Zope Project and Community Documentation

The Zope developer community

May 10, 2019

Contents

Zope is a free and open source web application server written in the object-oriented programming language "Python". Since its release in 1998, Zope continues to grow into many distinct applications, frameworks, libraries and tools.

CHAPTER 1

The World of Zope

1.1 The World of Zope

A Bit of History

In 1996 Jim Fulton, now Zope Corporation CTO, was drafted to teach a class on common gateway interface (CGI) programming, despite not knowing very much about the subject. CGI programming is a commonly-used web development model that allows developers to construct dynamic web sites. Traveling to the class, Jim studied all the existing documentation on CGI. On the way back, Jim considered what he didn't like about traditional CGI-based programming environments. From these initial musings the core of Zope was written while flying back from the CGI class.

Zope Corporation (then known as Digital Creations) went on to release three open-source software packages to support web publishing: Bobo, Document Template, and BoboPOS. These packages were written in a language called Python, and provided a web publishing facility, text templating, and an object database, respectively. Digital Creations developed a commercial application server based on these components and called it Principia. In November of 1998, investor Hadar Pedhazur convinced Digital Creations to open source Principia, thereby creating the foundation for the Zope application server.

In 2001, the Zope community began working on a component architecture for Zope, but after several years they ended up with something much more: Zope 3. While Zope 2 was powerful and popular, Zope 3 was designed to bring web application development to the next level.

During more than a decade Zope Corp. and the Zope Community have grown an outstanding set of products and technologies, influencing the general development of Python based Web application servers and tools.

1.1.1 Frameworks

ZCA The Zope Component Architecture provides facilities for defining, registering and looking up components. It's perfect for building enterprise applications based on loosely coupled components.

More information at the zope.component documentation and zope.interface documentation.

ZTK The Zope Toolkit (ZTK) is a set of libraries intended for reuse by projects to develop web applications or web frameworks. The ZCA is part of it.

More information at the Zopetoolkit documentation.

ZPT Zope Page Templates is Zope's templating mechanism.

More information at the http://docs.zope.org/zope2/zope2book/AppendixC.html. An alternative implementation is provided by Chameleon.

CMF The Content Management Framework (CMF) for Zope provides a powerful, tailorable platform for building content management applications together with the Zope Application Server.

More information at the CMF Product Page

Repoze Repoze integrates Zope technologies with WSGI and reusable Python middleware.

More information at http://repoze.org

1.1.2 Databases

ZODB The Z Object Database (ZODB) is a native object database, that stores your objects while allowing you to work with any paradigms that can be expressed in Python.

More information at http://zodb.org

1.1.3 Application Servers

Zope Zope is a Python-based application server for building secure and highly scalable web applications.

More information at http://zope.readthedocs.io

1.1.4 Tools

Buildout Buildout is a Python-based build system for creating, assembling and deploying applications from multiple parts, some of which may be non-Python-based.

More information at http://buildout.org

CHAPTER 2

Zope Documentation

An overview over the most important documentation resources.

2.1 Zope Documentation

This site links to various documentation sources for Zope and related software.

2.1.1 Zope

The Zope web application server, formerly called Zope 2, is under active development and nearing release 4.0.

- Main Zope documentation site
- The Zope Book
- The Zope Developers Guide
- Release notes

2.1.2 Older Zope 2 documentation

- zope_secrets/index

2.1.3 ZODB

The ZODB object database is the default storage engine for Zope.

- Main ZODB documentation site
- ZODB tutorial
- ZODB/ZEO programming guide

- API documentation
- ZODB articles

2.1.4 Zope Toolkit (ZTK)

The Zope Toolkit represents the various library packages that were created for Zope 3/Bluebream. Many of them live on in Zope itself.

- Zope Toolkit documentation

2.1.5 BlueBream

BlueBream is the the successor of the former Zope 3 web application server. It is no longer actively developed.

- BlueBream documentation
- Zope 3 API documentation (outdated)

CHAPTER 3

Zope Community

The Zope community is one of the largest and most professional open-source communities worldwide.

