## Contents

1 What's new in Zope 4.0  
1.1 Version numbering increase  
1.2 Memory use  

2 Installing Zope with *zc.buildout*  
2.1 About *zc.buildout*  
2.2 Prerequisites  
2.3 Installing standalone Zope using *zc.buildout*  
2.4 Creating a buildout-based Zope instance  
2.5 Building the documentation with *Sphinx*  

3 Installing Zope with *virtualenv*  
3.1 Create a Virtual Environment  
3.2 Install the Zope2 2.13.22 Software Packages  
3.3 Creating a Zope instance  
3.4 Using the *virtualenv* as the Zope Instance  

4 Configuring and Running Zope  
4.1 Configuring Zope  
4.2 Running Zope in the Foreground  
4.3 Running Zope as a Daemon  
4.4 Integrating with System Startup  
4.5 Logging In To Zope  
4.6 Troubleshooting  
4.7 Adding extra commands to Zope  

5 Running Zope2 as a WSGI Application  
5.1 Install the Supporting Software  
5.2 Update the Zope Application Configuration  
5.3 Create the WSGI Server Configuration  
5.4 Set up the Admin User  
5.5 Start the WSGI Server  
5.6 Running Other Applications in the same WSGI Server Process  

6 Special Users  
6.1 Adding Managers  
6.2 The Initial User  
6.3 The Emergency User  
6.4 Format of ‘inituser’ and ‘access’
What’s new in Zope 4.0

The article explains the new high-level features and changes found in this version of Zope.
You can have a look at the detailed change log to learn about all minor new features and bugs being solved in this release.
Note: This is currently work-in-progress!

1.1 Version numbering increase

Version numbers for Zope have been confusing in the past. The original Zope project iterated through version one to two up to version 2.13. In parallel a separate project was launched using the name Zope 3. Zope 3 wasn’t a new version of the original Zope project and in hindsight should have used a different project name. These days this effort is known as BlueBream.

In order to avoid confusion between the separate Zope 3 project and a new version of this project, it was decided to skip ahead and use Zope 4.0 as the next version number. The increase in the major part of the version also indicates the clear intention to allow backwards incompatible changes.

1.2 Memory use

Zope 4 depends on the new DateTime version 3. DateTime 3 has been optimized for better memory use. Applications using a lot of DateTime values like the Plone CMS have seen total memory usage to decrease by 10% to 20% for medium to large deployments.
Installing Zope with zc.buildout

This document describes how to get going with Zope using zc.buildout.

2.1 About zc.buildout

zc.buildout is a powerful tool for creating repeatable builds of a given software configuration and environment. The Zope developers use zc.buildout to develop Zope itself, as well as the underlying packages it uses.

2.2 Prerequisites

In order to use Zope, you must have the following pre-requisites available:

• A supported version of Python, including the development support if installed from system-level packages. Supported versions include:
  – 2.7.x
• Zope needs the Python zlib module to be importable. If you are building your own Python from source, please be sure that you have the headers installed which correspond to your system’s zlib.
• A C compiler capable of building extension modules for your Python (gcc recommended). This is not necessary for Windows as binary releases of the parts that would need compiling are always made available.
• If you wish to install Zope as a Service on Windows, you will need to have the pywin32 package installed.

2.3 Installing standalone Zope using zc.buildout

In this configuration, we use zc.buildout to install the Zope software, but then generate server “instances” outside the buildout environment.

2.3.1 Installing the Zope software

Installing the Zope software using zc.buildout involves the following steps:

• Download the Zope 2 source distribution from PyPI
• Bootstrap the buildout
2.3.2 Creating a Zope instance

Once you’ve installed Zope, you will need to create an “instance home”. This is a directory that contains configuration and data for a Zope server process. The instance home is created using the `mkzopeinstance` script:

```
$ bin/mkzopeinstance
```

You can specify the Python interpreter to use for the instance explicitly:

```
$ bin/mkzopeinstance --python=$PWD/bin/zopepy
```

You will be asked to provide a user name and password for an administrator’s account during `mkzopeinstance`. To see the available command-line options, run the script with the `--help` option:

```
$ bin/mkzopeinstance --help
```

Note: The traditional “inplace” build is no longer supported. If using `mkzopeinstance`, always do so outside the buildout environment.

