



My Year at Zest

Maurits van Rees
Student number 0545701
Class dua6a/dua7a
Hogeschool Rotterdam

January 15, 2007



My Year at Zest

Preface

This report spans the period of May 2006 till January 2007. During that time, I was working for Zest Software in Hoogvliet, The Netherlands. This was part of the “dual route” of my study Informatics at the Rotterdam Institute for Informatics Studies (RIVIO) of the Hogeschool Rotterdam. I worked four days a week for Zest, and studied on Fridays.

Zest Software is a Plone company, so my work there is mostly in Plone and in Zope and Python as underlying technologies. But other subjects pop to the surface from time to time, like subversion, LDAP and Linux administration in general.

The main goal of writing this report is to convince Hans Manni from the Hogeschool Rotterdam that I learned a lot and worked hard these past months at Zest Software and that I should receive a good number of study points. But I like to give some extra zest to this report by making it relevant for a bigger audience, specifically Plone developers. I will therefore focus on three subjects that I think give a broad overview of what I have been doing and are also interesting to the general Plone community.

Just like in my last report, I can say that I have learned and laughed¹ a lot at Zest Software and I thank my colleagues for that. I look forward to working on my final study assignment with them.

¹ This is illustrated nicely by the picture on the front page, made by Esther Ladage at the end of an office trip. I think that is a glass of Grimbergen I am enjoying there.

Contents

1	Introduction	1
1.1	Crash course	1
1.2	Structure	2
1.3	Subversion	2
1.4	ldapconfig	2
1.5	Zope 3	3
2	Importing your product in the collective	4
2.1	Introduction	4
2.2	Simple export	4
2.3	Code dumping	5
2.4	Filtering the dump file	5
2.5	Testing your dump file	6
2.6	Loading in the collective	7
2.7	Problems with svndumpfilter	7
3	ldapconfig: connecting Plone and LDAP	9
3.1	Introduction	9
3.2	Goal	9
3.3	Strategy	9
3.4	Install	9
3.5	Dependencies	10
3.6	Which plugin interfaces do we want to activate?	10
3.7	Notes for developers of ldapconfig	12
3.8	Bug reports	13
3.9	Disclaimer	13
4	Embrace and Extend: The Zope 3 Way	14
4.1	Use case: extending Quills	14
4.2	Brave New Content Type	15
4.3	The olden days of Zope 2	15
4.4	Zope 3: A New Hope	15
4.5	General Strategy	16
4.6	Quills Strategy	16

4.7	Event handler	17
4.8	Utility	18
4.9	Current Status	20
4.10	Adapter	20
4.11	Annotations	22
4.12	Adding Annotations	23
4.13	Viewing Annotations	23
4.14	Conclusions	24
5	Conclusion	26
A	Subversion import script	27
B	Makefile	32
C	Main file for this report	34
D	Helper script for reStructuredText and L^AT_EX	36

Abstract

This report focuses on three subjects that together paint a broad picture of my work at Zest Software. They should be interesting for both my teacher at school and for the Plone community.

After the introduction, the first subject is subversion. [chapter 2](#) (Importing your product in the collective) shows how to move the code of your product from your own subversion repository to the Plone collective (or any other repository) including all the history.

In [chapter 3](#) (ldapconfig: connecting Plone and LDAP) we take on a subject that is always good for a lot of questions on Plone mailing lists. Connecting with LDAP (or Microsoft Active Directory) is certainly doable, but there are some gotchas. To make it easier for us and for Plone developers in general to get this to work, we made ldapconfig, a product that tries to reduce the complexity of this subject to a relatively simple configuration file. This product is presented in the third chapter.

Last but not least is a tutorial on Zope 3 technologies: [chapter 4](#) (Embrace and Extend: The Zope 3 Way). Slowly but surely Zope 2 is being replaced by the newer and cleaner Zope 3. This makes it possible to extend existing Zope (or Plone) products in a very clean, non-intrusive way, that is future-proof.

By personal convention and as service to the reader, I have included the most important files that I created for this report itself in the appendices. [Appendix A](#) has the subversion script mentioned in [chapter 2](#) (Importing your product in the collective) [Appendix B](#) shows the Makefile that lets me create the report by simply typing *make*. [Appendix C](#) presents the *main.txt* file that puts all content at the right place in the report, using a rather strange combination of ?? (reStructuredText) and L^AT_EX , but it does the trick. [Appendix D](#) prints the python script that is called in the Makefile to change some of the code that *rst2latex* creates from the *main.txt* file.

Chapter 1

Introduction

1.1 Crash course

Most chapters of this report assume at least some knowledge of Plone and related technologies like LDAP. Readers who are not that familiar with these technologies may benefit from some introductory remarks.

Plone Content Management System, useful for websites that need easy, through-the-web editing and a lot of flexibility. Runs on top of Zope.

Zope Web Application Framework, useful on its own or as a basis for big systems like Plone. Written in Python. Zope comes in two versions.

Zope 2 This is the most widespread version. Plone needs this. Internally, however, more and more functionality is gradually taken over by Zope 3, which is shipped as part of Zope 2. That bears repeating: Zope 3 is now an integral part of Zope 2.

Zope 3 Next generation of Zope. The code of Zope 3 itself is far cleaner and saner than that of the previous version. Third party programs based on Zope 3 (or Zope 2 programs partly using Zope 3) also tend to be cleaner and better structured than pure Zope 2 applications. Plone is also using Zope 3 technologies more and more.

Python Dynamic, object-oriented programming language; preferably any programs you make with this are stored in, for instance, subversion.

Subversion Version control system, comparable with cvs (Concurrent Versioning System). At the minimum this is useful when you want to go back to a previous version of some program or documentation.

collective Subversion repository for third party products for Plone.

LDAP Lightweight Directory Access Protocol, a way of storing information about (in most cases) users on a central system. Of course there are differences between Microsoft's ActiveDirectory and other implementations.

1.2 Structure

The next three chapters all consist of content written by me that can already be found somewhere on the Internet. I did not want to copy those texts, change them to fit the demands of this report and then end up with two versions of basically the same text that I may want to keep in sync. So I have chosen to present the content in those chapters in their original form and add some explanatory remarks in this introductory chapter.

As a bonus, this is a nice way of experimenting with `??` (reStructuredText). All those documents have that markup. This can be turned into html code for displaying on the Web (Plone can do this automatically for you) and into L^AT_EX code for generating nice reports.

1.3 Subversion

In [chapter 2](#) (Importing your product in the collective) you will find a tutorial that has been placed on the documentation section¹ on the plone.org website. It was inspired by the wish to put the RealEstateBroker² product in the Plone collective. It was originally in the Zest subversion repository (and for the moment it still is), but there was interest from other developers to contribute to this product. Putting it in the collective is then the best choice, as will be explained in the chapter. The same technique will be used for getting eXtremeManagement into the collective.

1.4 ldapconfig

For the website of a big customer in Eindhoven³ we needed to connect Plone with an LDAP server, or actually an ActiveDirectory server. We had already done that for another project of that same customer. So the configuration was more or less known already. But the first project was using Plone 2.1 and the new project Plone 2.5. Lots of changes had meanwhile been made to the parts of Plone (and underlying products like LDAPMultiPlugins) that are used for user authentication. This meant that our current code that worked with the previous version of Plone had to be changed.

