Plone-Entwicklerhandbuch

Release 1.213

Veit Schiele

28.04.2023

Inhaltsverzeichnis

Das Plone-Entwicklerhandbuch möchte anhand eines konkreten Projekts aufzeigen, wie Plone den eigenen Bedürfnissen angepasst werden kann. Gleichzeitig kommt dabei die Überzeugung des Autors von agiler Software-Entwicklung, vor allem testgetriebener Entwicklung, zum Ausdruck.

Plone ermöglicht die schnelle Erstellung von funktionsreichen und performanten Websites, da es bereits mächtige Werkzeuge zur Verwaltung von Rechten und Inhalten bereitstellt. So ist es einfach, öffentliche Websites, Intra- und Extranets sowie branchenspezifische Web-Anwendungen schnell und zuverlässig zu erstellen.

Dabei wird Plone von einer großen internationalen Community getragen. Zudem hält die Plone-Foundation als eine nonprofit-Organisation die geistigen Eigentums- und Markenrechte, sodass der weitere Entwicklungsprozess von Plone stabil und kontrolliert möglich ist.

Plone basiert auf dem objektorientierten Web-Application-Framework Zope und steht für viele Betriebssysteme wie Windows, Mac OS X und Linux/Unix zur Verfügung.

Plone ist in Python geschrieben, einer mächtigen und dennoch einfach zu nutzenden Programmiersprache.

Dieses Plone-Entwicklerhandbuch wird anhand einer Fallstudie aufzeigen, wie ein prototypischer Verlauf eines Software-Entwicklungsprozesses mit Plone aussehen kann.

KAPITEL 1

Zielgruppe

Das Buch richtet sich in erster Linie an Entwickler, die ihr webbasiertes Content Management System mit Plone realisieren möchten. Zumindest einige Kenntnis von Python, HTML und CSS werden erwartet, auch eigene frühere Erfahrungen mit Zope und Plone können hilfreich sein.

KAPITEL 2

Konventionen

In diesem Buch werden unterschiedliche Schriftstile zur Differenzierung der Inhalte verwendet. So sieht die Darstellung von Codeabschnitten so aus:

```
[buildout]
parts =
    zope2
    productdistros
    instance
    zopepy
```

während Angaben im Terminal so dargestellt werden:

```
$ paster create -t plone3_buildout
```

bzw als root:

```
# curl -O http://peak.telecommunity.com/dist/ez_setup.py
```

KAPITEL 3

Inhalt

3.1 Entwicklungsumgebung

3.1.1 Konzepte

Buildout erlaubt, identische Entwicklungsumgebungen einfach aufzusetzen. Hierzu nutzt `buildout` die Fähigkeit der setuptools, automatisch Abhängigkeiten aufzulösen und Aktualisierungen durchzuführen (s.a.: Jim Fulton: Buildout Tutorial).

Python Eggs sind ein Deploymentformat für Python-Packages. Sie enthalten ein `setup.py`-Skript mit Metainformationen (Lizenz, Abhängigkeiten, etc.) Mit der Python-Bibliothek *Setuptools* können solche Abhängigkeiten automatisch nachgeladen werden, wobei in Eggs spezifische Versionen angegeben werden können.

Python Package Index PyPI (aka Cheese Shop) Index mit tausenden von Python-Paketen. Setuptools, `easy_install` und Buildout nutzen diesen Index, um Eggs automatisch zu installieren.

EasyInstall Python-Modul mit dem der *Python Package Index* durchsucht werden kann und das die Pakete in die globale Python-Umgebung installiert. Wir werden nur Buildout mit EasyInstall installieren, alle weiteren Eggs werden von Buildout in das lokale Buildout-Projekt heruntergeladen, unter anderem um Versionskonflikte zu vermeiden.

3.1.2 Installation

Bevor `zc.buildout` installiert werden kann, sind folgende Schritte erforderlich:

1. Installation von Python.
    1. Anforderungen
        - bash oder eine andere Shell
        - ein C- und C++-Compiler
        - GNU make

- Zope setzt das Python-zlib-Modul voraus, das Python beim Kompilieren erstellt sofern zlib.h in `/usr/include` installiert ist.

