatcher, Arecibo, repoze.debug, Chris Davies' Pylons-to-Pyramid porting guide, Cucumber, Freshen, FormEncode, Formish, Deform, ZODB, repoze.folder, repoze.catalog, #pylons IRC channel, Paste, mod_wsgi, uwsgi, repoze.workflow, Colander, repoze.tm2, pyramid_routehelper, pyramid_beaker, the Pyramid todo list, zope.sqlalchemy, and the zope transaction module.

CHAPTER 5

# Promote

To promote the Pylons Project and its related technologies, or your own work made with our tools, we provide logos, badges, and desktop wallpapers.

## 5.1 Official Pylons Project Logos

The Pylons Project logos are licensed as Creative Commons Attribution-NonCommercial; please link to http://pylonsproject.org/ as the attribution when using them as an hyperlink or The Pylons Project when using as text.

### 5.1.1 Pylons logo

Online and screen use.

- `PNG Pylons logo for light background`
- `PNG Pylons logo for dark background`
- `SVG Pylons logo for light background`
- `SVG Pylons logo for dark background`

Print use.

- `EPS Pylons logo for light background`
- `EPS Pylons logo for dark background`

### 5.1.2 Pyramid Logo

Online and screen use.

- `PNG Pyramid logo for light background`
- `PNG Pyramid logo for dark background`
- `SVG Pyramid logo for light background`
- `SVG Pyramid logo for dark background`

Print use.

- `EPS Pyramid logo for light background`

- `EPS Pyramid logo for dark background`

### 5.1.3 Usage guidelines

- The logo should maintain the official logo colors (including when using transparency and gradients).
- Any scaling must retain the original proportions.
- Additional text may not be added so that it appears to be part of the logo.

## 5.2 Pyramid Badges

Is your site or project powered or made with Pyramid? Show your support by displaying one of these following badges.

### 5.2.1 Download all badges

- `badges.zip`

### 5.2.2 Pyramid badges

105x28 transparent

105x28 white

105x28 black

148x45 transparent

148x45 white

148x45 black

### 5.2.3 Powered by Pyramid

148x45 transparent

148x45 white

148x45 black

### 5.2.4 Made with Pyramid

148x45 transparent

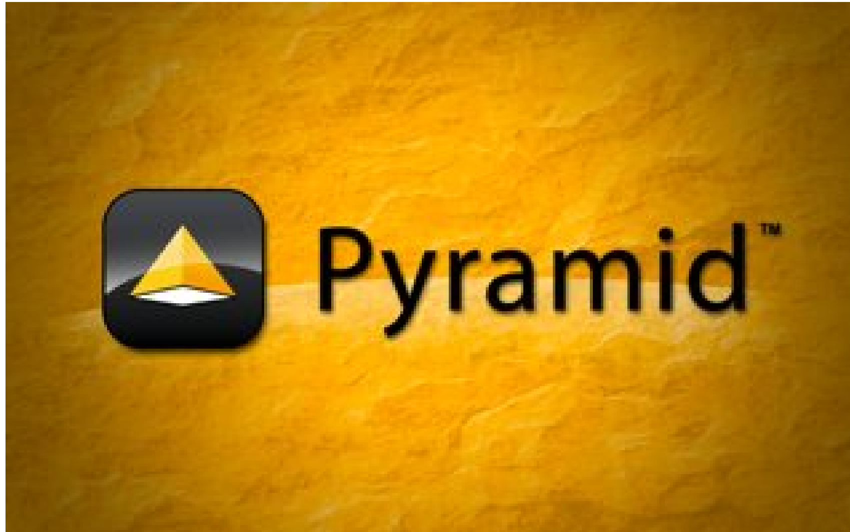
148x45 white

148x45 black

## 5.3 Pylons Project Desktops

Show others you're a Pylons Project user.