We decided to make a configuration product⁴ that would make this easier for us and other Plone developers. Read all about it in [chapter 3](#) (ldapconfig: connecting Plone and LDAP).

¹<http://plone.org/documentation/tutorial/importing-product-into-collective>

²<http://plone.org/products/realestatebroker>

³<http://plone.net/sites/philips-research>

⁴<http://plone.org/products/ldapconfig>

1.5 Zope 3

When you want to extend a current Zope or Plone product, you have a few options. The cleanest way to do this, if it is possible, is to use Zope 3 technologies. Like was mentioned in the beginning of this chapter, Zope 3 is available even when you actually use Zope 2. It is a clean technology, but is not always easy to understand. Developers could use some good examples so they see what the possibilities are and how they can use this advanced techniques today.

For me, `feedfeeder`⁵ was such an example. It is a product that we at Zest Software made, with the help of Rocky Burt⁶. With `feedfeeder` you can automatically add content from another site to your own site, using atom feeds. But actually the specifics of this product do not matter here. The importance of `feedfeeder` for this report is that it is by default useful for the general public, but that it can also easily and cleanly be extended in a different product for a customer. For extending it you use Zope 3 techniques.

I tried my hand at creating such a product myself. Read in [chapter 4](#) (Embrace and Extend: The Zope 3 Way) how I extended Quills, a weblog product for Plone, to fit the needs of my own blog. I presented this in one of the weekly presentations in the Zest office.

⁵<http://plone.org/products/feedfeeder>

⁶<http://serverzen.net/weblog/archive/2006/09/17/keeping-client-concerns-separate>

Chapter 2

Importing your product in the collective

There are two strategies for introducing a new product in the plone collective when you already have it in your own repository: a simple svn export or a dump file that is loaded by the plone.org admins. This tutorial explains the differences and shows how to do both.

2.1 Introduction

Description of the problem this tutorial handles

You are a developer at Spacely Sprockets. You have developed a Plone product called plockets. Your company is using it internally for its sprocket inventory. You are still sane, so apparently you have used a code repository. We assume that it is subversion.

You have told some others about your product. You even gave them read access to your repository. Some are now making patches and are asking for write access. You decide that you do not want to deal with that, so you want to move your code to the Plone collective. Resistance is after all still futile.

2.2 Simple export

The simplest way to import your product is by making a clean svn export.

Your company repository lives on a server and your product can be found at <http://spacely.space/svn/plockets>. One option is to simply make a subversion export of your product, and importing it in the collective:

```
svn export --ignore-externals http://spacely.space/svn/↵
  plockets plockets
svn import plockets https://svn.plone.org/svn/collective/↵
  plockets
```

One major **downside** when using a simple export is that you lose all subversion history information. If you are the only developer who worked on pockets or there are not so many files and revisions, this may not matter. But the history can give important information about why a change was made, especially in the revision log.

The **good** thing about an export is of course that it is by far the simplest method. Two simple commands and you are ready. And this is something that you can do without help from the administrators at plone.org.

If the history of your product is important, then you have the option of making a dump of the part of the subversion repository that has your pockets product in it.

2.3 Code dumping

How to make a dumpfile of your complete repository

First you make a dump file of the repository. You could play around on your company server, but if something goes wrong, Mr Spacely will be angry with you and you do not want that. So to be on the safe side, you replicate the server repository on your own computer.

You might be able to use svk for this, if you are used to it. I will not cover that here. If the name does not ring a bell, never mind.

Instead I assume that you actually have access to that server and are allowed to use the svnadmin command. So you login to that server and (quietly if you want to) dump the complete repository to a dump file. That would look something like this:

```
svnadmin dump --quiet /var/lib/svn/spacely > spacely.dump
```

Then you copy that file to your own computer and you can now safely play with it without having to say sorry to Mr Spacely when you make a mistake that kills the complete repository. Just a precaution.

2.4 Filtering the dump file

Getting only your project in a dumpfile

You now have a spacely.dump file on your computer. But that contains the complete repository. Now you want to create a new dump file that only contains the pockets product. You do that with the svndumpfilter command:

```
cat spacely.dump | svndumpfilter --drop-empty-revs --↵  
  renumber-revs \  
--quiet include pockets > pockets.dump
```

This does the following:

- you pipe the complete dump file to svndumpfilter,
- you only include revisions that contain changes to pockets,

- you drop (ignore) any revisions that have nothing to do with pockets,
- you renumber the revisions (this is recommended when you use the previous step),
- you do this quietly, though when you do this the first time you probably want to see what happens, so remove the `--quiet` part,
- you put the result in a new dump file `pockets.dump`.

Several things could go wrong here, but we assume for now that this command succeeds without warnings or errors. If you run into problems, then consult [section 2.7](#) (Problems with `svndumpfilter`).

2.5 Testing your dump file

How to make sure that your dumpfile is correct

During testing it may be handy to have the complete Spacely repository available on your own machine instead of just via the dump file. If you want to recreate the complete Spacely repository, you can do that like this:

```
svnadmin create /var/lib/svn/spacely
svnadmin load --quiet /var/lib/svn/spacely < spacely.dump
```

You can use these same commands for testing your new pockets dump file. Create a temporary subversion repository on your machine and load the sprockets product:

```
svnadmin create /var/lib/svn/temp
svnadmin load --quiet /var/lib/svn/temp < spacely.dump
```

This should just work. But you probably want to be certain that the resulting repository has exactly the same information as the original sprockets part of the spacely repository.

One simple way is to create svn exports of both repositories and compare them. If you have `svn:externals` in your repository, then it is best to ignore those. Create and compare two exports like this:

```
# Make an export from the Spacely repository:
svn export --ignore-externals --quiet http://spacely.space↵
  /svn/pockets pockets.ori
# Alternatively, make an export from your local Spacely ↵
  repository:
svn export --ignore-externals --quiet file://var/lib/svn/↵
  spacely/pockets pockets.ori
# Make an export from your local temporary repository:
svn export --ignore-externals --quiet file://var/lib/svn/↵
  temp/pockets pockets.new
# See the differences between those two exports:
diff -r pockets.ori pockets.new
```

This should not give a lot of lines of output. You may see some output like this:

```
3c3
< # $Id: config.py 551 2006-09-19 13:04:49Z sashav $
---
> # $Id: config.py 507 2006-09-19 13:04:49Z sashav $
```

That is okay. That is just a part of a file that is automatically adjusted by subversion based on the version number in the repository. If you see other differences then that is an indication that something went wrong. You need to investigate then.

2.6 Loading in the collective

Get your dumpfile noticed by the plone.org admins

You can now send the pockets dumpfile to the administrators at plone.org. The best way to do that is to file a ticket in the plone.org tracker¹. See the example for Importing RealEstateBroker² into the collective.

You can upload your dump file to the ticket by using the Attach File button. Note that this button only becomes visible *after* you submit your ticket.

2.7 Problems with svndumpfilter

When you run into problems when using svndumpfilter don't panic.

2.7.1 Unsupported dumpfile version

Some versions of svnadmin and svndumpfilter do not work nicely together. When you pipe a dump file created by svn through svndumpfilter you will then see this error:

```
svn: Unsupported dumpfile version: 3
```

This means that svndumpfilter expects a dumpfile of version 2 and does not know how to handle version 3. Then it may help to first load the server dump file in your local repository like mentioned above. If you do not get it to work, try upgrading to a newer version of your subversion packets, or consult the internet. For example, svndumpfilter vs svnadmin³ and version inconsistency⁴.