Für diese Anforderungen müssten auf Debian- und Ubuntu-Systemen folgende Pakete installiert werden:

```
$ sudo apt-get install build-essential zlib1g-dev
```

oder:

```
$ sudo yum install make gcc-c++ zlib-devel
```

Falls Sie die SSL-Bibliotheken benötigen, z.B. zum Verschicken von Mails mit TLS, sollten Sie auf Debian- und Ubuntu-Systemen das OpenSSL- Paket installieren:

```
$ sudo apt-get install libssl-dev
```

oder:

```
$ sudo yum install openssl-devel
```

Mir der GNU Readline-Bibliothek können Sie im Python-Prompt die zuletzt eingegebenen Befehle erneut aufrufen:

```
$ sudo apt-get install libreadline6-dev
```

Für die Indizierung von Word- und PDF-Dokumenten werden darüberhinaus noch die folgenden Pakete benötigt:

```
$ sudo apt-get install wv poppler-utils
```

oder:

```
$ sudo yum install wv poppler-utils
```

2. Erstellen und Installieren

Bemerkung: Im Folgenden wird Python 2.7 für Plone 4.3 installiert. Für Plone 4.1 und 4.0 benötigen Sie jedoch Python 2.6 und für Plone 3 Python 2.4. Die Installationsschritte für moderne Linux-Installationen unterscheiden sich jedoch, sodass sie besonders behandelt werden.

```
$ sudo apt-get install python-dev python-libxml2 python-libxslt1 python-
↪virtualenv
```

oder:

```
$ sudo yum install python27-devel python-lxml python-virtualenv
```

Anschließend können Sie ein Virtual Environment erstellen und darin `zc.buildout` installieren:

```
$ virtualenv --system-site-packages venv
$ cd venv
$ ./bin/pip install zc.buildout
```

Alternativ können Sie selbst Python aus den Sourcen kompilieren. Gehen Sie hierzu in das Verzeichnis, in dem Sie die aktuelle Python- Version installieren möchten, z.B. in `/opt/`. Anschließend laden Sie das Python-Paket herunter und entpacken es:

```
# curl -O http://www.python.org/ftp/python/2.7.9/Python-2.7.9.tgz
# tar -xvzf Python-2.7.9.tgz
```

Installieren Sie das Python-2.7.9-Paket:

```
# cd Python-2.7.9
# ./configure --prefix=/opt/python/2.7.9
```

`--prefix` Python-Installationspfad. Ohne Angabe wird Python in `/usr/local` erstellt.

Anschließend fahren Sie mit der Installation fort:

```
# make
# make install
```

2. Der Python-Interpreter kann nun in `PATH` eingetragen werden. Hierzu wird folgendes in die `~/.bashrc` (oder auf dem Mac in `~/.bash_profile`) eingetragen:

```
export PATH=/opt/python/2.7.9/bin:$PATH
```

Danach kann die Konfiguration neu eingelesen werden mit:

```
$ source ~/.bashrc
```

3. Buildout benötigt zusätzlich auch noch `easy_install`:

```
# mkdir /opt/python/2.7.9/Extensions
# cd $_
# curl -O http://peak.telecommunity.com/dist/ez_setup.py
# python ez_setup.py
```

Welche Versionen dieser Python-Pakete mit `easy_install` installiert wurde, erfahren Sie indem Sie das jeweilige Skript aufrufen mit der Option `--version`.