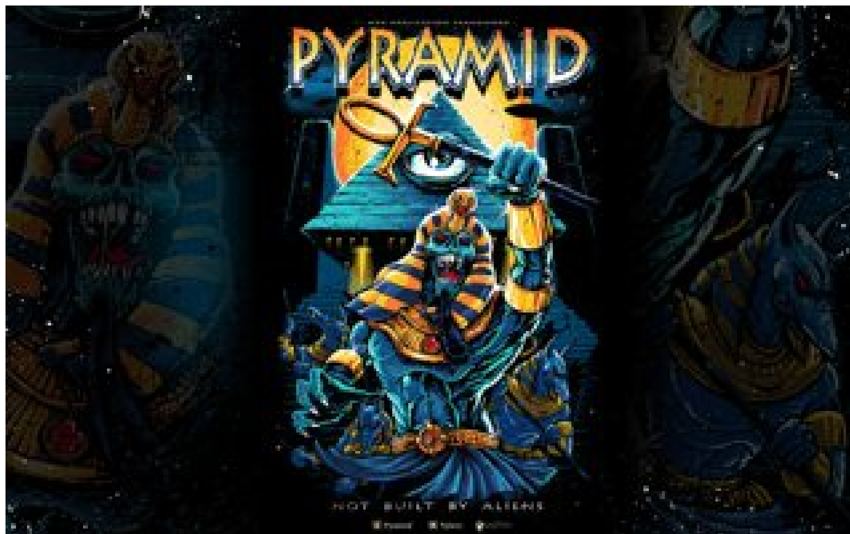
### 5.3.1 Pyramid Desktops




Choose a size

- Desktop 1024x640
- Desktop 1440x900
- Desktop 1680x1050
- Desktop 1920x1200

### 5.3.2 Not Built By Aliens #1

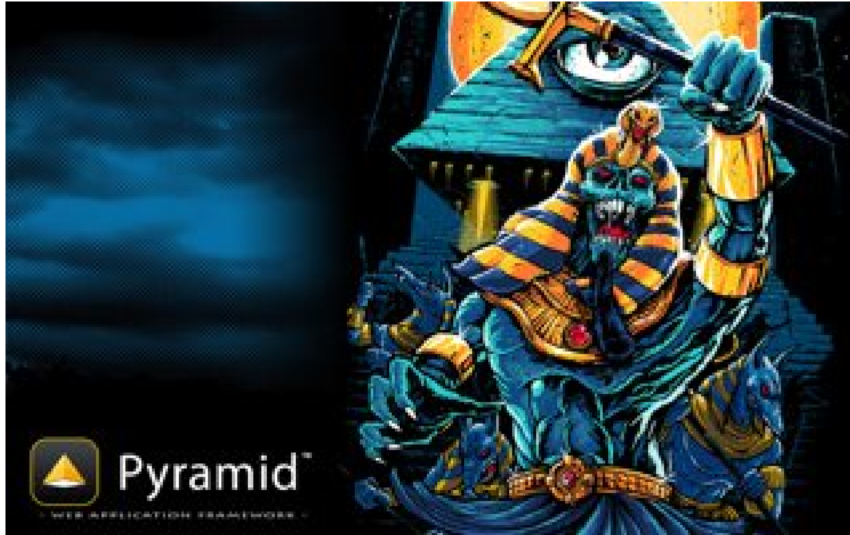



Choose a size

- Desktop 1024x640
- Desktop 1440x900
- Desktop 1680x1050

- `Desktop 1920x1200`

### 5.3.3 Not Built By Aliens #2




Choose a size

- `Desktop 1024x640`
- `Desktop 1440x900`
- `Desktop 1680x1050`
- `Desktop 1920x1200`

CHAPTER 6

# Contributing

The Pylons Project welcomes contributors. Please read the following documentation about how the Pylons Project functions, coding styles expected for contributions, and the community standards we expect everyone to abide by.

## 6.1 Community Code of Conduct

The Pylons Project developers work their hardest to adhere to a common community code of conduct based heavily on the Ubuntu Code of Conduct. We would greatly appreciate it if everyone contributing and interacting with projects under the Pylons Project also followed this Code of Conduct.

### 6.1.1 Be considerate.

Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and we expect you to take those consequences into account when making decisions. For example, when we are in a feature freeze, please don't upload dramatically new versions of critical system software, as other people will be testing the frozen system and will not be expecting big changes.

### 6.1.2 Be respectful.

The Pylons Project community and its members treat one another with respect. Everyone can make a valuable contribution to the Pylons Project. We may not always agree, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. We expect members of the Pylons Project community to be respectful when dealing with other contributors as well as with people outside the Pylons Project and with users of its projects.

### 6.1.3 Be collaborative.

The Pylons Project and Free Software are about collaboration and working together. Collaboration reduces redundancy of work done in the Free Software world, and improves the quality of the software produced. You should aim to collaborate with other Pylons Project maintainers, as well as with the upstream community that is interested in the work you do. Your work should be done transparently and patches to Pylons projects should be given back to the community when they are made, not just when the distribution releases. If you wish to work on new code for existing upstream projects, at least keep those projects informed of your ideas and progress. It may not be possible to get consensus from upstream or even from your colleagues about the correct implementation of an idea, so don't feel

obliged to have that agreement before you begin, but at least keep the outside world informed of your work, and publish your work in a way that allows outsiders to test, discuss and contribute to your efforts.

### 6.1.4 When you disagree,

. . . consult others. Disagreements, both political and technical, happen all the time and the Pylons Project community is no exception. The important goal is not to avoid disagreements or differing views, but to resolve them constructively. You should turn to the community and to the community process to seek advice and to resolve disagreements. There are several Project Teams and Team Leaders, who may be able to help you figure out which direction will be most acceptable. If you really want to go a different way, then we encourage you to make a derivative distribution or alternative set of packages that still build on the work we've done to utilize as common of a core as possible.

### 6.1.5 When you are unsure,

. . . ask for help. Nobody knows everything, and nobody is expected to be perfect in the Pylons Project community (except of course the BDFL). Asking questions avoids many problems down the road, and so questions are encouraged. Those who are asked should be responsive and helpful. However, when asking a question, care must be taken to do so in an appropriate forum. Off-topic questions, such as requests for help on a development mailing list, detract from productive discussion.

### 6.1.6 Step down considerately.

Developers on every project come and go and the Pylons Project is no different. When you leave or disengage from the project, in whole or in part, we ask that you do so in a way that minimizes disruption to the project. This means you should tell people you are leaving and take the proper steps to ensure that others can pick up where you leave off.

## 6.2 Contributing Source Code and Documentation

For substantive contributions to its major projects, The Pylons Project requires the following of its contributors:

- An assignment of half-ownership of submitted code or documentation for substantive contributions to its official projects. We require the assignment because we're interested in, eventually, giving the copyright to the code to a foundation. Obtaining half-ownership of the code makes it possible for us to do this credibly without chasing people for permission to do so when that time comes.
- Assurance that the contributor will not check in incompatibly-licensed code.
- Assurance that the contributor will not allow his submission credentials to be used by a third party who may not agree to the constraints of a contribution agreement.
- Assurance that the submitted code does not infringe upon or violate the rights of a third party.
- Assurance that the contributor understands any licensing exceptions local to the repository he is contributing to.