2.7.2 Invalid copy source path

Another problem can occur when you include or exclude a path and one revision has changes on files that are both inside and outside the path that you want to keep. For

¹<https://dev.plone.org/plone.org/>

²<https://dev.plone.org/plone.org/ticket/297>

³<http://svn.haxx.se/users/archive-2005-09/0280.shtml>

⁴<http://svn.haxx.se/dev/archive-2004-11/0805.shtml>

instance, I tried excluding the integration directory when making a dump for the eXtremeManagement product as it did not make sense to include that in the dump file. Then I saw this:

```
Revision 369 committed as 355.  
Revision 370 skipped.  
svndumpfilter: Invalid copy source path '/  
  eXtremeManagement/integration'
```

This means that the following is the case:

- revision 369 is fine;
- revision 370 is skipped because the excluded path was referenced there;
- revision 371 causes problems, as it changes something both inside and outside the excluded path: probably you copied something from an excluded path to an included path.

In my case, this was the problem:

```
$ svn log -r 371 file:///var/lib/svn/temp/ -v
```

```
-----  
r371 | maurits | 2006-02-23 15:21:18 +0100 (do, 23 feb   
 2006) | 1 line  
Changed paths:  
  D /eXtremeManagement/integration  
  A /eXtremeManagement/temp (from /eXtremeManagement/  
  integration:370)
```

Moved integration revision 4068 to temp.

So when you use svndumpfilter and run into this problem, you need to manually work around that. I included a script in this tutorial that does that. It also does some checking. Perhaps it is of use to others. You cannot use it off the shelf. You need to adapt it to your own needs. If you use it unthinkingly on your server, Mr Spacely could get very upset. (For the script, see [Appendix A](#))

Chapter 3

ldapconfig: connecting Plone and LDAP

3.1 Introduction

Zest Software¹ proudly presents ldapconfig, a configuration product for setting up a connection between Plone and an LDAP Server.

3.2 Goal

This product is here to make a connection between Plone and an LDAP server. That LDAP server may also be an Active Directory server.

3.3 Strategy

The product first loads the config from its own ldapconfig.py. Then it searches for an ldapconfig.py file in the etc/ dir of your Zope instance. Any settings in there will override the previously found settings.

Warning: you specify an id for your LDAPUserFolder in the LDAP_PLUGIN_ID setting. Any existing LDAPUserFolder with this id will first be **removed** and then readded with your new settings! If you are unsure that you want that, you should make a backup. You should make a backup anyway, even if you *are* sure.

3.4 Install

- Put it like any other Plone Product in the Products dir of your instance.
- Copy the ldapconfig.py.example file to <zope-instance>/etc/ldapconfig.py

¹<http://zestsoftware.nl>

- Adapt that file to fit your setup.
- Install ldapconfig via the QuickInstaller.

3.5 Dependencies

- Plone 2.5 with PlonePAS (we may get this to work for Plone 2.1 without PlonePAS as well, but not now).
- LDAPUserFolder (using LDAPUserFolder-2.8-beta.tgz now)
- LDAPMultiPlugins: Using a patched LDAPMultiPlugins-1.4.tgz now. See the patch instructions in docs/PLUGINS.txt. You will find information there that helps you decide which version you want and whether you need the patch or not.

3.6 Which plugin interfaces do we want to activate?

With PlonePAS and LDAPMultiPlugins you can and should specify which functions (called plugin interfaces) you want your LDAP to perform. Do you just want authentication or do you also want to add users to LDAP from within Plone? Things like that.

Below is a description of the various Plugin interfaces that are defined in PluggableAuthService or PlonePAS.

3.6.1 Patch for LDAPMultiPlugins

Some plugin interfaces are marked below as: “needs patch”. That means it only works with LDAPMultiPlugins patched with this patch².

Many thanks go to Wichert Akkerman for creating this patch. Note that this patch currently stands no chance of being included into the LDAPMultiPlugins source code itself, as it depends on PlonePAS and so it depends on Plone. Jens Vagelpohl (who is co author of LDAPMultiPlugins together with Chris McDonough) wants it to be usable in a pure Zope site. Many thanks go to Jens and Chris too, of course!

3.6.2 Patch instructions

This patch only works with the latest subversion checkout of LDAPMultiPlugins. At the moment of writing, that is revision 1378. If meanwhile LDAPMultiPlugins has a later revision number and the patch fails, you can try it with revision 1378 and it should still work. Please contact the plone-users mailing list or the authors of this ldapconfig product or Wichert Akkerman, to inform them of the problem. Go to the shell and do:

²<http://antiloop.plone.org/LDAPMultiPlugins-plone.org.patch>

```
# Get LDAPMultiPlugins 1.4:
svn co http://svn.dataflake.org/svn/LDAPMultiPlugins/trunk↵
    LDAPMultiPlugins
# Get the patch:
wget http://antiloop.plone.org/LDAPMultiPlugins-plone.org.↵
    patch
# Change to the subversion checkout dir
cd LDAPMultiPlugins
# Apply the patch:
patch -p0 < ../LDAPMultiPlugins-plone.org.patch
```

This should give the following outcome:

```
patching file LDAPPluginBase.py
patching file LDAPMultiPlugin.py
patching file property.py
```

3.6.3 Plugin Interface descriptions

Now we give a short description of the available plugins and say if it needs a patch or needs LDAPMultiPlugins version 1.3 (or higher). If nothing is said, it works with version 1.2 (and should also work with 1.3). If you have a completely patched LDAPMultiPlugins by the instructions above, all plugins will be available to you.

3.6.4 From Products.PluggableAuthService.interfaces.plugins

- IAuthenticationPlugin: Map credentials to a user ID.
- ICredentialsResetPlugin: Callback: user has logged out.
- IGroupEnumerationPlugin: Allow querying groups by ID, and searching for groups.
- IGroupsPlugin: Determine the groups to which a user belongs.
- IPropertiesPlugin: Return a property set for a user.
- IRoleEnumerationPlugin: Allow querying roles by ID, and searching for roles.
- IUserEnumerationPlugin: Allow querying users by ID, and searching for users; also needed for listing users in a group.
- IRolesPlugin: Determine the (global) roles which a user has.
- ICredentialsUpdatePlugin: Callback: user has changed her password. Needs LDAP-MultiPlugins 1.3
- IUserAdderPlugin: Create a new user record in a User Manager. Needs patch.

3.6.5 From Products.PlonePAS.interfaces.group

- IGroupManagement: add, update, remove, set roles for group. Needs patch.
- IGroupIntrospection: get group ids etc, get members of group.

3.6.6 From Products.PlonePAS.interfaces.plugins

- IUserManagement: Manage users (change password or delete user) Needs patch.

3.7 Notes for developers of ldapconfig

Here are some pointers for developers wanting to contribute to this product.

3.7.1 ldapconfig on plone.org

ldapconfig has a product page³ on plone.org. Most of the text there comes from the documentation here on the file system. reStructuredText is used for that. So if you want to update some text on the product page, please change it in the subversion code first and then upload the changed text.