Wollen Sie zu einem späteren Zeitpunkt eines dieser Python-Pakete aktualisieren, so können Sie dies mit der Option `-U`, also z.B.:

```
# easy_install -U setuptools
```

Sie können auch spezifische Versionen angeben, z.B.:

```
# easy_install setuptools==0.9.8
```

Bemerkung: Verwenden Sie z.B. Subversion 1.5 zusammen mit Buildout und erhalten folgende Fehlermeldung:

```
NameError: global name 'log' is not defined
```

dann benötigen Sie mindestens die `dev06`-Version der setuptools. Dies erhalten Sie mit:

```
# easy_install setuptools>=dev06
```

Mac OS X

1. Installieren der OSX development tools (XCode).
2. Installieren von Macports.
3. Um `bootstrap.py` aufzurufen, sollte folgender Befehl verwendet werden um zu gewährleisten, dass der Python-Interpreter von Macports verwendet wird:

```
$ python2.7 bootstrap.py
```

Weitere Informationen

Buildout hinter einem Proxy Häufig kann Buildout nicht direkt auf die Quellen zugreifen um Python Eggs o.ä.herunterzuladen. In diesem Fall sollte der Proxy zunächst als Environment-Variable z.B. in der `~/.bashrc`-Datei angegeben werden:

```
export http_proxy = http://localhost:8123/
export https_proxy = http://localhost:8123/
```

Alternativ kann auch über einen ssh-Tunnel auf den entfernten Server zugegriffen werden:

```
$ ssh -L 8123:localhost:8123 yourserver.com
```

Setzen des `LD_LIBRARY_PATH` `LD_LIBRARY_PATH` ist eine Unix-Environment-Variable, die angibt, aus welchem Verzeichnis dynamisch verlinkte Bibliotheken (`*.so`-Dateien) geladen werden sollen. Falls die systemweit verfügbaren Bibliotheken überschrieben werden sollen, kann dies mit `environment-vars` aus dem `zope2instance`-Rezept geschehen:

```
[instance]
...
# Use statically compiled libxml2
environment-vars =
    LD_LIBRARY_PATH ${buildout:directory}/parts/lxml/libxml2/lib:$
↪{buildout:directory}/parts/lxml/libxslt/lib
```

s.a. Issue 11715: Building Python on multiarch Debian and Ubuntu - Python tracker.

Python 2.6

Für Python 2.6 müssen zunächst einige Dateien geändert werden bevor sie auf moderneren Linux-Distributionen mit `multiarch`-Features, die manche Bibliotheken in architekturspezifischen Verzeichnissen speichern, so z.B. in `/usr/lib/x86_64-linux-gnu/libz.so`.

1. Zunächst wird die Python-Distribution heruntergeladen und entpackt:

```
# wget https://www.python.org/ftp/python/2.6.9/Python-2.6.9.tar.xz
# tar xvzf Python-2.6.9.tgz
```

2. Anschließend wechseln wir in dieses Verzeichnis und ergänzen in der Datei `setup.py` die `lib_dirs` um `/usr/lib/x86_64-linux-gnu`, sodass der Abschnitt anschließend folgendermaßen aussieht:

```
lib_dirs = self.compiler.library_dirs + [
    '/lib64', '/usr/lib64',
    '/lib', '/usr/lib',
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    '/usr/lib/x86_64-linux-gnu'
    ]
```

3. Anschließend ändern wir in der Datei `Modules/_ssl.c` den Abschnitt mit `PySSL_BEGIN_ALLOW_THREADS`:

```
PySSL_BEGIN_ALLOW_THREADS
if (proto_version == PY_SSL_VERSION_TLS1)
    self->ctx = SSL_CTX_new(TLSv1_method()); /* Set up context */
else if (proto_version == PY_SSL_VERSION_SSL3)
    self->ctx = SSL_CTX_new(SSLv3_method()); /* Set up context */
else if (proto_version == PY_SSL_VERSION_SSL23)
    self->ctx = SSL_CTX_new(SSLv23_method()); /* Set up context */
PySSL_END_ALLOW_THREADS
```

4. In derselben Datei sind auch noch die Protokoll-Versionen anzupassen:

```
/* protocol versions */
PyModule_AddIntConstant(m, "PROTOCOL_SSLv3",
                        PY_SSL_VERSION_SSL3);
PyModule_AddIntConstant(m, "PROTOCOL_SSLv23",
                        PY_SSL_VERSION_SSL23);
PyModule_AddIntConstant(m, "PROTOCOL_TLSv1",
                        PY_SSL_VERSION_TLS1);
```