"Signing" a contribution agreement is simple: just add your name and a date to the bottom of a "CONTRIBUTORS.txt" file found in the root of the Pylons project you'd like to contribute to. Optimally, this will be done when you submit code through GitHub (whether via a pull request from a separate repository fork, or by direct push if you have push access to the canonical project repository). Your intent to abide by the contributor agreement is signified by your commit to the "CONTRIBUTORS.txt" file with your name and a date.

Examples of "substantive" contributions:

- Submitting a new feature for review.

- Submitting artwork.
- Submitting a new chapter to documentation.

For bugfixes and other minor contributions, signing the contributor file is usually not required. However, the reviewer of a particular submission is the arbiter of whether that submission requires the signing of the contributors file.

A sample of the current contributor agreement is reproduced in full below.

```
Pylons Project Contributor Agreement
====================================

The submitter agrees by adding his or her name within the section below
named "Contributors" and submitting the resulting modified document to
the canonical shared repository location for this software project
(whether directly, as a user with "direct commit access", or via a "pull
request"), he or she is signing a contract electronically.  The submitter
becomes a Contributor after a) he or she signs this document by adding
their name beneath the "Contributors" section below, and b) the resulting
document is accepted into the canonical version control repository.

Treatment of Account
---------------------

Contributor will not allow anyone other than the Contributor to use his
or her username or source repository login to submit code to a Pylons
Project source repository. Should Contributor become aware of any such
use, Contributor will immediately notify Agendaless Consulting.
Notification must be performed by sending an email to
webmaster@agendaless.com.  Until such notice is received, Contributor
will be presumed to have taken all actions made through Contributor's
account. If the Contributor has direct commit access, Agendaless
Consulting will have complete control and discretion over capabilities
assigned to Contributor's account, and may disable Contributor's account
for any reason at any time.

Legal Effect of Contribution
----------------------------

Upon submitting a change or new work to a Pylons Project source
Repository (a "Contribution"), you agree to assign, and hereby do assign,
a one-half interest of all right, title and interest in and to copyright
and other intellectual property rights with respect to your new and
original portions of the Contribution to Agendaless Consulting. You and
Agendaless Consulting each agree that the other shall be free to exercise
any and all exclusive rights in and to the Contribution, without
accounting to one another, including without limitation, the right to
license the Contribution to others under the Repoze Public License. This
agreement shall run with title to the Contribution. Agendaless Consulting
does not convey to you any right, title or interest in or to the Program
or such portions of the Contribution that were taken from the
Program. Your transmission of a submission to the Pylons Project source
Repository and marks of identification concerning the Contribution itself
constitute your intent to contribute and your assignment of the work in
accordance with the provisions of this Agreement.

License Terms
-------------

Code committed to the Pylons Project source repository (Committed Code)
```

```
must be governed by the Repoze Public License
(http://repoze.org/LICENSE.txt, aka "the RPL") or another license
acceptable to Agendaless Consulting.  Until Agendaless Consulting
declares in writing an acceptable license other than the RPL, only the
RPL shall be used.  A list of exceptions is detailed within the
"Licensing Exceptions" section of this document, if one exists.

Representations, Warranty, and Indemnification
----------------------------------------------

Contributor represents and warrants that the Committed Code does not
violate the rights of any person or entity, and that the Contributor has
legal authority to enter into this Agreement and legal authority over
Contributed Code. Further, Contributor indemnifies Agendaless Consulting
against violations.

Cryptography
------------

Contributor understands that cryptographic code may be subject to
government regulations with which Agendaless Consulting and/or entities
using Committed Code must comply. Any code which contains any of the
items listed below must not be checked-in until Agendaless Consulting
staff has been notified and has approved such contribution in writing.

- Cryptographic capabilities or features

- Calls to cryptographic features

- User interface elements which provide context relating to cryptography

- Code which may, under casual inspection, appear to be cryptographic.

Notices
-------

Contributor confirms that any notices required will be included in any
Committed Code.

Licensing Exceptions
====================

None

List of Contributors
====================

The below-signed are contributors to a code repository that is part of
the project named "XXX".  Each below-signed contributor has read,
understand and agrees to the terms above in the section within this
document entitled "Pylons Project Contributor Agreement" as of the date
beside his or her name.

Contributors
------------

- Wile E. Coyote, 2010/11/08
```

## 6.3 Coding Style and Standards

Projects under the Pylons Project scope have rigorous standards for both coding style, testing, and documentation.

### 6.3.1 Documentation Styling

Every project needs to have documentation built with Sphinx using the Pylons Sphinx Theme for consistency.

To build documentation using the Pylons theme, add the following boilerplate near the top of your Sphinx `conf.py`:

```
import os
import sys

# Add and use Pylons theme
# protect against dumb importers
if 'sphinx-build' in ' '.join(sys.argv):
    from subprocess import call, Popen, PIPE

    p = Popen('which git', shell=True, stdout=PIPE)
    git = p.stdout.read().strip()
    cwd = os.getcwd()
    _themes = os.path.join(cwd, '_themes')

    if not os.path.isdir(_themes):
        call([git,
              'clone',
              'git://github.com/Pylons/pylons_sphinx_theme.git',
               '_themes'])
    else:
        os.chdir(_themes)
        call([git, 'checkout', 'master'])
        call([git, 'pull'])
        os.chdir(cwd)

    sys.path.append(os.path.abspath('_themes'))

    parent = os.path.dirname(os.path.dirname(__file__))
    sys.path.append(os.path.abspath(parent))
    os.chdir(parent)

html_theme_path = ['_themes']
html_theme = 'pylons'
html_theme_options = {github_url:'https://github.com/Pylons/yourprojname'}
```

Then cause the resulting `_themes` directory to be ignored in your version control system.