3.7.2 Online documentation

- Howto⁴: Plone 2.5 and OpenLDAP Integration for Users and Groups
- User experience⁵: Getting Plone 2.5 / PlonePAS working with LDAP

3.7.3 What is being done in this product?

We want to get Plone and LDAP to communicate nicely together. The following actions are needed then.

- Add our LDAP server to PlonePAS with an LDAPMultiPlugin.
- Tell PlonePAS which interfaces we want to activate. In other words: do we want to use LDAP only for authentication or also for adding users, managing groups, etcetera. Actually, at this point we activate them all; later we can switch some off.
- Move our LDAP plugin to the front of the list for a few interfaces where we want LDAP to be the first plugin that is consulted.

³<http://plone.org/products/ldapconfig>

⁴<http://plone.org/documentation/how-to/plone-2-5-and-openldap-integration-for-users-and-groups>

⁵<http://www2.le.ac.uk/Members/nd51/blog-rdf-pkb-9jd/blogentry.2006-06-22.7848776335>

3.7.4 Ideas for the future

It is a hassle to checkout LDAPMultiPlugins and apply the patch. It would be far easier if we could just add that patch to this ldapconfig product and apply it on the fly. We can do that via monkey patching or better via the patching technique used in for instance the CacheSetup product. If that works, then we can simply instruct users to download LDAPMultiPlugins and ldapconfig, install them and they are ready.

3.8 Bug reports

An issue tracker⁶ has been added to our product page at plone.org. Only possible bugs or feature requests for this package should be reported here. This is NOT meant for problems with LDAPUserFolder itself or LDAPMultiPlugins or PlonePAS or your ldap or active directory server. The distinction will probably be difficult, so we will not yell at you. :) You can always try the plone-users mailing list.

3.9 Disclaimer

Using this product is no substitute for actually understanding at least the basics of LDAP servers and Plone user management in general. Knowledge about LDAPMultiPlugins, LDAPUserFolder and PlonePAS is also helpful if you get into trouble. This is just an integration product. It has no way of knowing what the correct settings for your situation are, unless you tell it. The example config file should be very helpful though, so don't despair. :)

⁶<http://plone.org/products/ldapconfig/issues>

Chapter 4

Embrace and Extend: The Zope 3 Way

4.1 Use case: extending Quills

I have a weblog¹. At the time of writing this is a hand-made Zope blog, consisting entirely of page templates and scripts, which sit in the Zope Database. It works, it is fast, but it is kludgy. I have to do too many things before I can publish a new weblog entry. So there is room for improvement. Plus several other weblogs are available in Plone. Quills seems to be good, and I know a few persons who also use it, for instance my brother², so I choose to use that too.

Actually, I have two weblogs at my site. One is a general blog where I post personal things or entries relating to programming. The other is a completely Dutch blog³ for my church⁴. Actually it is more of a podcast. It contains small entries with a link to an external mp3 file containing a complete service of my church, and bigger entries with a link to a local mp3 file with just the sermon in case our pastor Erik was leading this service. So one service could actually have two entries. Again, it works, it is fast, but it is kludgy.

For this second blog, Quills seems like a reasonable choice as well. But to really make it useful, I would want two extra fields for the audio blog, that can contain links to those mp3 files. For the sermon it could be even nicer to be able to upload this file via Quills and have the link correctly made. But that may be added later. For now, I am happy with just two extra links in a Quills WeblogEntry.

¹<http://maurits.vanrees.org/weblog>

²<http://vanrees.org/weblog>

³<http://maurits.vanrees.org/preken>

⁴<http://www.herbergonderweg.nl>

4.2 Brave New Content Type

Alright, let's create a totally new content type, shall we? With ArchGenXML⁵ we can easily create a basic new content type, let's call it `AudioEntry`, that has the exact attributes that we need. Then again, other people have already thought a lot about what a weblog entry should contain and what it should do. We would introduce yet another weblog application, which Plone does not really need and that is currently only maintained by us. There are certainly occasions where this is a splendid option, but not here.

4.3 The olden days of Zope 2

Our second option is to extend Quills in the Zope 2 style: we create a class `AudioEntry` that inherits from the Quills `WeblogEntry`. If our class really offers something new, this is a fine choice. A good example is `RichDocument`. This extends the Schema of `ATDocument` to add a few attributes. The class inherits from `ATDocument` and so it uses the functions already defined for that basic `ATDocument` and adds a few functions that make sense for `RichDocument`. This is very well documented in *Extending ATContentTypes*⁶ by Martin Aspeli.

What would that mean for our use case of *Extending Quills*? `AudioEntry` would inherit from `WeblogEntry` and extend it with two attributes for the links. But it would not stop at that. For starters, a Quills Weblog can currently only contain `Folders` and `WeblogEntries`. Adding our `AudioEntry` there is not possible out of the box. Of course this is changeable, but then we might also have to override the `Weblog` class with our own `AudioWeblog` class. This can easily get out of hand and result in a cascade of overrides. Also, when someone else has a similar idea and introduces a `VideoWeblogEntry`, these two extension products would be incompatible.

There are certainly use cases like `RichDocument` where this Zope 2 way makes sense. But for our use case we simply want to add two links. Is there no easier and less intrusive way to do this?

4.4 Zope 3: A New Hope

When we extend a product in Zope 3 style, we adapt an existing content type without making a new content type. In our case this means that we only need to adapt `WeblogEntry`, without the need for changing the `Weblog` class. Also, we can arrange that our adaptations work for the `VideoWeblogEntry` class as well, as long as that class plays nice and implements the `IWeblogEntry` interface that Quills defines.

So already we can see that the Zope 3 way indeed brings new hope, in that our changes can be much less intrusive and much more compatible with other changes.

⁵<http://plone.org/products/archgenxml>

⁶<http://plone.org/documentation/tutorial/richdocument/extending-atct>

We will be using and introducing the following Zope 3 technologies:

- events
- utilities
- adapters
- annotations

4.5 General Strategy

We are going to make two products: one general product for extending an object when it fulfils some condition, and one specific product for extending a Quills WeblogEntry when it has audio content. The strategy for the general product is this:

- An **event** is handled by an event handler, which uses a...
- **utility** to decide if an...
- **adapter** is necessary, in which case...
- **annotations** are added to the original object.

So: when an event takes place, we ask a utility if this event warrants an adaptation of an object. The adaptation is implemented by adding annotations to that object.

This is implemented by the **keywordannotator** product. And this strategy probably has most people blinking their eyes and rereading that list. :) Don't panic. You probably *do* want to reread it, but if you do not fully understand it, you can just go to the next section, which will translate this general idea into a specific strategy for our use case. It should be much clearer then.

4.6 Quills Strategy

Do you remember the four Zope 3 technologies that we planned on using? Events, utilities, adapters and annotations. This is how our specific strategy uses those:

event Someone adds a keyword *audio* to a WeblogEntry. When this happens, Zope fires a so-called IObjectModified event. We will register an event handler that springs into action when a WeblogEntry (or actually any object that claims to implement the IWeblogEntry interface) is modified or added.

utility The code that handles this event, calls a utility. This utility looks at the WeblogEntry and decides that the addition of this keyword means that this entry now also implements the marker interface IMAudio that we will define. If an object implements or provides a 'normal' interface, then this object has all the functions

and attributes that are defined by that interface. A marker interface has no functions or attributes. So it is basically just a label: the object is marked as being `IMaudio`.