3.1 The Zope Community

The Zope community is one of the Largest and most professional open-source communities worldwide.

Security To report a security problem, send an email to security-response@zope.org

IRC freenode.net hosts lots of Zope and Zope products / application related IRC channels. Visit irc.freenode.net and try one of the Following channels: #zope, #zope.de, #zope3-dev #plone, #bluebream, #grok

Web sites Localized Zope related web sites, eg www.zope.de Audience / Tool / Product targeted websites, eg zope2.zope.org , bluebream.zope.org , grok.zope.org , docs.zope.org , buildout.zope.org

Planets News collections from different Zope related blogs, like Planet Plone and Planet Python .

3.2 The Foundation

3.2.1 The Zope Foundation

The Zope Foundation had the goal to promote, maintain, and develop the Zope platform, which it did successfully for many years. It is merging with the Plone Foundation and passed its responsibilities to the Plone Foundation in 2018.

3.2.2 The Plone Foundation

Our community includes the open source community of contributors to the Zope software, contributors to the documentation and web infrastructure, as well as the community of businesses and organizations that use Zope.

The Plone Foundation is the copyright holder of the Zope software and many extensions and associated software maintained in the *zopefoundation* GitHub organization at https://www.github.com/zopefoundation.

The Plone Foundation also manages the zope.org website as well as supporting infrastructure for open source collaboration.

For more information, visit https://plone.org/foundation

3.3 Developer Information

3.3.1 Intended Audience

This guide is for developers who are working **on** the various Zope-related software projects, rather than for developers who work **with** those projects' released software to build applications. Developers in the latter category should consult the relevant documentation for that software, e.g.:

- Zope Toolkit Documentation
- The Zope2 Book
- The Zope2 Developers' Guide
- Zope3 API Documentation

3.3.2 Contributor Roles

There are many different ways you might contribute to a Zope-related project. This guide tries to allow for different levels of participation in such projects. For instance:

- Susan might find a bug in the project while using the project's software in her own application.
- Later, she might volunteer to help triage and fix bugs during a bugday.
- She might attend a sprint to work on an important new feature in the project with other volunteers.
- Eventually, Susan might volunteer to assume responsibility with other team members for the ongoing maintenance and release management of the project.

3.3.3 Resources for Contributors

Browsing the Repository

Subversion Web Interfaces

The main Subversion Repository provides a web interface allowing developers to browse the repository. This interface uses the ViewCVS software; please consult the documentation there for using its features.

The Programmers of Vilnius team maintains a Trac-based repository browser.

Bazaar Branches

Many Zope-related projects have mirrors of the "trunk" of their project in Subversion imported into Launchpad as Bazaar branches. These imports provide a web interface as well, allowing developers to browse the source, the changeset history, and any outstanding merge requests. See for instance the ZTK project group's branch overview:

> https://code.launchpad.net/zopetoolkit

and the "zope.org" project group's branch overview:

https://code.launchpad.net/zope

Bug Trackers for Zope-related Projects

The Zope developer community uses the Launchpad platform for managing bug reports.

Zope Toolkit

The zopetoolkit project group on Launchpad is an "umbrella" for the various ZTK projects, each of which has its own bug tracker (see the project links on that page). You can see an overview of bugs across all `zopetoolkit` projects:

> https://bugs.launchpad.net/zopetoolkit

Zope2

Zope2 developers track bugs within Zope2 using the `zope2` project on Launchpad:

> https://bugs.launchpad.net/zope2/

Notifications from Launchpad for each new or updated issue are sent to the `zope2-tracker` mailing list:

> http://mail.zope.org/mailman/listinfo/zope2-tracker

To work on bugs in the `zope2` project, you need to be member of the Zope 2 developer team on Launchpad. Join the team:

> https://launchpad.net/~zope2-dev

Grok

The Grok developers track their bugs using the `grok` project on Launchpad:

> https://bugs.launchpad.net/grok

To work on bugs in the `grok` project, you need to be member of the Grok developer team on Launchpad. Join the team:

> https://launchpad.net/~grok-development-team

BlueBream

The BlueBream developers track their bugs using the `bluebream` project on Launchpad:

> https://bugs.launchpad.net/bluebream

To work on bugs in the `bluebream` project, you need to be member of the BlueBream developer team on Launchpad. Join the team:

> https://launchpad.net/~bluebream

Zope 3

Zope3 bugs are tracked within the `zope3` project on Launchpad:

> https://bugs.launchpad.net/zope3

Zope IRC Channels

The various Zope developer communities use several IRC channels for communication:

#zope

This channel serves the Zope2, Zope3, and ZTK developers. These developers hold a weekly half-hour meeting in the channel on Tuesdays from 15:00 - 15:30 UTC.

Logs for the channel are available: http://zope3.pov.lt/irclogs-zope/

#zope3-dev

This channel serves the Zope3 and ZTK developers. Some people prefer it over *#zope* since it's free of legacy Zope2 issues that sometimes muddle things up.

Logs for the channel are available: http://zope3.pov.lt/irclogs/

#grok

This channel serves the Grok developers.

Logs for the channel are available: http://jw.n–tree.net/irclogs/grok

#bluebream

This channel serves the BlueBream developers.

Zope Mailing Lists

Active Zope 2, Zope3, and Zope Toolkit developers should subscribe to the zope-dev mailing list.

Grok developers should subscribe to the grok-dev mailing list.

BlueBream developers should subscribe to the bluebream mailing list.

Depending on your interest you can subscribe to other Zope mailing lists as well:

> http://mail.zope.org/mailman/listinfo

Zope Development Culture

Coding Standards

As a general rule, projects in the `Zope` repository abide by the following standards:

- Code in Zope-related projects should generally conform to PEP 8 coding style. In particular, *Python code should never exceed 80 columns*. Existing code should be updated to this standard only conservatively, to ease integration of patches made against older releases.
- Project trunks should be kept in "ready-to-release" state: all unit tests pass, changelogs are kept updated, etc.
- We prefer that each project's unit tests be runnable using the default `setuptools` testrunner:

- Integration or functional tests may require a more elaborate test runner, such as the one provided by `zope.testrunner`. Most projects have built-in support for setting up this testrunner using `zc.buildout`. (see *Running Tests using zc.buildout*).
- All features and APIs should be fully documented using Sphinx.
- Solid release management, including releases to PyPI corresponding to "pristine" tags, detailed change logs, etc.

While some older projects may not be completely in line with this culture, we are committed to moving them all closer with any change. As a corollary: if you are submitting a patch to a project in this repository, and you want to expedite its acceptance, ensure that your patch maintains or improves the target project's conformance to these goals.

See these other resources on coding style in Zope projects:

- Zope3 Coding Standards
- Zope Toolkit Coding Style

Layout and Conventions

Each project should consist of a single, top-level project folder in Subversion, containing three conventional folders: `trunk`, where the majority of development work occurs, `tags`, containing the "pristine" tags made when releasing the project, and `branches`, containing both "maintenance" branches where bug fixes to a released version might be made, and "development" branches, for work which would otherwise de- stabilize the trunk.

Because we are mostly working on Python code here, the trunk and folders under the `tags` or `branches` folders are normally arranged as a `distutils` project, e.g.:

```
<directory>
- setup.py
- README.txt
- CHANGES.txt
+ docs/
  - Makefile
  - conf.py
  - index.rst
  - api.rst
  + .static/
  + .build/
  + .templates/
+ package/
  - __init__.py
  + subpacakge/
    - __init__.py
    - module.py
    + tests/
      - __init__.py
      - test_module.py
    + templates/
      - template.pt
```

Many packages place the first-level `package` directory in a `src` subdirectory inside the checkout.