5. Auch in `Lib/ssl.py` sind die Protokoll-Versionen noch anzupassen:

   Die Zeile

```
from _ssl import PROTOCOL_SSLv2, PROTOCOL_SSLv3, PROTOCOL_SSLv23, PROTOCOL_TLSv1
```

   sollte ersetzt werden durch

```
from _ssl import PROTOCOL_SSLv3, PROTOCOL_SSLv23, PROTOCOL_TLSv1
```

6. Nun kann Python 2.6 konfiguriert und erstellt werden mit

```
# env CPPFLAGS="-I/usr/lib/x86_64-linux-gnu" LDFLAGS="-L/usr/include/x86_64-linux-
↪gnu"  ./configure --prefix=/opt/python/2.6.9
# make
# make install
```

7. Schließlich kann noch EasyInstall installiert werden mit:

```
# cd /opt/python/2.4.6/
# mkdir Extensions
# cd $_
# wget https://pypi.python.org/packages/2.4/s/setuptools/setuptools-0.6c11-py2.4.
↪egg#md5=bd639f9b0eac4c42497034dec2ec0c2b
# export PATH=/opt/python/2.4.6/bin:$PATH
# sh setuptools-0.6c11-py2.4.egg
```

Python 2.4

Auch für Python 2.4 müssen zunächst einige Änderungen vorgenommen werden bevor es auf moderneren Linux- oder Debian-Distributionen mit sog. `multiarch`- Architektur lauffühig ist.

1. Zunächst wird die Python-Distribution heruntergeladen und entpackt:

```
# wget https://www.python.org/ftp/python/2.4.6/Python-2.4.6.tgz
# tar xvzf Python-2.4.6.tgz
```

2. Anschließend wechseln wir in dieses Verzeichnis und ergänzen in der Datei `setup.py` die `lib_dirs` um `'/usr/lib/x86_64-linux-gnu'`, sodass der Abschnitt anschließend folgendermaßen aussieht:

```
lib_dirs = self.compiler.library_dirs + [
    '/lib64', '/usr/lib64',
    '/lib', '/usr/lib',
    '/usr/lib/x86_64-linux-gnu'
    ]
```

3. Weiter unten in derselben Datei sind die Pfade nochmals anzupassen in `ssl_libs`:

```
ssl_libs = find_library_file(self.compiler, 'ssl',lib_dirs,
                             ['/usr/local/ssl/lib',
                              '/usr/contrib/ssl/lib/',
                              'x86_64-linux-gnu'
                             ] )
```

4. Nun wird Python 2.4 konfiguriert und erstellt mit:

```
# env CPPFLAGS="-I/usr/lib/x86_64-linux-gnu" LDFLAGS="-L/usr/include/x86_64-linux-
↪gnu"  ./configure --prefix=/opt/python/2.4.6
# make
# make install
```

5. Schließlich kann noch EasyInstall installiert werden mit:

```
# cd /opt/python/2.4.6/
# mkdir Extensions
# cd $_
# wget https://pypi.python.org/packages/2.4/s/setuptools/setuptools-0.6c11-py2.4.
↪egg#md5=bd639f9b0eac4c42497034dec2ec0c2b
# export PATH=/opt/python/2.4.6/bin:$PATH
# sh setuptools-0.6c11-py2.4.egg
```

Erstellen eines Buildout-Projekts

1. Ein Buildout-Projekt für Plone lässt sich am einfachsten aus meiner vs_buildout-Vorlage bei github erstellen:

```
$ curl -o master.zip  https://codeload.github.com/veit/vs_buildout/zip/master
$ unzip master.zip
$ cd vs_buildout-master
$ python bootstrap.py -c devel.cfg
...
Got distribute 0.6.28.
Generated script '/Users/plone/vs_buildout-master/bin/buildout'.
```

-d verwendet Distribute anstatt der Setuptools.

-c erlaubt die Angabe eines Pfades zu einer Buildout-Konfigurationsdatei, in unserem Fall `devel.cfg`.