We also make sure that we can later add annotations to this entry by letting it provide the standard `IAttributeAnnotatable` interface defined by Zope 3.

adapter We can now actually adapt (extend) this object that implements `IMaudio` and add annotations to it. Adapting basically does the same that inheritance (the Zope 2 way) does, but with a different technique. You temporarily put a *wrapper* around an object, do something to it (in our case add annotations) and then remove the wrapper or just forget about it. Any other code (for example the code in `Quills` itself) does not know or care that the `WeblogEntry` object has been adapted or wrapped and now has something extra. To that code, the `WeblogEntry` is still a normal `WeblogEntry`, even though it now has annotations.

annotations Annotations can be added in several ways, but most used is the method associated with the `IAttributeAnnotatable` interface that I mentioned above. In our case, two links are added in a new hidden attribute of that `WeblogEntry`.

This is implemented by the **quadapter** (Quills Adapter) product, which uses the `keywordannotator` product. In fact, this `quadapter` product is added in the `examples` directory of `keywordannotator`. By the way, both products have tests, so you can just run them and see for yourself that all this actually works.

This section is rather important. If you understand what our goal and our strategy is, you stand a good chance of understanding the next sections which present the actual code. On the other hand, if you prefer a *bottom up* approach, then reading the next sections may help you in understanding this strategy.

Each of the next few sections first present the code from the general `keywordannotator` product and then the code from the specific `quadapter` product. Sometimes I find it hard myself to follow what `keywordannotator` is actually doing, but reading the corresponding `quadapter` code usually helps. :)

4.7 Event handler

4.7.1 zcml registration

Okay, for the general `keywordannotator` product we want an event handler that gets activated whenever an object implementing the `IAttributeAnnotatable` interface gets modified or added. We register that in the `configure.zcml` file like this:

```
<subscriber
  for="zope.app.annotation.interfaces.IAttributeAnnotatable
       zope.app.event.interfaces.IObjectModifiedEvent"
  handler=".events.annotationEventHandler" />
```

That takes care of modified objects. For added objects we replace 'IObjectModifiedEvent' with 'IObjectCreatedEvent'.

In the case of quadapter we want an event handler that gets activated only for object implementing the IWeblogEntry interface:

```
<subscriber
  for="Products.Quills.interfaces.IWeblogEntry
      zope.app.event.interfaces.IObjectModifiedEvent"
  handler="Products.keywordannotator.events.↔
      annotationEventHandler" />
```

4.7.2 Event handler code

In keywordannotator the code is this:

```
def annotationEventHandler(ob, event):
    from zope.component import getUtility
    decider = getUtility(IAnnotationDecider, context=ob)
    if decider.matchesKeywords(ob):
        decider.provideInterfaces(ob)
```

So this event handler looks for a utility. That utility tells us whether the object the event was fired for has a keyword that has been marked as special in our code or not. If this condition has been met, we instruct the utility to make sure that the object now provides a few more interfaces.

Note: quadapter simply uses this event handler from keywordannotator without overwriting it.

Now let's look at what that utility looks like.

4.8 Utility

4.8.1 zcml registration

In keywordannotator it looks like this:

```
<utility
  provides=".interfaces.IAnnotationDecider"
  factory=".events.DefaultAnnotationDecider" />
```

This means that when some code wants to have a utility that provides the IAnnotationDecider interface (the event handler code wants this, see above), such a utility can be created by calling the DefaultAnnotationDecider class in the events.py file in the keywordannotator product.

The quadapter overrides this utility in overrides.zcml:

```
<utility
```

```

provides="Products.keywordannotator.interfaces.IAnnotationDecider" ←
    IAnnotationDecider "
factory="Products.quadapter.events.AudioDecider" />

```

Since this is an override, this means that the only known way to create a utility that provides the `IAnnotationDecider` interface, is now calling the `AudioDecider` class in the `events.py` file in the `quadapter` product.

4.8.2 Utility code

So what does that code look like? In `keywordannotator` it is this:

```

class DefaultAnnotationDecider(object):
    implements(IAnnotationDecider)
    keywords = KEYWORDS
    ifaces = (IKeywordMatch,)
    def matchesKeywords(self, object):
        ...
    def provideInterfaces(self, object):
        for iface in self.ifaces:
            if not iface.providedBy(object):
                alsoProvides(object, iface)

```

The implementation of the function `matchesKeywords` is not interesting here. It simply checks whether the keywords of the object match one of the special keywords. By default, only the literal word `'special'` is considered special.

The function `provideInterfaces` is more interesting. It makes sure that a certain object provides all wanted interfaces. By default this is the `IKeywordMatch` marker interface. In the next section we will register an adapter for objects implementing that interface.

In `quadapter` the utility code is just four lines:

```

class AudioDecider(DefaultAnnotationDecider):
    implements(IAnnotationDecider)
    keywords = KEYWORDS
    ifaces = PROVIDE_INTERFACES

```

It uses a different list of keywords that are special, and different interfaces that need to be provided to objects that match one of the keywords. These values are specified in the `config.py` file:

```

KEYWORDS = ['audio', 'preken']
PROVIDE_INTERFACES = (IMaudio, IAttributeAnnotatable,)

```

Note: `'preken'` is the Dutch word for `'sermons'`.

So, with just a few lines, the `quadapter` product changes the default adapter so it reacts to different keywords and provides different interfaces.

Note that a `Quills WeblogEntry` does not by default provide the `IAttributeAnnotatable` interface, so we must instruct the utility to make sure that `WeblogEntries` with one

of the special keywords now implement that interface as well. If wanted, we could add code to quadapter that makes sure that *all* WeblogEntries also provide the IAttributeAnnotatable interface. If you want that, look at the `utils.py` file of keywordannotator.

At this point you may want to look back at [subsection 4.7.2](#) (Event handler code) to see how this utility is used by the event handler.

4.9 Current Status

We have covered a lot of ground already. So before we continue with adapters and annotations, let's take a break and see what we have accomplished so far.

If you are only using the keywordannotator product, the situation is this:

- An object implementing the IAttributeAnnotatable interface,
- given the special keyword ('special'),
- now also implements the IKeywordMatch interface.

If next to keywordannotator you are also using Quills and the quadapter product, then your situation is this:

- An object implementing the IWeblogEntry interface,
- given one of the special keywords ('audio' or 'preken'),
- now also implements the IMAudio interface
- and the IAttributeAnnotatable interface.

If this is not clear yet, then it is better to reread some of the previous sections. If you *do* understand the current status, then it is time for adapters and annotations.

4.10 Adapter

4.10.1 zcml registration

Remember what we want to do: we want to adapt objects implementing a certain interface by giving them annotations. In keywordannotator we register an adapter for objects implementing the IKeywordMatch interface. This adapter should provide such an object the new IKeywordBasedAnnotations interface, which is just a marker interface (or label). This adapter can be created by calling the KeywordBasedAnnotations class in `events.py`. We register that adapter in the `configure.zcml` file:

```
<adapter
  for=".interfaces.IKeywordMatch"
  provides=".interfaces.IKeywordBasedAnnotations"
  factory=".events.KeywordBasedAnnotations"
/>
```

In quadapter we do something similar:

```
<adapter
  for=".interfaces.IMaudio"
  provides=".interfaces.IAudioAnnotations"
  factory=".events.AudioAnnotations"
/>
```

This means that the adapter can be created by calling the AudioAnnotations class in the events.py file of quadapter. This adapter provides the IAudioAnnotations interface for objects that already implement the IMaudio interface.