Running Tests using `zc.buildout`

Most projects in the Zope repository are already configured to support building in-place and running tests using `zc.buildout`.

```
$ svn co svn://svn.zope.org/repos/main/zope.event/trunk event-trunk
$ cd event-trunk
$ /opt/Python-2.6.5/bin/python bootstrap.py
...
Generated script '/tmp/event-trunk/bin/buildout'.
$ bin/buidout
Develop: '/tmp/event-trunk/.'
...
Generated script '/tmp/event-trunk/bin/test'.
$ bin/test --all
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Ran 3 tests with 0 failures and 0 errors in 0.006 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

3.3.4 Contributing as a Non-Committer

Reporting Bugs against Zope Packages

Note: This outline needs fleshing out.

Identifying the Project which Has the Bug

- Look at tracebacks from “bottom up”.
- Look for recently updated eggs.

Writing the Bug Report

- Identify version(s) of code which manifest the bug
- Describe the symptom as clearly as possible.
- Provide a complete recipe for reproducing the bug.

Useful Patches for the Bug Report

In order of increasing helpfulness:

- Changes to non-test code which work around the problem
- Unit tests which fail against the current code
- Changes to non-test code, which pass against those tests.

Bug Triage and Workflow

- Assigning bugs
- Prioritization

  - Bugs with patches jump the queue
- Semantics of status values
  - 'wontfix' vs. 'incomplete'
- Bugs affecting more than one project
  - Status may differ.
  - Track bug in the project whose *release* will resolve it, not in dependent projects (these should be "wontfix").

Documenting Zope Packages Using Sphinx

Note: This outline needs fleshing out.

One of the most valuable contributions that non-core developers make is helping keep a project's documentation up-to-date and useful.

Each Zope packages should have its own documentation, managed within the checkout of the project (see *Contributing as a Non-Committer using Subversion* and *Contributing as a Non-Committer using Bazaar* for notes on getting checkouts / branches as a non-committer).

This documentation is maintained as a set of files in the `docs` subdirectory of the project, containing source files in "restructured text" format (see the reStructuredText Refererence) as well as control files which convert those source texts into HTML, Latex, PDF, etc., using Sphinx (see the Sphinx manual).

The top-level document is conventionally `docs/index.rst`. This document should explain the purpose of the project, and will often link to other documents outlining usage, APIs, etc.

Building the HTML Documentation

If you have Sphinx installed somewhere on your system, the `docs/Makefile` can be used to build various flavors of documentation. Assuming that the Sphinx scripts are installed in `/opt/Sphinx/bin`:

```
$ cd /tmp
$ svn co svn://svn.zope.org/repos/main/zope.event/trunk event-trunk
$ cd event-trunk/docs
$ PATH=/opt/Sphinx/bin:$PATH make html
```

After running that command, you can view the generated docs in your browser: file:///tmp/event-trunk/docs/.build/html/index.html

Re-running the `make` command after making changes to the docs lets you see the rendered HTML corresponding to your updates.

You can also use the Sphinx `doctest` extension to test example code snippets in the documentation:

```
$ PATH=/opt/Sphinx/bin:$PATH make doctest
```

Building the HTML Documentation using `zc.buildout`

Most Zope projects have support for building and running their tests using `zc.buildout`. Some of them also support building their docs using `zc.buildotu`:

```
$ cd /tmp
$ svn co svn://svn.zope.org/repos/main/zope.event/trunk event-trunk
$ cd event-trunk
$ /opt/Python-2.6.5/bin/python bootstrap.py
...
$ bin/buildout
...
$ bin/make-docs
```

If the `make-docs` script isn't included yet, then adding it would be a really useful contribution. Add a section like the following to the project's `buildout.cfg`:

```
[sphinx]
recipe = collective.recipe.sphinxbuilder
build = ${buildout:directory}/docs/_build
source = ${buildout:directory}/docs
outputs = doctest html
script-name = make-docs
extra-paths = ${buildout:directory}
```

If your project keeps its top-level package in the `src` subdiretory, the last line of that section would be:

```
extra-paths = ${buildout:directory}/src
```

Contributing as a Non-Committer using Subversion

Even without commit access to the Zope Subversion repository (see *Becoming a Contributor*), you can still use Subversion to contribute to Zope projects, using a read-only checkout of the project's sources.

How-To: Get a Read-only Subversion Checkout

There are several top-level modules in the repository - chief among them is the Zope sources - we'll use them for our example. You can browse them all at http://svn.zope.org/

Read-only access uses Subversion `svnserve` server. Here's how you can do your initial checkout.

For a Zope 2 trunk checkout:

```
$ svn co svn://svn.zope.org/repos/main/Zope/trunk Zope
```

For a Zope 2.11 checkout:

```
$ svn co svn://svn.zope.org/repos/main/Zope/branches/2.11 Zope
```

For a Zope 2.10 checkout:

```
$ svn co svn://svn.zope.org/repos/main/Zope/branches/2.10 Zope
```

etc.

For a Zope 3 trunk checkout:

```
$ svn co svn://svn.zope.org/repos/main/Zope3/trunk Zope3
```

For a Zope 3.4 checkout:

```
$ svn co svn://svn.zope.org/repos/main/Zope3/branches/3.4 Zope3
```

How-To: Get a Read-only Checkout from a Subversion Mirror

The main Zope repository can also be checked out read-only over HTTP, http://svn.zope.org/repos/main/ . E.g.:

```
$ svn co http://svn.zope.org/svn/repos/Zope2/trunk.
```

The Deutschsprachige Zope User Group (DZUG) hosts another SVN mirror http://svn.zope.de that can be checked out over HTTP. E.g.:

```
$ svn co http://svn.zope.de/Zope2/trunk.
```

Using a SVN mirror over HTTP is convenient if you are behind a firewall which blocks the `svnserve` port.

Note: Some Zope projects pull-in other parts of the repository using `svn:externals` using the `svn://` protocol. Checking out such a project over HTTP is problematic: you may even encounter timeouts.

You can also browse the official SVN repository through HTTP read-only through http://svn.zope.org/repos/main/ .

Working inside the Read-only Checkout

You should then be able to work inside your checkout, fixing a bug or adding a feature. You can use Subversion commands as normal, e.g.:

```
$ svn co svn://svn.zope.org/repos/main/zope.event/trunk event-trunk
$ cd event-trunk/
$ svn info
Path: .
URL: svn://svn.zope.org/repos/mainzope.event/trunk
Repository Root: svn://svn.zope.org/repos/main
...
Last Changed Date: 2010-03-26 16:21:39 -0400 (Fri, 26 Mar 2010)
```

Let's say you wanted ot add a bit of explanation to the `README.txt` file:

```
$ vi README.txt
...
```

Subversion knows about the changes you made:

```
$ svn stat
M      README.txt
$ svn diff
Index: README.txt
===================================================================
--- README.txt        (revision 8276)
+++ README.txt        (working copy)
@@ -6,6 +6,8 @@

 - An event publishing system

```

(continues on next page)

(continued from previous page)

```
+- BLAH, BLAH...
+
 - A very simple event-dispatching system on which more sophisticated
   event dispatching systems can be built. For example, a type-based
   event dispatching system that builds on ``zope.event`` can be found in
```

You can keep your checkout updated with ongoing changes, too:

```
$ svn up
U    docs/api.rst
U    docs/conf.py
Updated to revision 8673.
```

and you may have to deal with changes which conflict with those you have made.

However, because you are working in an anonymous, read-only checkout, you cannot commit your changes back to the repository.

```
$ svn commit -m "R00l da world."
svn: Commit failed (details follow):
svn: Authorizatino failed
```

Oops, is all your hard work in vain?

How-to: Submit a patch from your Subversion checkout

Once you have fixed the bug or added the feature in your checkout, double- check that you have touched all the bases (see *Coding Standards* and *Layout and Conventions*). All is well, the tests pass, you added documentation for your cool new feature, so it is time to submit the patch.

First, **don't** try to cut and paste the output from `svn diff` into an e-mail message or a web-browser textarea: such operations usually end up mangling the line endings or other bits of the diff, and make it impossible to apply cleanly. The maintainer who has to do reconstructive surgery on such a victim may just give up and ignore the patch.

Avoiding the cut-and-paste train wreck is straightforward: just create the patch as a file:

```
$ svn diff > /tmp/zope.event-my_cool_feature.patch
```

And then send or upload that file as an attachment: mailers and web-browsers are nearly as good at leaving attachments alone as they are at destroying sensitive inline text!

The preferred place to submit patches is to the project's Launchpad bug tracker (see *Bug Trackers for Zope-related Projects* for how to find your project's tracker). You will need to register for a Launchpad account, but you should then be able to create a new issue and upload your patch file to it.

Good titles, descriptions, and other metadata on the issue should help it get the attention of the right maintainer for the project: if you don't hear back fairly quickly, try asking on the appropriate IRC channel for your project (see *Zope IRC Channels*), or follow up to the project developers' mailing list (see *Zope Mailing Lists*).

Contributing as a Non-Committer using Bazaar

bzr is one of the current generation of distributed version control systems (DVCS).

- a five minute overview of using Bazaar.
- the Bazaar User Guide

How-to: Branch using the Bazaar mirror

Many Zope projects have a "mirror" of the trunk of their Subversion repository as a Bazaar branch on Launchpad:

> https://code.launchpad.net/zopetoolkit
>
> https://code.launchpad.net/zope

The mirror is updated periodically as commits are made to the SVN repository. You can create a branch from the URLs there as with any other web-hosted Bazaar branch:

```
$ bzr branch https://code.launchpad.net/zope.event event-trunk
```

In many cases, this can be shortened to:

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

How-to: Branch with Bazaar directly from Subversion

Recent versions of bzr and the bzr-svn plugin allow the developer to interoperate with a project whose main repository is in Subversion. Using this plugin, you can check out the branch from its native HTTP URL:

```
$ cd ~/zope
$ bzr branch svn://svn.zope.org/repos/main/zope.event/trunk event-trunk
```

Inside that branch, you can commit as usual using `bzr`, but you won't be able to `bzr push` the code back to Subversion unless you use an `svn+ssh` checkout URL, which requires obtaining commit access to the repository (see *Becoming a Contributor*).

Because all Zope projects are hosted in a centralized repository, the DVCS needs to pull down lots of information about revisions which aren't directly related to the branch you want to work on. In order to keep this overhead down, especially if you plan to work on multiple Zope projects, you can create a "shared repository" for your branches before checking any of them out:

```
$ mkdir ~/zope
$ cd ~/zope
$ bzr init-repo
$ bzr branch lp:zope.event
```

How-to: Submit a patch from your Bazaar branch

From your Bazaar branch, you can use `bzr diff` to create a patch file, and then submit it just as in *How-to: Submit a patch from your Subversion checkout*.

Bazaar has another feature, `bzr send`, which you can use to automate submitting the patch, either via e-mail:

```
$ bzr send --message="Cool feature" --mail-to=maintainer@example.com
```

or as a file to be uploaded to the bug tracker:

```
$ bzr send --message="Cool feature" -o /tmp/zope.event-my_cool_feature.bzr
```

In either case, bzr creates a "Bazaar bundle", including both the patch and extra revision metadata to ease merging your changes using bzr.

How-to: Push your Bazaar branch to Launchpad

As an alternative to uploading a patch from your Bazaar branch (or e-mailing it), you can also publish your branch to a server where it can be cloned over HTTP for others to use, as well as for review and merging by the package maintainer.

Let's assume that you have been hacking on `zope.event`, and want to publish your 'dictchannel' feature branch in hopes of landing it in the next release. Let's also assume that you have an account on Launchpad, and want to publish your branch there.