2.4 Creating a buildout-based Zope instance

Rather than installing Zope separately from your instance, you may wish to use `zc.buildout` to create a self-contained environment, containing both the Zope software and the configuration and data for your server. This procedure involves the following steps:

- Create the home directory for the buildout, including `etc`, `log` and `var` subdirectories.
- Fetch the buildout bootstrap script into the environment.
- Fetch the version file into the environment, for example: https://raw.githubusercontent.com/zopefoundation/Zope/master/versions.cfg
- Create a buildout configuration as follows:
buildout.cfg

```ini
[buildout]
parts = instance
extends = versions.cfg

[instance]
recipe = zc.recipe.egg
eggs = Zope2
interpreter = py
scripts = runzope zopectl
initialization =
   import sys
   sys.argv[1:1] = ['-C',r'${buildout:directory}/etc/zope.conf']
```

This is the minimum but all the usual buildout techniques can be used.

- Bootstrap the buildout
- Run the buildout
- Create a Zope configuration file. A minimal version would be:

etc/zope.cfg

```ini
%define INSTANCE <path to your instance directory>
python $INSTANCE/bin/py[.exe on Windows]
instancehome $INSTANCE
```

A fully-annotated sample can be found in the Zope2 egg:

```bash
$ cat eggs/Zope2--*/Zope2/utilities/skel/etc/zope.conf.in
```

<rest of the stuff that goes into a zope.conf, e.g. databases and log files.>

An example session:

```bash
$ mkdir /path/to/instance
$ cd /path/to/instance
$ mkdir etc logs var
$ wget http://downloads.buildout.org/2/bootstrap.py
$ vi buildout.cfg
$ /path/to/your/python bootstrap.py
$ bin/buildout
$ cat eggs/Zope2--*/Zope2/utilities/skel/etc/zope.conf.in > etc/zope.conf
$ vi etc/zope.conf  # replace <<INSTANCE_HOME>> with buildout directory
$ bin/zopectl start
```

In the `bin` subdirectory of your instance directory, you will find `runzope` and `zopectl` scripts that can be used as normal.

You can use `zopectl` interactively as a command shell by just calling it without any arguments. Try `help` there and `help <command>` to find out about additionally commands of zopectl. These commands also work at the command line.

2.4. Creating a buildout-based Zope instance
Note that there are recipes such as `plone.recipe.zope2instance` which can be used to automate this whole process.

After installation, refer to `Configuring and Running Zope` for documentation on configuring and running Zope.

2.5 Building the documentation with Sphinx

To build the HTML documentation, run the `make-docs` script (installed by the buildout):

```bash
$ bin/make-docs
```
CHAPTER 3

Installing Zope with virtualenv

This document describes how to install Zope into a virtualenv.

3.1 Create a Virtual Environment

```bash
$ /opt/Python-2.7.9/bin/virtualenv z213
New python executable in z213/bin/python
Installing setuptools, pip, wheel...done.
$ cd z213
```

3.2 Install the Zope2 2.13.22 Software Packages

```bash
$ bin/pip install --trusted-host download.zope.org --index http://download.zope.org/Zope2/index/2.13.22/ Zope2
Collecting Zope2
... Successfully installed ...
```

3.3 Creating a Zope instance

Once you’ve installed Zope, you will need to create an “instance home”. This is a directory that contains configuration and data for a Zope server process. The instance home is created using the mkzopeinstance script:

```bash
$ bin/mkzopeinstance
```

You can specify the Python interpreter to use for the instance explicitly:

```bash
$ bin/mkzopeinstance --python=bin/python
```

You will be asked to provide a user name and password for an administrator’s account during mkzopeinstance. To see the available command-line options, run the script with the --help option:

```bash
$ bin/mkzopeinstance --help
```
3.4 Using the `virtualenv` as the Zope Instance

You can choose to use the `virtualenv` as your Zope instance:

```
$ bin/mkzopeinstance -d .
```

In this case, the instance files will be located in the subdirectories of the `virtualenv`:

- `etc/` will hold the configuration files.
- `log/` will hold the log files.
- `var/` will hold the database files.
Whichever method you used to install Zope and create a server instance (see Installing Zope with zc.buildout and Installing Zope with virtualenv), the end result is configured and operated the same way.