IAudioAnnotations is not a marker interface but a 'normal' interface, if you want to make that distinction. In the interfaces.py file of quadapter we state that any object that claims to implement the IAudioAnnotations interface should have the attributes completeURL and partURL:

```
class IAudioAnnotations(Interface):
    """Provide access to the audio annotations of an ↵
       IMaudio object.
    """

    completeURL = schema.URI(title=u'URL to complete audio↵
                             content')
    partURL = schema.URI(title=u'URL to a part of the ↵
                          audio content')
```

At this point it may be good to say that Zope 3 tends to use a lot of interfaces. But you have probably already noticed that. :)

4.10.2 Adapter Code

We have registered two adapters. Now how does the code look? In keywordannotator:

```
class KeywordBasedAnnotations(object):
    ...
    def __init__(self, context):
        self.context = context
        annotations = IAnnotations(self.context)
        self._metadata = annotations.get(self._anno_key, ↵
                                       None)
        if self._metadata is None:
            self._metadata = PersistentDict()
            annotations[self._anno_key] = self._metadata
```

The `__init__` function is the factory; in other words, it is the function that creates a KeywordBasedAnnotations object based on another object that is passed in via the 'context' parameter. This is the spot where we first really see annotations. It is this line:

```
annotations = IAnnotations(self.context)
```

Here the context object is adapted to the IAnnotations interface and the resulting wrapped or adapted object is stored in the annotations variable. That variable is now basically a python dictionary which at this point probably does not contain any values. The other lines make sure that a basic structure for storing annotations is now available in that object.

Now on to the adapter code in quadapter:

```
class AudioAnnotations(KeywordBasedAnnotations):
    ...
    def __get_completeURL(self):
        return self._metadata.get(COMP_ANNO)
    def __set_completeURL(self, url):
        self._metadata[COMP_ANNO] = url
    completeURL = property(__get_completeURL, ←
        __set_completeURL)
```

This class subclasses the KeywordBasedAnnotations class from keywordannotator, so it inherits the init function we saw above. But the lines in this code do the stuff that we actually started this product for: they add the completeURL property, where we can store the link to the complete service of my church. This link is stored in the self._metadata property, which is an annotation to this object.

The same is done (in some code that is not shown here as it is basically the same) for the partURL property, where we can now store a link to the partial service or more sane terms: the sermon.

4.11 Annotations

In the previous section we have actually already seen how annotations are added to an object and that they are basically just a python dictionary. But before this is possible, we need to add one line to the configure.zcml of keywordannotator:

```
<include package="zope.app.annotation" />
```

This directive loads another zcml directive which is in the zope software directory, in the file *lib/python/zope/app/annotation/configure.zcml*:

```
<adapter
    for=".interfaces.IAttributeAnnotatable"
    provides=".interfaces.IAnnotations"
    factory=".attribute.AttributeAnnotations"
/>
```

This actually concludes our strategy. All the code and configuration is now in place to add those annotations to a WeblogEntry by adapting it if it implements or provides the IMAudio interface, which is added to it by a utility if it decides that the condition is

met after an event takes places that is handled by an event handler. If you understood what I just wrote, then you are very smart. :)

4.12 Adding Annotations

At this point, if you look at the state of things on the Plone level, we can add a normal `WeblogEntry` and give it a keyword 'audio'. Then our code makes sure that it provides the `IMaudio` interface. And then it stops! There is not yet a way to actually put something in the annotations. But we can arrange that. First we install a new product: `CMFonFive`⁷.

That product may get assimilated into the core of CMF itself some day (Plone is based on CMF, the Content Management Framework). But for now this extra product is needed for the following lines that we add in the `configure.zcml` of `quadapter`:

```
<browser:menu
    id="object_tabs"
    title="Object tabs" />

<browser:menuItem
    for="Products.quadapter.interfaces.IMaudio"
    menu="object_tabs"
    title="Audio urls"
    action="maudio_edit"
    description="Edit form for audio urls"
    permission="zope2.ManageProperties"
/>
```

This adds a menu item in the Plone site to the object tabs of any object that implements the `IMaudio` interface. When you add this code, restart your Zope instance, and look at a `WeblogEntry` that has one of the special words and thus provides the `IMaudio` interface, you will see a tab that links to the action `maudio_edit`.

At that point we are almost done. We now need to make an edit form for that tab, that calls the code that sets the Annotations for this object. If you know how to make an edit form in html, then you should be able to make this yourself, possibly with the use of the `Archetypes` product.

4.13 Viewing Annotations

When you have added those links in the annotation of a `WeblogEntry`, you also want to be able to see them when you actually look at a `WeblogEntry` in your browser. We can use one more Zope 3 technique for this: the `BrowserView`. Essentially this also is an adapter. We need to define a `BrowserView` in `quadapter`:

⁷<http://codespeak.net/z3/cmfonfive/>

```
<browser:page
  name="audio_entry_view"
  for="Products.quadapter.interfaces.IMaudio"
  permission="zope2.View"
  allowed_interface="Products.quadapter.interfaces.I←
    IAudioWeblogView"
  class=".browser.AudioWeblogView"
/>
```

This basically means that for an object providing the IMaudio interface we have a python class that gives us some functions to call in an html page template. In fact, we can now copy the file `weblogentry_view.pt` from Quills and add these lines at the right spot:

```
<tal:extra tal:define="view context/@@audio_entry_view|←
  nothing;"
  tal:condition="view">
  <metal:block metal:use-macro="here/maudio_macros/macros/←
    extratext">
    Display extra text provided by view of IMaudio.
  </metal:block>
</tal:extra>
```

This uses a `maudio_macros.pt` file that gets the links from the object. Now when you view a normal WeblogEntry you just see the normal page that you would otherwise see. But when you view an entry with the IMaudio interface, for which you have added links with the edit form in the previous section, you will now see some extra text containing those links. The specifics are left as an exercise to the reader as they are just standard page template techniques, which should be familiar and which are not too interesting for this tutorial. If you do not get it working, contact me.

4.14 Conclusions

Using Zope 3 like this is:

- clean:
 - Only one page template from Quills is changed and no code is overwritten.
 - If we added a metal:macro with a fill-slot to Quills, even that single overwrite would not be necessary.
- compatible:
 - We only *add* features, we do not change existing behaviour of WeblogEntries.

- Our audio enhanced WeblogEntries are still first class WeblogEntries. Existing code that expects a normal WeblogEntry can handle our audio entries just fine.
- Our changes work on *any* content type that implements the IWeblogEntry interface. So if someone really does create a VideoWeblogEntry implementing IWeblogEntry, our code would work for such an object as well.

In other words: Zope 3 is the place to be!

Chapter 5

Conclusion

I learned a lot at Zest Software the last year. I now have more experience with subversion, LDAP and Zope 3. Of course I also did a lot with Plone itself, as that is the main focus of Zest Software. We have enough clients to keep us busy and we are constantly on the lookout for new developers.