This will allow you to build the project utilizing the theme, and when updates are made to the theme the changes to the theme will be pulled automatically when your docs are rebuilt.

#### PDF output

Set the following values for `latex_documents` in `docs/conf.py`:

```
# Grouping the document tree into LaTeX files. List of tuples
# (source start file, target name, title, author, document class [howto/manual]).
latex_documents = [
```

```
('latexindex', 'pyramid_<project name>.tex',
'Pyramid\_<project name>',
'Author', 'manual'),
    ]
```

It is important to use _ to escape the underscore in the document title to prevent a failure in LaTeX.

Comment the following line:

```
#latex_logo = '_static/pylons_small.png'
```

Copy the folder `pyramid/docs/_static` (contains two .png files) and the file `pyramid/docs/convert_images.sh` into your `docs/` folder.

#### ePub output

Make sure you have the following value for `epub_exclude_files` in `docs/conf.py`:

```
# A list of files that should not be packed into the epub file.
epub_exclude_files = ['_static/opensearch.xml', '_static/doctools.js',
    '_static/jquery.js', '_static/searchtools.js', '_static/underscore.js',
    '_static/basic.css', 'search.html', '_static/websupport.js' ]
```

### 6.3.2 New Feature Code Requirements

In order to add a feature to any Pylons Project package:

- The feature must be documented in both the API and narrative documentation (in `docs/`).
- The feature must work fully on the CPython 2.6 and 2.7 on both UNIX and Windows and PyPy on UNIX. Most Pylons Project packages now either run or want to run on Python 3; if you're working on such a package and it already runs on Python 3.2, it must continue to run under Python 3.2 after your change. Some packages explicitly list Python 2.4 or Python 2.5 support; such support should be maintained if it exists. The `tox.ini` of most Pylons Project packages indicates which versions the package is tested under.
- The feature must not depend on any particular persistence layer (filesystem, SQL, etc).
- The feature must not add unnecessary dependencies (where "unnecessary" is of course subjective, but new dependencies should be discussed).

The above requirements are relaxed for paster template dependencies. If a paster template has an install-time dependency on something that doesn't work on a particular platform, that caveat should be spelled out clearly in *its* documentation (within its `docs/` directory).

To determine if a feature should be added to an existing package, or deserves a package of its own, feel free to talk to one of the developer teams.

### 6.3.3 Documentation Coverage

If you fix a bug, and the bug requires an API or behavior modification, all documentation in the package which references that API or behavior must change to reflect the bug fix, ideally in the same commit that fixes the bug or adds the feature.

### 6.3.4 Change Log

Feature additions and bugfixes must be added to the `CHANGES.txt` file in the prevailing style. Changelog entries should be long and descriptive, not cryptic. Other developers should be able to know what your changelog entry means.

### 6.3.5 Test Coverage

The codebase *must* have 100% test statement coverage after each commit. You can test coverage via `python setup.py nosetests --with-coverage` (requires the `nose` and `coverage` packages).

Testing code in a consistent manner can be difficult, to help developers learn our style ("dogma") of testing we've made available a set of testing notes at *Unit Testing Guidelines*.

### 6.3.6 Coding Style

All Python code should follow a derivation of PEP-8 style guide-lines. Whitespace rules and other rules are relaxed (for example, it is not necessary to put 2 newlines between classes though that's just fine if you do). Other rules are relaxed too, such as spaces around operators, and other-such. 79-column lines, however, are mandatory.

If you do use the `pep8` tool for automated checking, here is an invocation that is close to what is required by the project (subject to change):

```
pep8 --ignore=E302,E261,E231,E123,E301,E226,E262,E225,E303,E125,E251,E201,E202,E128,E122,E701,E203,E2
```

- Single-line imports

  Do this:

```
import os
import sys
```

  Do **not** do this:

```
import os, sys
```

  Importing a single item per line makes it easier to read patches and commit diffs.

  If you need to import lots of names from a single package, use:

```
from thepackage import (
    foo,
    bar,
    baz,
    )
```

- Import Order

  Imports should be ordered by their origin. Names should be imported in this order:

  1. Python standard library
  2. Third party packages
  3. Other modules from the current package

- Wildcard Imports

Do *not* import all the names from a package (e.g. never use `from package import *`), import just the ones that are needed. Single-line imports applies here as well, each name from the other package should be imported on its own line.

- No mutable objects as default arguments

  Remember that since Python only parses the default argument for a function/method just once, they cannot be safely used as default arguments.

  Do **not** do this:

```
def somefunc(default={}):
    if default.get(...):
        ...
```

Either of these is fine:

```
def somefunc(default=None):
    default = default or {}
```

```
def somefunc(default=None):
    if default is None:
        default = {}
```

- Causing others to need to rely on import-time side effects is highly discouraged.