4.1 Configuring Zope

Your instance’s configuration is defined in its etc/zope.conf file. Unless you created the file manually, that file should contain fully-annotated examples of each directive.

You can also pass an explicit configuration file on the command line:

```
$ /path/to/zope/instance/bin/zopectl -c /tmp/other.conf show
...
Config file: /tmp/other.conf
```

When starting Zope, if you see errors indicating that an address is in use, then you may have to change the ports Zope uses for HTTP or FTP. The default HTTP and FTP ports used by Zope are 8080 and 8021 respectively. You can change the ports used by editing ./etc/zope.conf appropriately.

The section in the configuration file looks like this:

```
<http-server>
  # valid keys are "address" and "force-connection-close"
  address 8080
  # force-connection-close on
</http-server>
```

The address can just be a port number as shown, or a host:port pair to bind only to a specific interface.

After making any changes to the configuration file, you need to restart any running Zope server for the affected instance before changes are in effect.

4.2 Running Zope in the Foreground

To run Zope without detaching from the console, use the fg command (short for foreground):

```
$ /path/to/zope/instance/bin/zopectl fg
```

In this mode, Zope emits its log messages to the console, and does not detach from the terminal. This also automatically enables debug-mode. Do not use this for production servers.
4.3 Running Zope as a Daemon

Once an instance home has been created, the Zope server can now be started using this command:

```
$ /path/to/zope/instance/bin/zopectl start
```

During startup, Zope emits log messages into /path/to/zope/instance/log/event.log. You can examine it with the usual tools (cat, more, tail, etc) and see if there are any errors preventing Zope from starting.

**Note:** For this to work on Windows, the Zope instance must be installed as a Service. This is done with:

```
bin\zopectl install
```

If you later want to remove this Service, do the following:

```
bin\zopectl remove
```

For the full list of options available for setting up Zope as a Windows Service, do:

```
bin\zopectl install --help
```

4.4 Integrating with System Startup

zopectl can be linked as re-script in the usual start directories on Linux or other System V Unix variants.

You can use zopectl interactively as a command shell by just calling it without any arguments. Try `help` there and `help <command>` to find out about additionally commands of zopectl. These commands also work at the command line.

**Note:** On Windows, a Service can be installed and set to start automatically with the following:

```
bin\zopectl install --startup=auto
```

4.5 Logging In To Zope

Once you’ve started Zope, you can then connect to the Zope webserver by directing your browser to:

```
http://yourhost:8080/manage
```

where ‘yourhost’ is the DNS name or IP address of the machine running Zope. If you changed the HTTP port as described, use the port you configured.

You will be prompted for a user name and password. Use the user name and password you provided in response to the prompts issued during the “make instance” process.

Now you’re off and running! You should be looking at the Zope management screen which is divided into two frames. On the left you can navigate between Zope objects and on the right you can edit them by selecting different management functions with the tabs at the top of the frame.

If you haven’t used Zope before, you should head to the Zope web site and read some documentation. The Zope Documentation section is a good place to start. You can access it at http://docs.zope.org/
4.6 Troubleshooting

- This version of Zope requires Python 2.6.4 or better. It will not run with Python 3.x.
- The Python you run Zope with must have threads compiled in, which is the case for a vanilla build. Warning: Zope will not run with a Python version that uses libpth. You must use libpthread.
- To build Python extensions you need to have Python configuration information available. If your Python comes from an RPM you may need the python-devel (or python-dev) package installed too. If you built Python from source all the configuration information should already be available.
- See the Changelog for important notes on this version of Zope.

4.7 Adding extra commands to Zope

It is possible to add extra commands to zopectl by defining entry points in setup.py. Commands have to be put in the zopectl.command group:

```python
setup(name="MyPackage",
    ....
    entry_points="",
    [zopectl.command]
    init_app = mypackage.commands:init_application
"
)
```

**Note:** Due to an implementation detail of zopectl you can not use a minus character (-) in the command name.

This adds a init_app command that can be used directly from the command line:

```
bin\zopectl init_app
```

The command must be implemented as a Python callable. It will be called with two parameters: the Zope2 application and a list with all command line arguments. Here is a basic example:

```python
def init_application(app, args):
    print 'Initializing the application'
```

Make sure the callable can be imported without side-effects, such as setting up the database connection used by Zope 2.
This document assumes you have installed Zope into a virtualenv (see Installing Zope with virtualenv).