The chapter on subversion was hopefully interesting. But the steps described there have one downside: they depend on other people to actually do the import of a subversion dump into the Plone collective. And the people that can do that are pretty much swamped in other work, so it can take a while before a product is actually imported into the collective using this technique.

The ldapconfig product currently is subject of a debate between a few major developers. The idea is now to extend this product with some code that makes it unnecessary to manually patch the LDAPMultiPlugins product. If that succeeds, then connecting Plone with LDAP will have become far easier.

The keywordannotator and accompanying quadapter product will be released on plone.org for the general public, so everyone can benefit. More important perhaps is that the chapter describing those products will be added as a tutorial on plone.org, so other developers can use that as an example.

And most important of all, as a conclusion about my year at Zest Software: it was fun!

Appendix A

Subversion import script

```
#!/bin/sh
# Make a dump file for a project that can't be done by a ↵
  single
# svndumpfilter command as stuff has been copied from ↵
  other projects.

# How to call several svn commands
SVNADMIN="sudo svnadmin"
SVNADMIN_LOAD="$SVNADMIN load --quiet"
SVNDUMPFILTER="svndumpfilter --drop-empty-revs --renumber-↵
  revs --quiet"
SVNEXPORT="svn export --ignore-externals"
SVNCHECKOUT="svn checkout --ignore-externals"

# Project we want to dump
PROJECT="eXtremeManagement"

# Main repository
ORI_REPOS="/var/lib/svn/zest"
# Temporary repository used for creating a good dump file
TMP_REPOS="/var/lib/svn/temp"
# Final repository where we check that it actually works
FIN_REPOS="/var/lib/svn/fin"

# Dump file for the complete repository
# (numbers may be added after this)
DUMPFILE_ALL="zest.dump"

# Dump file for the project we want:
# (numbers may be added after this)
```

```
DUMPFILe_PROJECT="xm.dump"
```

```
#####↵
```

```
# Give the user some fair warnings.
```

```
cat <<EOF
```

```
WARNING!!!
```

```
This will remove the following subversion repositories if ↵  
they exist:
```

```
$TMP_REPOS
```

```
$FIN_REPOS
```

```
and in this directory it removes the files and directories↵  
:
```

```
$DUMPFILe_ALL*
```

```
$DUMPFILe_PROJECT*
```

```
$PROJECT*
```

```
diff*
```

```
Are you 100 percent sure you want this?
```

```
Enter y to continue. Enter n (or any other key) to stop.
```

```
EOF
```

```
read ANSWER
```

```
if test x$ANSWER != 'xy'; then
```

```
    exit 0
```

```
fi
```

```
# Cleanup previous attempts
```

```
sudo rm -rf $TMP_REPOS $FIN_REPOS
```

```
rm -rf $DUMPFILe_ALL* $DUMPFILe_PROJECT* $PROJECT* diff*
```

```
echo "Creating temporary repository at $TMP_REPOS"
```

```
$SVNADMIN create $TMP_REPOS
```

```
sudo chmod -R a+w $TMP_REPOS
```

```
# Dump revisions 0 to 2844 as these give no trouble.
```

```
GOOD_REV=2844
```

```
TROUBLE_REV=$((GOOD_REV + 1))
```

```
NEXT_REV=$((TROUBLE_REV + 1))
```

```
echo "Dumping revisions 0 to $GOOD_REV"
$SVNADMIN dump --quiet --revision 0:$GOOD_REV $ORI_REPOS >↵
  ${DUMPFILe_ALL}.1

echo "Filtering our Project into ${DUMPFILe_PROJECT}.1 ↵
  ..."
cat ${DUMPFILe_ALL}.1 | $SVNDUMPFILeTER include $PROJECT > ↵
  ${DUMPFILe_PROJECT}.1

echo "Load this dump file into the temporary repository↵
  ..."
$SVNADMIN_LOAD $TMP_REPOS < ${DUMPFILe_PROJECT}.1

echo "Checking out till revision $GOOD_REV"
$SVNCHECKOUT --quiet file://$TMP_REPOS/$PROJECT

echo "Exporting the troubled revision from the original ↵
  repository."
$SVNEXPORT --force --quiet -r $TROUBLE_REV file://↵
  $ORI_REPOS/$PROJECT

cd $PROJECT
# Now do something by hand, depending on where exactly ↵
  your problem is
svn add integration
svn commit --quiet --message "Added $PROJECT/integration."
cd ..

echo "Dumping revisions $NEXT_REV to HEAD"
$SVNADMIN dump -q -r $NEXT_REV:HEAD --incremental ↵
  $ORI_REPOS > ${DUMPFILe_ALL}.2

echo "Filtering our Project into ${DUMPFILe_PROJECT}.2 ↵
  ..."
cat ${DUMPFILe_ALL}.2 | $SVNDUMPFILeTER include $PROJECT > ↵
  ${DUMPFILe_PROJECT}.2

echo "Load this dump file into the temporary repository↵
  ..."
$SVNADMIN_LOAD $TMP_REPOS < ${DUMPFILe_PROJECT}.2

echo "Now we must check if everything works."
```

```

echo "Dumping all revisions from $TMP_REPOS"
$SVNADMIN dump --quiet $TMP_REPOS > ${DUMPFILe_PROJECT}.↵
    all

# For eXtremeManagement we still want to exclude two paths↵
:
# eXtremeManagement/integration and
# eXtremeManagement/temp
echo "Filtering our Project into ${DUMPFILe_PROJECT}.2 ↵
    ..."
cat ${DUMPFILe_PROJECT}.all | $SVNDUMPFILeTER exclude ↵
    $PROJECT/integration $PROJECT/temp > ${DUMPFILe_PROJECT}↵
    }.noint

echo "Creating temporary project repository at $FIN_REPOS"
$SVNADMIN create $FIN_REPOS
sudo chmod -R a+w $FIN_REPOS

echo "Load this dump file into the project repository..."
$SVNADMIN_LOAD $FIN_REPOS < ${DUMPFILe_PROJECT}.noint

echo "Now an export of our project from all three"
echo "repositories should be the same.  Exporting..."

$SVNEXPORT --quiet file://$ORI_REPOS/$PROJECT ${PROJECT}.↵
    ORI
$SVNEXPORT --quiet file://$TMP_REPOS/$PROJECT ${PROJECT}.↵
    TMP
$SVNEXPORT --quiet file://$FIN_REPOS/$PROJECT ${PROJECT}.↵
    FIN

echo "Done exporting.  Starting comparison now..."

echo ↵
    "#####"↵

echo "##### BEGIN of differences between original and ↵
    final. #####"
diff -q -r ${PROJECT}.ORI ${PROJECT}.FIN
echo "##### END    of differences between original and ↵
    final. #####"
echo ↵
    "#####"↵

```

```
echo "Listing details, if any..."
diff -r ${PROJECT}.ORI ${PROJECT}.TMP > diff-ori-tmp
diff -r ${PROJECT}.TMP ${PROJECT}.FIN > diff-tmp-fin
diff -r ${PROJECT}.ORI ${PROJECT}.FIN > diff-ori-fin
echo "Details are in the diff* files."
```