  Creating code that requires someone to import a module or package for the singular purpose of causing some module-scoped code to be run is highly discouraged. It is only permissible to add such code to the core in paster templates, where it might be required by some other framework (e.g. SQLAlchemy "declarative base" classes must be imported to be registered).

## 6.4 Unit Testing Guidelines

The Pylons Project rather rigorously follows a unit testing dogma along the lines described by Tres Seaver in Avoiding Temptation: Notes on using unittest effectively which this document is based on.

**Note:** This document deals almost exclusively with *unit* tests. We have no particular dogma for integration tests or functional tests, although many of the tips below can be reused in that context.

### 6.4.1 Tips for Avoiding Bad Unit Tests

- Some folks have drunk the "don't repeat yourself" KoolAid: we agree that not repeating code is a virtue in most cases, but unit test code is an exception: cleverness in a test both obscures the intent of the test and makes a subsequent failure massively harder to diagnose.
- Others want to avoid writing both tests and documentation: they try to write test cases (almost invariably as "doctests") which do the work of real tests, while at the same time trying to make "readable" docs.

Most of the issues involved with the first motive are satisfactorily addressed later in this document: refusing to share code between test modules makes most temptations to cleverness go away. Where the temptation remains, the cure is to look at an individual test and ask the following questions:

- Is the intent of the test clearly explained by the name of the testcase?
- **Does the test follow the "canonical" form for a unit test? I.e., does it:**
  - set up the preconditions for the method/function being tested.

  - call the method/function exactly one time, passing in the values established in the first step.
  - make assertions about the return value, and/or any side effects.
  - do absolutely nothing else.

Fixing tests which fail along the "don't repeat yourself" axis is usually straightforward:

- Replace any "generic" setup code with per-test-case code. The classic case here is code in the setUp method which stores values on the self of the test case class: such code is always capable of refactoring to use helper methods which return the appropriately-configured test objects on a per-test basis.
- If the method/function under test is called more than once, clone (and rename appropriately) the test case method, removing any redundant setup/assertions, until each test case calls it exactly once.

Rewriting tests to conform to this pattern has a number of benefits:

- Each individual test case specifies exactly one code path through the method/function being tested, which means that achieving "100% coverage" means you really did test it all.
- The set of test cases for the method/function being tested define the contract very clearly: any ambiguity can be solved by adding one or more additional tests.
- Any test which fails is going to be easier to diagnose, because the combination of its name, its preconditions, and its expected results are going to be clearly focused.

### 6.4.2 Goals

The goals of the kind of testing outlined here are simplicity, loose or no coupling, and speed:

- Tests should be as simple as possible, while exercising the application-under-test (AUT) completely.
- Tests should run as quickly as possible, to encourage running them frequently.
- Tests should avoid coupling with other tests, or with parts of the AUT which they are not responsible for testing.

Developers write such tests to verify that the AUT is abiding by the contracts the developer specifies. While an instance of this type of test case may be illustrative of the contract it tests, such test cases do not take the place of either API documentation or of narrative "theory of operations" documentation. Still less are they intended for end-user documentation.

### 6.4.3 Rule: Avoid doctests

Doctests seem to fulfill the best of both worlds, providing documentation *and* testing. In reality, tests written using doctest almost always serve as both poor tests and poor documentation.

- Good tests often need to test obscure edge cases, and tests for obscure edge cases don't make particularly good reading as documentation.
- Doctests are harder to debug than "normal" unit tests. It's easy to "pdb" step through a normal unit test, it's much harder to do so for doctests.
- Doctests expose too many implementation details of the interpreter (such as the representation format of a class when printed). Often doctests break when interpreter versions change, and the ameliorations that allow doctests to straddle representations between versions then cause the doctest to become ugly and fragile.
- Doctests have an execution model that makes it difficult to follow many of the rest of the rules in this document.
- Doctests often encourage bad testing practice (cutting an unverified outcome of a function call and pasting it into a test suite).

### 6.4.4 Rule: Never import the module-under-test at test module scope

Import failures in the module-under-test (MUT) should cause individual test cases to fail: they should never prevent those tests from being run. Depending on the testrunner, import problems may be much harder to distinguish at a glance than normal test failures.

For example, rather than the following:

```
# test the foo module
import unittest

from package.foo import FooClass

class FooClassTests(unittest.TestCase):

    def test_bar(self):
        foo = FooClass('Bar')
        self.assertEqual(foo.bar(), 'Bar')
```

prefer:

```
# test the foo module
import unittest

class FooClassTests(unittest.TestCase):

    def _getTargetClass(self):
        from package.foo import FooClass
        return FooClass

    def _makeOne(self, *args, **kw):
        return self._getTargetClass()(*args, **kw)

    def test_bar(self):
        foo = self._makeOne('Bar')
        self.assertEqual(foo.bar(), 'Bar')
```

### 6.4.5 Guideline: Minimize module-scope dependencies

Unit tests need to be runnable even in an environment which is missing some required features: in that case, one or more of the test case methods (TCMs) will fail. Defer imports of any needed library modules as late as possible.