## 5.1 Install the Supporting Software

To run as a WSGI application, you need to install some additional software.

```
$ bin/pip install \
   --trusted-host download.zope.org \
   --index http://download.zope.org/Zope2/index/2.13.22/ \
   repoze.who repoze.tm2 repoze.retry Paste PasteDeploy PasteScript
Collecting repoze.who
```

...  

```
Successfully installed Paste-1.7.5.1 PasteDeploy-1.3.4 PasteScript-1.7.5 repoze.retry-1.2 repoze.tm2-1.0
```

## 5.2 Update the Zope Application Configuration

The generated etc/zope.conf file assumes that Zope will be running using the built-in ZServer.

```
$ vim etc/zope.conf
```

Update the contents as follows.

```
%define INSTANCE /path/to/virtualenv
instancehome $INSTANCE
```

**Note:** The `%define instance /path/to/virtualenv` element must point to the environment: there is no “relative to this file” support built in.

Set up logging for the application.

```
<eventlog>
  level info
  <logfile>
    path $INSTANCE/log/event.log
    level info
  </logfile>
</eventlog>
```
Configure the database (note that you could use ZEO or Relstorage rather than a bare FileStorage):

```xml
<zodb_db main>
    # Main FileStorage database
    <filestorage>
        # See .../ZODB/component.xml for directives (sectiontype
        # "filestorage").
        path $INSTANCE/var/Data.fs
    </filestorage>
    mount-point /
</zodb_db>

<zodb_db temporary>
    # Temporary storage database (for sessions)
    <temporarystorage>
        name temporary storage for sessioning
    </temporarystorage>
    mount-point /temp_folder
    container-class Products.TemporaryFolder.TemporaryContainer
</zodb_db>
```

Because we will be running a separately-configured WSGI server, remove any `<http-server>` configuration from the file.

### 5.3 Create the WSGI Server Configuration

```
$ vim etc/zope.wsgi
```

First, configure the “application” endpoint for Zope:

```
[app:zope]
use = egg:Zope2#main
zope_conf = %(here)s/zope.conf
```

Next, set up the WSGI middleware pipeline:

```
[pipeline:main]
pipeline =
    egg:paste#evalerror
    egg:repoze.retry#retry
    egg:repoze.tm2#tm
    zope
```

The middleware layers are “wrapped” around the application endpoint as follows:

- `paste#evalerror` is debugging middleware, which shows tracebacks for errors raised from the application. It should not be configured for production use.
- `repoze.retry#retry` is middleware which retries requests when retriable exceptions are raised. By default, it retries 3 times, and only for requests which raise `ZODB.ConflictError`. See
http://repozeretry.rtfd.org/ for details on configuring it otherwise.

- `repoze.tm2#tm` is middleware which begins a new transaction for each request, and then either aborts the transaction (if the request raises an exception) or commits it (if not). See http://repozetm2.rtfd.org/ for details on configuring it.

Finally, configure the WSGI server:

```ini
[server:main]
use = egg:paste#http
host = localhost
port = 8080
```

**Note:** Any server conforming to PEP 333/3333 should work, although the parameters could change.

## 5.4 Set up the Admin User

Before starting the WSGI server, run the `addzope2user` script to configure the administrative user.

```bash
$ bin/addzope2user admin <yourpasswordhere>
No handlers could be found for logger "ZODB.FileStorage"
User admin created.
```

## 5.5 Start the WSGI Server

```bash
$ bin/paster serve etc/zope.wsgi
Starting server in PID 24934.
serving on http://127.0.0.1:8080
```

## 5.6 Running Other Applications in the same WSGI Server Process

You can use any of the normal Paste WSGI features to combine Zope and other WSGI applications inside the same server process. E.g., the following configuration uses the composite application support offered by PasteDeploy to host Zope at the `/` prefix, with static files served from disk at `/static`:

```ini
[app:zope-app]
use = egg:Zope2#main
zope_conf = %(here)s/zope.conf

[pipeline:zope-pipeline]
pipeline =
  egg:paste#evalerror
  egg:repoze.retry#retry
  egg:repoze.tm2#tm
  zope-app

[app:static]
use = egg:Paste#static
document_root = %(here)s/static

[composite:main]
```
use = egg:Paste#urlmap
/ = zope-pipeline
/static = static
Because Zope is managed through the web, user names and passwords must be used to assure that only authorized people can make changes to a Zope installation.