Bemerkung: Falls das `buildout`-Skript für den Nutzer, z.B. `plone`, noch nicht in `PATH` eingetragen wurde, ändern wir die `~/.bashrc` (oder auf dem Mac in `~/.bash_profile`) folgendermaßen:

```
export PATH=/opt/python/Python-2.7.5/bin/:$PATH
```

Danach kann die Konfiguration neu eingelesen werden mit:

```
$ source ~/.bashrc
```

---

2. Bevor nun Plone mit Buildout installiert werden kann, sollten die für die Python Imaging Library (PIL) benötigten Bibliotheken installiert werden.

   Anforderungen:

   - Für JPEG-Unterstützung benötigen Sie die *IJG JPEG library,* Version 6a oder 6b:

     http://www.ijg.org

   - Für PNG- und ZIP-Unterstützung benötigen Sie die *ZLIB library.*

     http://www.info-zip.org/pub/infozip/zlib/

   - Für TrueType/OpenType-Unterstützung benötigen Sie die *FreeType 2.0 library:*

     http://www.freetype.org

   Unter Debian und Ubuntu können Sie die Pakete installieren mit:

   ```
   $ sudo apt-get install libjpeg62-dev libfreetype6
   ```

   Unter Fedora und CentOS können Sie die Pakete installieren mit:

   ```
   $ sudo yum install libjpeg-turbo-devel freetype-devel
   ```

   Unter Mac OS X können Sie die Pakete z.B. mit Homebrew installieren:

   ```
   $ brew install freetype jpeg libtiff
   ```

   Sofern diese Anforderungen erfüllt sind, wird die PIL mit folgendem Eintrag in der `base.cfg`-Datei installiert:

   ```
   [buildout]
   ...
   versions = versions
   ...
   eggs =
       Pillow

   [versions]
   ...
   Pillow = 1.7.8
   ```

3. Schließlich wird ab Plone 4.2 auch lxml benötigt.

   Unter Debian und Ubuntu können Sie die Pakete installieren mit:

   ```
   $ sudo apt-get install libxml2-dev libxslt1-dev
   ```

   Unter Fedora und CentOS können Sie die Pakete installieren mit:

```
$ sudo yum install libxml2-devel libxslt-devel
```

4. Nun kann Buildout aufgerufen werden:

```
$ ../venv/bin/buildout -c devel.cfg
...
--------------------------------------------------------------------
SETUP SUMMARY (Pillow 1.7.8 / PIL 1.1.7)
--------------------------------------------------------------------
version      1.7.8
platform     darwin 2.7.3 (default, Feb  8 2013, 10:02:27)
             [GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)]
--------------------------------------------------------------------
--- TKINTER support available
--- JPEG support available
--- ZLIB (PNG/ZIP) support available
--- FREETYPE2 support available
*** LITTLECMS support not available

--------------------------------------------------------------------
```

   Dieser Prozess kann längere Zeit dauern, da Zope, Plone und alle Zusatzprodukte heruntergeladen und installiert werden.

5. Ist der Prozess abgeschlossen, kann der Zope-Server gestartet werden mit:

```
$ ./bin/instance start
```

   Und das Stoppen des Zope-Servers geht mit:

```
$ ./bin/instance stop
```

   Schlägt das Starten des Zope-Servers fehl, können Sie den Zope-Server im Vordergrund starten und bekommen dann auf der Konsole ausgegeben, an welcher Stelle Zope den Startvorgang abbricht:

```
$ ./bin/instance fg
```

   Mit `STRG-c` kann dieser Prozess wieder beendet werden.

6. Schließlich sollten Sie noch den `admin`-Zugang ersetzen. Hierzu starten Sie zunächst die Instanz und gehen dann in den *User Folder* des Zope Management Interface (ZMI): `http://localhost:8080/acl_users/manage`.

   Hier können Sie unter `http://localhost:8080/acl_users/manage_users` einen neuen Nutzer anlegen und diesem die Rolle *Manager* zuweisen.

   Anschließend können im ZMI `Logout` auswählen und sich gleich anschließend wieder mit den neuen Zugangsdaten anmelden.