For instance, this example generates no test failures at all if the `qux` module is not importable:

```
# test the foo module
import unittest
import qux

class FooClassTests(unittest.TestCase):

    def _getTargetClass(self):
        from package.foo import FooClass
        return FooClass

    def _makeOne(self, *args, **kw):
        return self._getTargetClass()(*args, **kw)

```

```
    def test_bar(self):
        foo = self._makeOne(qux.Qux('Bar'))
```

while this example raises failures for each TCM which uses the missing module:

```
# test the foo module
import unittest

class FooClassTests(unittest.TestCase):

    def _getTargetClass(self):
        from package.foo import FooClass
        return FooClass

    def _makeOne(self, *args, **kw):
        return self._getTargetClass()(*args, **kw)

    def test_bar(self):
        import qux
        foo = self._makeOne(qux.Qux('Bar'))
```

It may be a reasonable trade off in some cases to import a module (but not the MUT!) which is used widely within the test cases. Such a trade off should probably occur late in the life of the TCM, after the pattern of usage is clearly understood.

### 6.4.6 Rule: Make each test case method test Just One Thing

Avoid the temptation to write fewer, bigger tests. Ideally, each TCM will exercise one set of preconditions for one method or function. For instance, the following test case tries to exercise far too much:

```
def test_bound_used_container(self):
    from AccessControl.SecurityManagement import newSecurityManager
    from AccessControl import Unauthorized
    newSecurityManager(None, UnderprivilegedUser())
    root = self._makeTree()
    guarded = root._getOb('guarded')

    ps = guarded._getOb('bound_used_container_ps')
    self.assertRaises(Unauthorized, ps)

    ps = guarded._getOb('container_str_ps')
    self.assertRaises(Unauthorized, ps)

    ps = guarded._getOb('container_ps')
    container = ps()
    self.assertRaises(Unauthorized, container)
    self.assertRaises(Unauthorized, container.index_html)
    try:
        str(container)
    except Unauthorized:
        pass
    else:
        self.fail("str(container) didn't raise Unauthorized!")

    ps = guarded._getOb('bound_used_container_ps')
    ps._proxy_roles = ( 'Manager', )
    ps()
```

```
28
29        ps = guarded._getOb('container_str_ps')
30        ps._proxy_roles = ( 'Manager', )
31        ps()
```

This test has a couple of faults, but the critical one is that it tests too many things (eight different cases).

In general, the prolog of the TCM should establish the one set of preconditions by setting up fixtures/mock objects/static values, and then instantiate the class or import the FUT (function-under-test). The TCM should then call the method/function. The epilog should test the outcomes, typically by examining either the return value or the state of one or more fixtures/mock objects.

Thinking about the separate sets of preconditions for each function or method being tested helps clarify the contract, and may inspire a simpler, cleaner, faster implementation.

### 6.4.7 Rule: Name TCMs to indicate what they test

The name of the test should be the first, most useful clue when looking at a failure report: don't make the reader (yourself, most likely) grep the test module to figure out what was being tested.

Rather than adding a comment:

```
1  class FooClassTests(unittest.TestCase):
2
3      def test_some_random_blather(self):
4          # test the 'bar' method in the case where 'baz' is not set.
5          ...
```

prefer to use the TCM name to indicate its purpose:

```
1  class FooClassTests(unittest.TestCase):
2
3      def test_getBar_wo_baz(self):
4          ...
```

### 6.4.8 Guideline: Share setup via helper methods, not via attributes of `self`

Doing unneeded work in the `setUp` method of a test case class sharply increases coupling between TCMs, which is a Bad Thing. For instance, suppose the class-under-test (CUT) takes a context as an argument to its constructor. Rather than instantiating the context in `setUp`:

```
1  class FooClassTests(unittest.TestCase):
2
3      def setUp(self):
4          self.context = DummyContext()
5
6      # ...
7
8      def test_bar(self):
9          foo = self._makeOne(self.context)
```

add a helper method to instantiate the context, and keep it as a local:

```
1  class FooClassTests(unittest.TestCase):
2
3      def _makeContext(self, *args, **kw):
4          return DummyContext(*args, **kw)
```

```
5
6       def test_bar(self):
7           context = self._makeContext()
8           foo = self._makeOne(context)
```

This practice allows different tests to create the mock context differently, avoiding coupling. It also makes the tests run faster, as the tests which don't need the context don't pay for creating it.

### 6.4.9 Guideline: Make fixtures as simple as possible

When writing a mock object, start off with an empty class, e.g.:

```
1   class DummyContext:
2       pass
```

Run the tests, adding methods only enough to the mock object to make the dependent tests pass. Avoid giving the mock object any behavior which is not necessary to make one or more tests pass.

### 6.4.10 Guideline: Use hooks and registries judiciously

If the application already allows registering plugins or components, take advantage of the fact to insert your mock objects. Don't forget to clean up after each test!

It may be acceptable to add hook methods to the application, purely to allow for simplicity of testing. For instance, code which normally sets datetime attributes to "now" could be tweaked to use a module-scope function, rather than calling `datetime.now()` directly. Tests can then replace that function with one which returns a known value (as long as they put back the original version after they run).

### 6.4.11 Guideline: Use mock objects to clarify dependent contracts

Keeping the contracts on which the AUT depends as simple as possible makes the AUT easier to write, and more resilient to changes. Writing mock objects which supply only the simplest possible implementation of such contracts keeps the AUT from acquiring "dependency creep."