### 6.1 Adding Managers

If you need to add a Manager to an existing Zope instance, you can do this using `zopectl` as follows:

```
zoogle adduser 'name' 'password'
```

### 6.2 The Initial User

An initial username and password is needed to “bootstrap” the creation of normal managers of your Zope site. This is accomplished through the use of the ‘inituser’ file in the directory specified as the instance home.

The first time Zope starts, it will detect that no users have been defined in the root user folder. It will search for the ‘inituser’ file and, if it exists, will add the user defined in the file to the root user folder.

Normally, ‘inituser’ is created by the Zope install scripts. Either the installer prompts for the password or a randomly generated password is created and displayed at the end of the build script.

You can use the ‘zpasswd.py’ script to create ‘inituser’ yourself. Execute ‘zpasswd.py’ like this:

```
python zpasswd.py inituser
```

The script will prompt you for the name, password, and allowed domains. The default is to encode the password with SHA, so please remember this password as there is no way to recover it (although ‘zpasswd.py’ lets you reset it.)

### 6.3 The Emergency User

In some situations you may need to bypass normal security controls because you have lost your password or because the security settings have been mixed up. Zope provides a facility called an “emergency user” so that you can reset passwords and correct security settings.

The emergency user password must be defined outside the application user interface. It is defined in the ‘access’ file located in the Zope directory. It should be readable only by the user as which your web server runs.

To create the emergency user, use ‘zpasswd.py’ to create the ‘access’ file like this:
python zpasswd.py access

In order to provide a somewhat higher level of security, various encoding schemes are supported which provide access to either SHA-1 encryption or the standard UNIX crypt facility if it has been compiled into Python. Unless you have some special requirements (see below), you should use the SHA-1 facility, which is the default.

### 6.4 Format of ‘inituser’ and ‘access’

A password file should consist of a single line of the form:

```
name:password
```

Note that you may also add an optional third component to the line in the access file to restrict access by domain. For example, the line:

```
mario:nintendoRules:*.mydomain.com
```

in your ‘access’ file will only allow permit emergency user access from *.mydomain.com machines. Attempts to access the system from other domains will fail, even if the correct emergency user name and password are used.

Please note that if you use the ZServer monitor capability, you will need to run with a clear text password.
You need to set permissions on the directory Zope uses to store its data. This will normally be the *var* directory in the instance home. Zope needs to read and write data to this directory. Before running Zope you should ensure that you give adequate permissions to this directory for the userid Zope will run under.

Depending on how you choose to run Zope you will need to give different permissions to the directory. If you use Zope with an existing web server, it will probably run Zope as ‘nobody’. In this case ‘nobody’ needs read and write permissions to the *var* directory.

If you change the way you run Zope, you may need to modify the permissions of the directory and the files in it to allow Zope to read and write under its changed userid.
Zope effective user support

Note: It is best practice to run Zope behind a reverse proxy like Apache, Squid or Varnish. In this case, you do not need to run or install Zope with root privileges, since the reverse proxy will bind to port 80 and proxy back all request to Zope running on an unprivileged port.

Zope can bind its network service to low ports such as 21 (FTP) and 80 (HTTP). In order to bind to low ports, Zope must be started as the root user. However, Zope will only run as root long enough to bind to these low ports. It will then attempt to setuid to a less privileged user.

You must specify the user to which Zope will attempt to setuid by changing the ‘effective-user’ parameter in the zope.conf configuration file to an existing username or UID. All runtime files will be written as this user. If you do not specify an ‘effective-user’ in the configuration file, and you attempt to start Zope, it will refuse to start.

Zope additionally emits a warning if you specify ‘nobody’ as the ‘effective-user’. The rationale for this warning stems from the fact that, historically, many other UNIX services dropped privileges to the ‘nobody’ account after starting as root. Any security defects in these services could cause someone to gain access as the ‘nobody’ account on your system. If someone was to gain control of your ‘nobody’ account they could compromise your Zope files.