```
$ bzr launchpad-login <userid>
$ bzr push lp:~<userid>/zope.event/dictchannel
```

Replace `<userid>` with your Launchpad account ID.

How-to: Request a merge

After pushing your branch, you can include its URL in an e-mail you send to the maintainer, requesting a merge of your branch. You can also link your branch to a Launchpad issue, as well as using Launchpad's "merge request" feature to alert the maintainer(s) that your branch is ready to merge.

3.3.5 Becoming a Zope Committer

Becoming a Contributor

Applying for Committer Status

Detailed instructions for gaining commit rights to the Zope repositories are at https://plone.org/foundation/contributors-agreement/contributors-agreement-explained.

1. Print, fill out and sign the `Plone Contributor Agreement`. On page 2 enter *Zope* in the field for *program*.
2. Scan the agreement and submit it by email to agreements@plone.org
3. You will be added to the GitHub repository shortly.

Subscribe to the Mailing Lists

Please subscribe to the *Zope Mailing Lists* which correspond to your area(s) of interest.

Writable checkouts

Below are instructions for hooking up with our Public Subversion repository with checkin ability.

Before you start

Make sure you have *registered as a committer*.

Doing checkouts (Linux, MacOS-X, Un*X)

Any time ssh makes a key connection for subversion it will require your key passphrase. You can use ssh-agent to stash that key once for your shell, and not have to specify it again while you're issuing commands from the same shell. It's worthwhile getting acquainted with ssh-agent - check the man pages.

Now you're ready to do a checkout. The best way to convey the specifics is with an example:

```
% svn co svn+ssh://username@svn.zope.org/repos/main/Zope/trunk Zope
```

You should substitute the zope.org account name by which you are registered.

Subversion commands using SSH this way will require you to provide the passphrase for the key being invoked - you will be prompted for it unless you have ssh-agent taking care of that for you.

Doing checkouts (Windows)

Some of the command line access methods that work on non-Windows systems don't work on Windows. The TortoiseSVN project also offers a very nice integration of svn commands into the Windows Explorer GUI.

The easiest way to set up both (command line and TortoiseSVN) for svn+ssh access is to first download the popular PuTTY set of connection tools for Windows. Then, as a one-time setup cost, run 'putty.exe' to create a new PuTTY session for Zope svn+ssh access:

- Under Session, use Host Name 'svn.zope.org', and select the SSH protocol.
- Under Connection, put your zope.org username in the "auto-login username" box.
- Under Connection -> SSH -> Auth, enter the path to your private key file (whether generated by 'puttygen.exe', or otherwise).
- Back under Session, save the session under some unique name. For example, 'svnzope'. Do note that 'svn.zope.org' can be used as the name! The examples here do not, just to make the distinction clear, but setting the name to 'svn.zope.org' can be a good idea, especially if you run on Windows and Linux (simply because it's less confusing if you can type the same strings on all your platforms).
- Click "Open". You should then be asked to accept the server's key. Do so, then log out.
- Close 'putty.exe'.

  PuTTY saves this config info in the Windows registry, where other programs can get at it via the session name you chose.

  Now when using any TortoiseSVN action where a svn+ssh 'svn.zope.org' URL is needed, just use 'svnzope' (or whatever name you picked for your session) instead of 'svn.zope.org' For example:

  ```
  svn+ssh://svnzope/repos/main/ZConfig/trunk
  ```

  instead of:

  ```
  svn+ssh://svn.zope.org/repos/main/ZConfig/trunk
  ```

  If you have an SSH passphrase, you can also run PuTTY's 'pageant.exe' to supply it for you for as long as you leave 'pageant' running ('pageant' is like 'ssh-agent' on non-Windows systems).

  For command-line access, first set environment variable 'SVN_SSH' to the path to PuTTY's 'plink.exe' (use forward slashes instead of back slashes in the path or it won't work) – or you can set this once in your local svn config file. Look for the definition of 'ssh' in it, which is commented out by default. Uncomment and edit so that this section looks like:

```
[tunnels]
ssh = $SVN_SSH plink.exe
```

Then, again, use the name of your saved PuTTY session instead of ‘svn.zope.org’ in svn command lines that need to reference the repository explicitly.

CHAPTER 4

Zope Projects

Zope community projects are hosted in the Zope Foundation organization on GitHub.