   Nun sollten Sie noch den `admin`-Nutzer löschen.

Zusätzliche Informationen für Windows

Installation und Konfiguration von Python 2.6.6 und 2.4.4, MinGW, Libxml- und Libxslt-Python-Bindings.

Python 2.6.6

1. Herunterladen und Installieren von Python 2.6.6 mit python-2.6.6.msi.

   Wählen Sie *Install for all users*.

   Der Standard-Installationsort ist `C:\Python26`.

1. Herunterladen und Installieren der pywin32-Extension von http://sourceforge.net/projects/pywin32/files/pywin32/Build%20214/pywin32-214.win32-py2.6.exe/download.

1. Eintragen von Python in die Systemvariable `PATH`, sodass nicht jedesmal der gesamte Pfad angegeben werden muss.

   1. Öffnen Sie die *Systemeigenschaften* und klicken anschließend zunächst auf den *Erweitert*-Reiter, dann auf *Umgebungsvariablen*.

   2. Fügen Sie anschließend die Pfade zu Ihrer Python-Installation ein, z.B.:

      ```
      C:\Python26;C:\Python26\Scripts;
      ```

      Beachten Sie bitte, dass die verschiedenen Pfade durch Semikolon voneinander getrennt sind:

   3. Öffnen Sie nun eine neue Shell mit `Windows-r` und geben `cmd` in das Popup-Fenster ein.

   4. Mit `python -V` wird Ihnen die Versionsnummer des verwendeten Python ausgegeben – dies sollte `Python 2.6.6` sein.

Python 2.4.4

1. Herunterladen und Installieren von Python 2.4.4 mit python-2.4.4.msi;

   Wählen Sie *Install for all users*.

   Der Standard-Installationsort ist `C:\Python24`.

2. Herunterladen und Installieren der pywin32-Extension von http://downloads.sourceforge.net/pywin32/pywin32-210.win32-py2.4.exe.

1. Eintragen von Python in die Systemvariable `PATH`, sodass nicht jedesmal der gesamte Pfad angegeben werden muss.

   1. Öffnen Sie die *Systemeigenschaften* und klicken anschließend zunächst auf den *Erweitert*-Reiter, dann auf *Umgebungsvariablen*.