The most important thing to remember about effective user support is that you don’t have to start Zope as root unless you want to listen for requests on low ports (ports beneath 1024). In fact, if you don’t have this need, you are much better off just starting Zope under a dedicated user account.
Signals (POSIX only)

Signals are a POSIX inter-process communications mechanism. If you are using Windows then this documentation does not apply.

Zope responds to signals which are sent to the process id specified in the file ‘$INSTANCE_HOME/var/Z2.pid’:

**SIGHUP** - close open database connections, then restart the server process. A idiom for restarting a Zope server is:

```
kill -HUP `cat $INSTANCE_HOME/var/z2.pid`
```

**SIGTERM** - close open database connections then shut down. A common idiom for shutting down Zope is:

```
kill -TERM `cat $INSTANCE_HOME/var/z2.pid`
```

**SIGINT** - same as SIGTERM

**SIGUSR1** - dump a stack trace of all threads to stdout. This can help diagnosing ‘stuck’ Zope processes if all threads are stuck.

**SIGUSR2** - close and re-open all Zope log files (z2.log, event log, detailed log.) A common idiom after rotating Zope log files is:

```
kill -USR2 `cat $INSTANCE_HOME/var/z2.pid`
```
A utility known as ‘zopectl’ is installed into generated instance homes.

If you wish to run Zope in debug mode, run zopectl in foreground mode:

```
$ bin/zopectl fg
```