For example, in a relational application, the SQL queries used by the application can be mocked up as a dummy implementation which takes keyword parameters and returns lists of dictionaries:

```
1   class DummySQL:
2
3       def __init__(self, results):
4           # results should be a list of lists of dictionaries
5           self.called_with = []
6           self.results = results
7
8       def __call__(self, **kw):
9           self.called_with.append(kw.copy())
10          return results.pop(0)
```

In addition to keeping the dependent contract simple (in this case, the SQL object should return a list of mappings, one per row), the mock object allows for easy testing of how it is used by the AUT:

```
1   class FooTest(unittest.TestCase):
2
3       def test_barflies_returns_names_from_SQL(self):
4           from foo.sqlregistry import registerSQL
```

```
        RESULTS = [[{'name': 'Chuck', 'drink': 'Guiness'},
                    {'name': 'Bob', 'drink': 'Knob Creek'},
                   ]]
        query = DummySQL(RESULTS[:])
        registerSQL('list_barflies', query)
        foo = self._makeOne('Dog and Whistle')

        names = foo.barflies()

        self.assertEqual(len(names), len(RESULTS))
        self.failUnless('NAME1' in names)
        self.failUnless('NAME2' in names)

        self.assertEqual(query.called_with, [{'bar': 'Dog and Whistle'}])
```

### 6.4.12 Rule: Don't share text fixtures between test modules

The temptation here is to save typing by borrowing mock objects or fixture code from another test module. Once indulged, one often ends up moving such "generic" fixtures to shared modules.

The rationale for this prohibition is simplicity: unit tests need to exercise the AUT, while remaining as clear and simple as possible.

- Because they are not in the module which uses them, shared mock objects and fixtures impose a lookup burden on the reader.
- Because they have to support APIs used by multiple clients, shared fixtures tend to grow APIs and data structures needed only by one client: in the degenerate case, they become as complicated as the application they are supposed to stand in for!

In some cases, it may be cleaner to avoid sharing fixtures even among test case methods (TCMs) within the same module or class.

### 6.4.13 Conclusion

Tests which conform to these rules and guidelines have the following properties:

- The tests are straightforward to write.
- The tests yield excellent coverage of the AUT.
- They reward the developer through predictable feedback (e.g., the growing list of dots for passed tests).
- They run quickly, and thus encourage the developer to run them frequently.
- Expected failures confirm missing or incomplete implementations.
- Unexpected failures are easy to diagnose and repair.
- When used as regression tests, failures help pinpoint the exact source of the regression (a changed contract, for instance, or an underspecified constraint).
- Writing such tests clarifies thinking about the contracts of the code they test, as well as the dependencies of that code.

## 6.5 Adding Features and Dealing with Bugs

Unfortunately no code is perfect, sometimes bugs will occur, or a feature is desired. When reporting bugs, being as thorough as possible, and including additional details makes a huge improvement. No one should feel discouraged in attempting to fix a bug or suggest a feature that might be missing.

### 6.5.1 Reporting a Bug

Bugs with a Pylons Project package should be reported to the individual issue tracker on GitHub. First, some general guidelines on reporting a bug.

#### 1) Create a GitHub account

You will need to create a GitHub account account to report and correspond regarding the bug you are reporting.

#### 2) Determine if your bug is really a bug

You should not file a bug if you are:

- **Proposing features and ideas**: you should follow the policy below on *Proposing Features and Ideas*.
- **Requesting support**: there are a variety of ways to request support, from the mailing list, Stackoverflow, or IRC at `#pylons` on FreeNode.

#### 3) Make sure your bug hasn't already been reported

Search through the appropriate Issue tracker on GitHub (see *Issue Trackers* below). If a bug like yours was found, check to see if you have new information that could be reported to help the developers fix it.

#### 4) Collect information about the bug

To have the best chance of having a bug fixed, we need to be able to easily replicate the conditions that caused it. Most of the time this information will be from a Python traceback message, though some bugs might be in design, spelling, or other errors on the website, docs, or code.

If the error is from a Python traceback (see a Python traceback), include it in the bug report being filed. We will also need to know what platform you're running (Windows, OSX, Linux), and which Python interpreter you're running if it's not CPython (e.g. Jython, Google App Engine).

#### 5) Submit the bug

By default GitHub will email you to let you know when new comments have been made on your bug. In the event you've turned this feature off, you should check back on occasion to ensure you don't miss any questions a developer trying to fix the bug might ask.

### 6.5.2 Issue Trackers

Bugs are reported and tracked on GitHub's issue trackers. Each Pylons Project has their own:

- pyramid issue tracker
- pyramid_beaker issue tracker
- pyramid_xmlrpc issue tracker
- pyramid_jinja2 issue tracker
- Pylons Project issue tracker (All bugs with the pylonshq.com/pylonsproject.org website should be reported here.)

### 6.5.3 Working on Code

To fix bugs or add features to a package managed by the Pylons project, an account on GitHub is required. All Pylons Project packages are under the Pylons organization on GitHub.

The general practice for contributing new features and bug fixes is to fork the package in question and make changes there. Then send a pull request. This allows the developers to review the patches and accept them, or comment on what needs to be addressed before the change sets can be accepted.

### 6.5.4 Proposing Features and Ideas

When proposing an idea that isn't just a fix or a plain bug report, the best place to do so is on the Pylons-Devel mail list . Another reasonable venue is IRC at `#pylons` on FreeNode.

## 6.6 Add-on and Development Environment Guidelines

Along with the "100% test coverage, 100% documentation" requirements of all packages that wish to be part of the Pylons Project, there are some more specific guidelines for creating add-ons and "development environments" for Pyramid. If you would like your add-on to be considered for inclusion into the *Pyramid Add-ons* or *Sample Pyramid Development Environments* sections of the Pylons Project web site, you should attempt to adhere to these guidelines.