Appendix B

Makefile

```
### Default target

default: main.pdf

ready: front.pdf default maurits-year-at-zest.pdf

### Definitions

STYLESHEET = preamble.tex

RST2LATEX_OPTIONS = \
    --documentclass=report \
    --documentoptions=12pt,a4paper \
    --use-latex-toc \
    --use-latex-docinfo \
    --toc-entry-backlinks \
    --stylesheet=$(STYLESHEET) \
    --use-verbatim-when-possible \
    --strict

REST_FILES = main.txt preface.txt abstract.txt intro.txt ↵
    common.txt \
    conclusion.txt

INCLUDE_FILES = /home/maurits/svn/mvrsite/keywordannotator↵
    /doc/tutorial.txt \
    /home/maurits/svn/debianize/svnhowto.rst \
    /home/maurits/svn/ldapconfig/README.txt \
    /home/maurits/svn/ldapconfig/docs/PLUGINS.txt \
    /home/maurits/svn/ldapconfig/docs/DEVELOPERS.txt \
    front.pdf
```

```
HELPER_FILES = Makefile $(STYLESHEET) fixup.py

### Main targets

# For printing:
main.tex: $(REST_FILES) $(INCLUDE_FILES) $(HELPER_FILES)
    @rst2latex $(RST2LATEX_OPTIONS) --hyperlink-color=↔
    black main.txt temp.pre
    @python fixup.py temp.pre > main.tex

# For the web:
maurits-year-at-zest.tex: $(REST_FILES) $(INCLUDE_FILES) $↔
    (HELPER_FILES)
    @rst2latex $(RST2LATEX_OPTIONS) main.txt temp.pre
    @python fixup.py temp.pre > maurits-year-at-zest.↔
    tex

%.pdf: %.tex
    pdflatex $<
    pdflatex $<

front.pdf: images/zestlogo.jpg images/mauritsdrinkt.jpg ↔
    preamble.tex

### House keeping targets

clean:
    @rm -rf *~ *log *aux auto *.out *.toc *.pre

clobber: clean
    @rm -rf main.tex maurits-year-at-zest.tex

dist-clean: clobber
    @rm -rf main.pdf maurits-year-at-zest.pdf
```

Appendix C

Main file for this report

```
=====  
My Year at Zest  
=====
```

```
.. include:: common.txt
```

```
.. raw:: latex
```

```
    \setcounter{tocdepth}{1}  
    \bibliographystyle{alpha}  
    \pagenumbering{roman}
```

```
.. include:: preface.txt
```

```
.. raw:: latex
```

```
    \tableofcontents  
    \newpage  
    \begin{abstract}
```

```
.. include:: abstract.txt
```

```
.. raw:: latex
```

```
    \end{abstract}  
    \pagenumbering{arabic}
```

```
.. include:: intro.txt
```

```
.. Chapter on svn howto:
```

```

.. include:: /home/maurits/svn/debianize/svnhowto.rst

.. raw:: latex

    (For the script, see \autoref{subversion-script})

.. Chapter on ldapconfig product:

.. include:: /home/maurits/svn/ldapconfig/README.txt

.. Chapter on keywordannotator product:

.. include:: /home/maurits/svn/mvrsite/keywordannotator/↵
    doc/tutorial.txt

.. include:: conclusion.txt

.. raw:: latex

\appendix
\chapter{Subversion import script}
\label{subversion-script}
\lstinputlisting[showstringspaces=false, keywordstyle=↵
    bash]{/home/maurits/svn/debianize/svnxm.sh}

\chapter{Makefile}
\label{chap-makefile}
\lstinputlisting[showstringspaces=false, keywordstyle=↵
    make]{Makefile}

\chapter{Main file for this report}
\label{chap-main-file}
\lstinputlisting[showstringspaces=false, keywordstyle=↵
    rst]{main.txt}

\chapter{Helper script for reStructuredText and \LaTeX\@↵
    }
\label{chap-fixup}
\lstinputlisting[showstringspaces=false, keywordstyle=↵
    python]{fixup.py}

```

Appendix D

Helper script for reStructuredText and L^AT_EX

```
# Based on code from Reinout van Rees.
import codecs
import re
import sys

filename = sys.argv[1]
file = codecs.open(filename, 'r', 'utf-8')

# regex patterns
verbatimStartPattern = re.compile(r"begin{quote}\\begin{↵
    verbatim}")
verbatimEndPattern = re.compile(r"end{verbatim}\\end{quote}↵
    ")
prefacePattern = re.compile(r"chapter{Preface}")
spacelyPattern = re.compile(r"href{http://spacely.space/↵
    svn/pockets}{http://spacely.space/svn/pockets}")

urlNotePattern = re.compile(r"\\href{http(.*)}{(.*)}")
internalLinkPattern = re.compile(r"\\href{\\#(.*)}{(.*)}↵
    ")
hypertargetPattern = re.compile(r"^\\hypertarget{(.*)}{}$↵
    ")
sectionPattern = re.compile(r"^\\(chapter|section|↵
    subsection)")
weblogEntryPattern = re.compile(r"WeblogEntr") # might be ↵
    plural
titlePattern = re.compile(r"^\\maketitle")
```

```

settings = {'inVerbatim': False,
            'inHyperTarget': False,}

titlesnippet = r"""
\pagestyle{empty}
\\begin{figure}
\includegraphics{front.pdf}
\end{figure}
\clearpage
\pagestyle{headings}
\maketitle
"""

def fixup(line, settings):
    if titlePattern.search(line):
        line = titlePattern.sub(titlesnippet, line, 1)
        return line
    if weblogEntryPattern.search(line) and (settings['inVerbatim'] == False):
        line = weblogEntryPattern.sub('Weblog\-Entr', line, 1)
    if spacelyPattern.search(line):
        line = spacelyPattern.sub('url{http://spacely.←
space/svn/pockets}', line, 1)
    if urlNotePattern.search(line):
        line = urlNotePattern.sub(r'\2\\footnote{\url{http←
\1}}', line, 1)
        line = fixup(line, settings)
    if prefacePattern.search(line):
        # No chapter number on the Preface, please
        line = prefacePattern.sub('chapter*{Preface}', ←
line, 1)
    if verbatimStartPattern.search(line):
        assert(settings['inVerbatim'] == False)
        line = verbatimStartPattern.sub('begin{lstlisting←
}', line, 1)
        settings['inVerbatim'] = True
    elif verbatimEndPattern.search(line):
        assert(settings['inVerbatim'] == True)
        line = verbatimEndPattern.sub('end{lstlisting}', ←
line, 1)
        settings['inVerbatim'] = False
    if internalLinkPattern.search(line):
        line = internalLinkPattern.sub(r'\\autoref{\1} ←

```

```

        (\2)', line, 1)
if hypertargetPattern.search(line):
    # Turn this:
    #   \hypertarget{preface}{}
    #   \chapter*{Preface}
    # into this:
    #   \chapter*{Preface}
    #   \hypertarget{preface}{}
    #   \label{preface}{}

    assert(settings['inHyperTarget'] == False)
    settings['line1'] = line
    settings['line2'] = hypertargetPattern.sub(r'\\↔
        label{\1}', line, 1)
    settings['inHyperTarget'] = True
    line = ''
if sectionPattern.search(line) and settings['↔
inHyperTarget'] == True:
    line = line + settings['line1'] + settings['line2↔
        ']
    settings['inHyperTarget'] = False
    del settings['line1']
    del settings['line2']
return line

# The main work
for line in file:
    line = fixup(line, settings)
    print line,

file.close()

```