You can also use it to inspect a Zope instance’s running state via an interactive Python interpreter by passing zopectl the ‘debug’ parameter on the command line. The ‘top-level’ Zope object (the root folder) will be bound to the name ‘app’ within the interpreter. You can then use normal Python method calls against app and use the Python interpreter normally to inspect results:

```
$ bin/zopectl debug
Starting debugger (the name "app" is bound to the top-level Zope object)

>>> app.keys()
['acl_users']
>>>```

CHAPTER 11

Maintenance information

Note: This is internal documentation for Zope developers having to create official Zope releases.

11.1 Release process

11.1.1 Maintainers

The following persons have access to the Zope2 package on PyPI (in order to release new versions):

- Hanno Schlichting
- Tres Seaver

11.1.2 Steps for creating a new Zope release

- check the versions.cfg file for outdated or updated packages and update version information where necessary
- update version information:
  - setup.py (remove dev postfix)
  - versions.cfg (pin Zope2)
- update docs/CHANGES.rst
- run all tests:
  ```
  bin/alltests
  ```
- tag the release
- upload the tagged release to PyPI:
  ```
  python2.7 setup.py egg_info -RDb '' sdist --formats=zip register upload
  ```
- update version information:
  - setup.py (bump version number, add dev postfix)
  - versions.cfg (remove Zope2 pin)
- check the visible releases on readthedocs.org at (should default to showing the active branches):
• update the status of all bugs associated with the released milestone:
  https://launchpad.net/zope2/+milestone/2.13.x
This file contains change information for the current Zope release. Change information for previous versions of Zope can be found at http://docs.zope.org/zope2/

12.1 4.0a1 (unreleased)

12.1.1 Bugs Fixed

- Document running Zope as a WSGI application.
- Queue additional warning filters at the beginning of the queue in order to allow overrides.
- Issue #16: prevent leaked connections when broken EndRequestEvent subscribers raise exceptions.
- Ensure that the WSGIPublisher begins and ends an interaction at the request/response barrier. This is required for instance for the checkPermission call to function without an explicit interaction parameter.
- Made sure getConfiguration().default_zpublisher_encoding is set correctly.

12.1.2 Features Added

- Optimized the OFS.Traversable.getPhysicalPath method to avoid excessive amounts of method calls.
- During startup open a connection to every configured database, to ensure all of them can indeed be accessed. This avoids surprises during runtime when traversal to some database mountpoint could fail as the underlying storage cannot be opened at all.
- Explicitly close all databases on shutdown, which ensures Data.fs.index gets written to the file system.
- Always configure a blob-dir in the default skeleton.
- ZPublisher: If IBrowserPage is provided by a view, form input is decoded. This makes it easier to use zope.formlib and z3c.form in Zope 2.
- Remove control panel object from the ZODB.
- Updated to Zope Toolkit 2.0dev.
- Updated distributions:
  - AccessControl = 3.0.11
  - Acquisition = 4.0.3
Zope 2 Documentation, Release 4.0

- BTrees = 4.0.8
- DateTime = 4.0.1
- ExtensionClass = 4.1a1
- docutils = 0.9.1
- manuel = 1.6.0
- Missing = 3.0
- Persistence = 3.0a1
- Products.ExternalMethod = 2.13.1
- Products.MailHost = 2.13.2
- Products.ZCatalog = 3.1
- Record = 3.0
- ZopeUndo = 4.0

12.1.3 Restructuring

- **Products.SiteErrorLog**: Is now a separated package.
- **OFS**: Removed duplicate code in ZopeFind and ZopeFindAndApply
- **Five**: Removed obsolete metaclass.
- **Five**: Refactored `browser:view` and `browser:page` directives. This makes their implementation more similar to that in `zope.browserpage` and adds allowed interface support for the `browser:view` directive. By default the `aq_*` attributes are no longer available on those views/pages. If you still use them, you have to mix in Five’s BrowserView.
- **Removed** the (very obsolete) thread lock around the cookie parsing code in HTTPRequest.py; the python `re` module is thread-safe, unlike the ancient `regex` module that was once used here.
- **Removed** the special handling of `Set-Cookie` headers in `HTTPResponse.setHeader`. Use the `setCookie/appendCookie/expireCookie` methods instead, or if low-level control is needed, use `addHeader` instead to get the exact same effect.
- **Removed** the `App.version_txt.getZopeVersion` API, you can use `pkg_resources.get_distribution('Zope2').version` instead.
- **On the application object**, removed `PrincipiaTime` in favor of `ZopeTime` and `PrincipiaRedirect` in favor of `Redirect` or `ZopeRedirect`.
- **Removed** `OFS.DefaultObservable` - an early predecessor of `zope.event`.
- **Removed** `mime-types` option from `zope.conf`. You can use the `add_files` API from `zope.contenttype` instead.
- **Removed** `OFS.ZDOM`. `OFS.SimpleItem.Item` now implements `getParentNode()`.
- **Removed** the last remaining code to support `SOFTWARE_HOME` and `ZOPE_HOME`.
- **Removed** ZMI controls for restarting the process, these no longer apply when managed as a WSGI application.
- **Removed** `bobobase_modification_time` from `Persistence.Persistent`, you can use `DateTime(object._p_mtime)` instead.
- **Removed** `AccessRule` and `SiteRoot` from `Products.SiteAccess`.
- **Removed** `Products.ZReST` and the `reStructuredText` wrapper, you can use `docutils` directly to gain reST support.
• Removed special code to create user folders and page templates while creating new `OFS.Folder` instances.

• Removed persistent default code like the `error_log` and `temp_folder`.

• Removed persistent default content, including the `standard_error_message` template.

• Retired icons from the `Zope Management Interface` and various smaller cleanups of ZMI screens.

• Removed the old help system, in favor of the current Sphinx documentation hosted at http://docs.zope.org/zope2/. For backwards compatibility the `registerHelp` and `registerHelpTitle` methods are still available on the ProductContext used during the `initialize` function.

• Removed various persistent product related code and options. The `enable-product-installation zope.conf` setting is now a no-op.

• Changed the value for `default-zpublisher-encoding` and `management_page_charset` to `utf-8`.

• Removed the `enable-ms-author-via` directive which was only required for very old web folder implementations from before 2007.

• Changed `zope.conf` default settings for `zserver-threads` to 2 and `python-check-interval` to 1000.

• Simplified instance skeleton, removing old `Extensions`, `import`, `lib/python` and `Products` from the default. You can continue to manually add these back.

• `Five.browser`: Marked `processInputs` and `setPageEncoding` as deprecated. `processInputs` was replaced by the `postProcessInputs` request method and the charset negotiation done by `setPageEncoding` was never fully supported.

• Dropped the direct dependencies on packages that have been factored out of the main Zope 2 tree. Make sure you declare a dependency in your own distribution if you still use one of these: `Products.BTreeFolder2`, `Products.ExternalMethod`, `Products.MailHost`, `Products.MIMETools`, `Products.PythonScripts` or `Products.StandardCacheManagers`. 