An "add-on" is a package which relies on Pyramid itself. If your add-on does not rely on Pyramid, it's not an add-on (just a library), and it will not be listed on the add-ons page.

"Development environments" are packages which use Pyramid as a core, but offer alternate services and scaffolding. Each development environment presents a set of opinions and a "personality" to its users. Although users of a development environment can still use all of the services offered by the Pyramid core, they are usually guided to a more focused set of opinions offered by the development environment itself. Development environments often have dependencies beyond those of the Pyramid core.

Below, we talk about what makes a good add-on and what makes a good development environment.

### 6.6.1 Contributing Add-ons

Pyramid provides a repository that allows everyone to share add-ons.

Please refer to the community docs.

### 6.6.2 Making Good Add-Ons

Add-on packages should be named `pyramid_foo` where `foo` describes the functionality of the package. For example, `pyramid_mailer` is a great name for something that provides outbound mail service. If the name you want has already been taken, try to think of another, e.g. `pyramid_mailout`. If the functionality of the package cannot easily be described with one word, or the name you want has already been taken and you can't think of another name related to functionality, use a codename, e.g. `pyramid_postoffice`.

If your package provides "configuration" functionality, you will be tempted to create your own framework to do the configuration, like the following:

```
class MyConfigurationExtender(object):
    def __init__(self, config):
        self.config = config

    def doit(self, a, b):
        self.config.somedirective(a, b)

extender = MyConfigurationExtender(config)
extender.doit(1, 2)
```

Instead of doing so, use the `add_directive` method of a configurator as documented at Adding Methods to the Configurator via add_directive:

```
def doit(config, a, b):
    config.somedirective(a, b)

config.add_directive('doit', doit)
```

If your add-on wants to provide some default behavior, provide an `includeme` method in your add-on's `__init__.py`, so `config.include('pyramid_foo')` will pick it up. See Including Configuration From External Sources.

### 6.6.3 Making Good Development Environments

If you are creating a higher-level framework atop the Pyramid codebase that contains "template" code (skeleton code rendered by a user via `paster create -t foo`), for the purposes of uniformity with other "development environment" packages, we offer some guidelines below.

- It should not be named with a `pyramid_` prefix. For example, instead of `pyramid_foo` it should just be named `foo`. The `pyramid_` prefix is best used for add-ons that plug some discrete functionality in to Pyramid, not for code that simply uses Pyramid as a base for a separate framework with its own "opinions".
- It should be possible to subsequently run `paster serve development.ini` to start any `paster create` -rendered application.
- `development.ini` should ensure that the `pyramid_debugtoolbar` package is active.
- There should be a `production.ini` file that mirrors `development.ini` but disincludes `pyramid_debugtoolbar`.
- The `[server:main]` section of both `production.ini` and `development.ini` should start `paste.httpserver` on port 6543, ala:

```
[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 6543
```

- `development.ini` and `production.ini` should configure logging (see any existing template).
- It should be possible to use `paster pshell development.ini` to visit an interactive shell using a `paster create`-rendered application.
- Startup/configuration code should live in a function named `main` within the `__init__.py` of the main package of the rendered template. This function should be linked within a `paster.app_factory` section in the template's `setup.py` like so:

```
entry_points = """\
[paste.app_factory]
main = {{package}}:main
"""
```

This makes it possible for users to use the following pattern (particularly `use = egg:{{project}}`):

```
[app:{{project}}]
use = egg:{{project}}
reload_templates = true
.. other config ..
```

- WSGI middleware configuration should not be inlined into imperative code within the `main` function. Instead, middleware should be configured within a `[pipeline:main]` section in the configuration file, e.g.:

```
[pipeline:main]
pipeline =
    egg:WebError#evalerror
    tm
    {{project}}
```

CHAPTER 7

# Denials

Don't worry, none of these are actually true. We swear!

## 7.1 Pyramid Denials

A number of parties have promulgated a certain amount of misinformation regarding the **Pyramid** project. We would like to officially put to rest some of the wild rumors and extravagant myths that have been circulating in board rooms and chat rooms around the world.

- **Pyramid** is not built by aliens.
- These aliens are not telepathic. They do not look like human babies.
- **Pyramid** was not conceived from inside of a pyramid.
- The pyramid is not black. It does not shoot a beam of light out of its top.
- That beam of light is not a communications link to a more massive black pyramid that is orbiting the earth.
- This space pyramid, as it were, does not use light bending technology to make itself invisible.
- The **Pyramid** developers are not members of a shadow government.
- Neither are they performing experiments on the dark side of the moon, where they can work unobserved.
- The **Pyramid** developers do not worship an ancient crocodile god.
- The source code for **Pyramid** is not over 5,000 years old.
- The source code for **Pyramid** was not discovered carved into stone tablets in the Karnak temple complex. It was not translated to Python from hieroglyphs.
- The **Pyramid** developers do not have a giant, magic eye they use to spy on their enemies.
- **Pyramid** is not "doing something weird" to your code during application initialization.

We hope that these denials are sufficient to put to rest some of the wild speculation that has been circulating about the **Pyramid** project. We're not sure where people get such wild ideas. We hope that people will refrain from spreading such slanderous lies in the future.

## 7.2 Glossary

**Pylons Project** The main Pylons Project website.

**Pyramid** A web framework under the Pylons Project.

## 7.3 Pylons RTD

Pylons RTD is a project that serves as a home to maintain Pylons Project documentation on http://readthedocs.org.

# Index

## P

Pylons Project, **41**
Pyramid, **42**