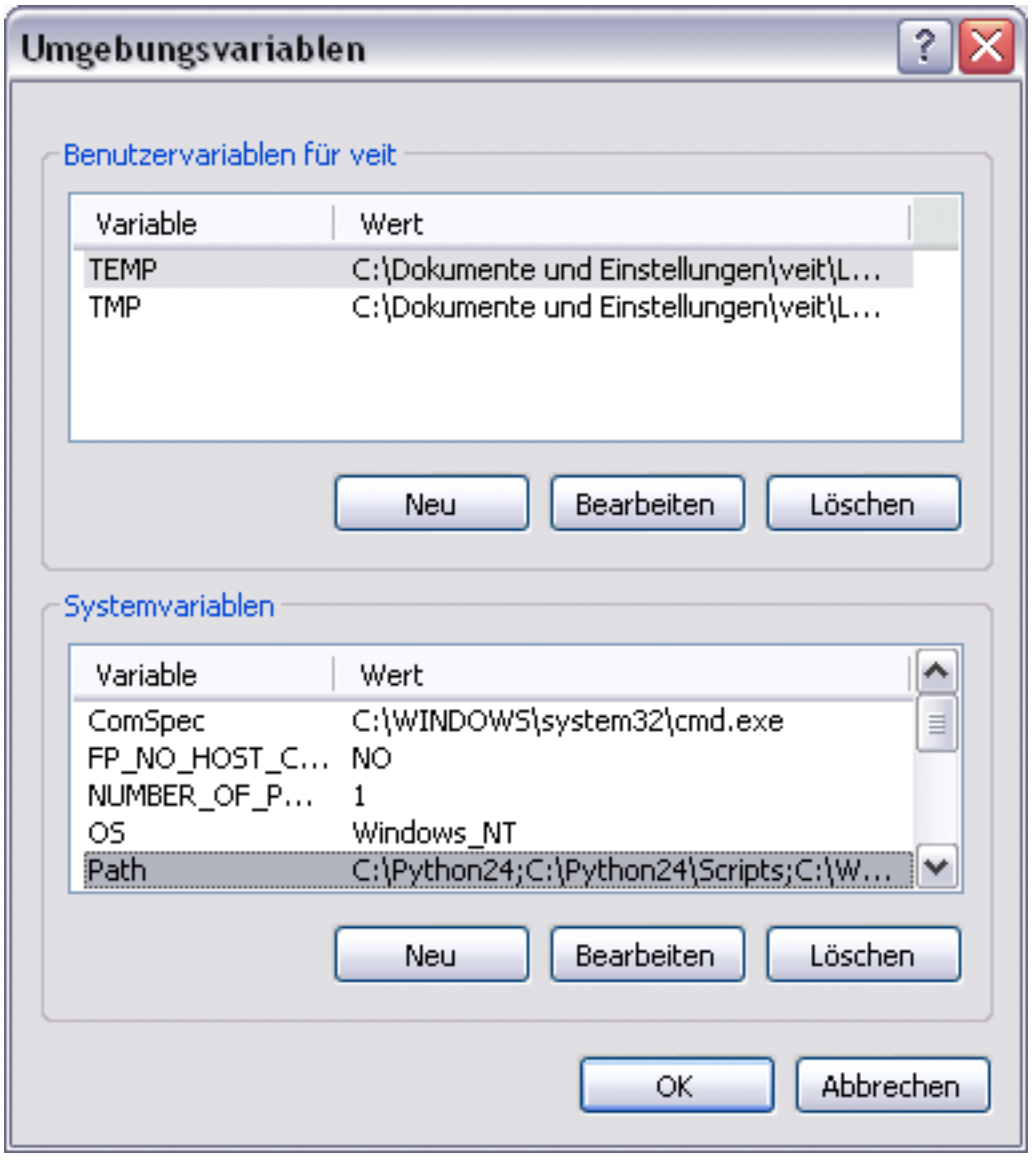
   2. Fügen Sie anschließend den Pfad zu Ihrer Python-Installation ein, z.B.:

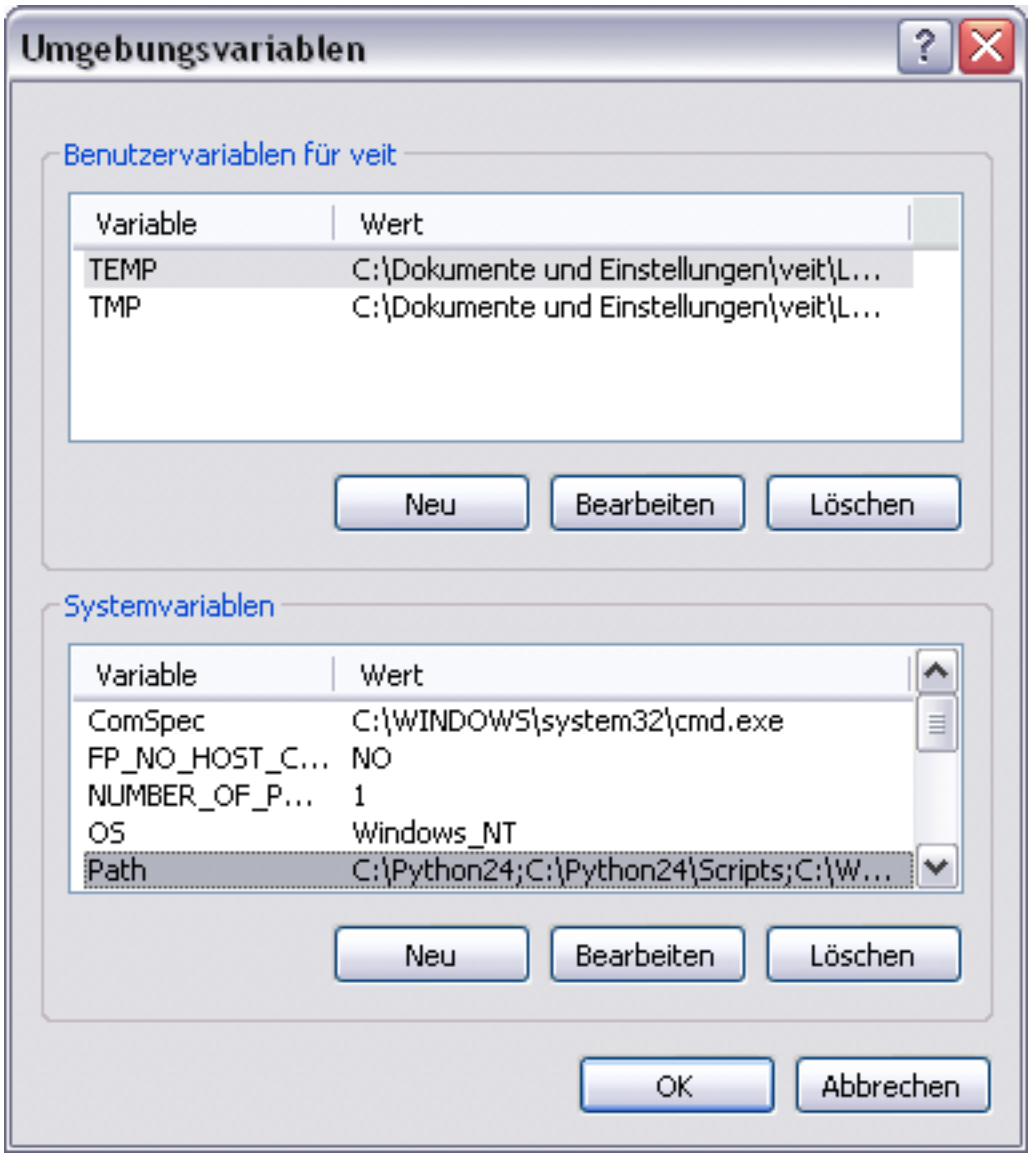
      ```
      C:\Python24;C:\Python24\Scripts;
      ```

      Beachten Sie bitte, dass die verschiedenen Pfade durch Semikolon voneinander getrennt sind.

   3. Öffnen Sie nun eine neue Shell mit `Windows-r` und geben `cmd` in das Popup-Fenster ein.

   4. Mit `python -V` wird Ihnen die Versionsnummer des verwendeten Python ausgegeben – dies sollte `Python 2.4.4` sein.


Umgebungsvariablen
Benutzervariablen für veit
Variable
Wert
TEMP
C:\Dokumente und Einstellungen\veit\L...
TMP
C:\Dokumente und Einstellungen\veit\L...
Neu
Bearbeiten
Löschen
Systemvariablen
Variable
Wert
ComSpec
C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...
NO
NUMBER_OF_P...
1
OS
Windows_NT
Path
C:\Python24;C:\Python24\Scripts;C:\W...
Neu
Bearbeiten
Löschen
OK
Abbrechen


Umgebungsvariablen
Benutzervariablen für veit
Variable
Wert
TEMP
C:\Dokumente und Einstellungen\veit\L...
TMP
C:\Dokumente und Einstellungen\veit\L...
Neu
Bearbeiten
Löschen
Systemvariablen
Variable
Wert
ComSpec
C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...
NO
NUMBER_OF_P...
1
OS
Windows_NT
Path
C:\Python24;C:\Python24\Scripts;C:\W...
Neu
Bearbeiten
Löschen
OK
Abbrechen

MinGW

Dies ist ein gcc-Compiler für Windows, womit C-Komponenten von Zope auf Windows kompiliert werden können.

1. Herunterladen des MinGW-Installationsprogramms von http://downloads.sourceforge.net/mingw/MinGW-5.1.4.exe.
2. Geben Sie beim Ausführen des Installationsprogramms als Optionen *MinGW base tools* und *MinGW Make* an.

   Das Programm wird üblicherweise nach `C:\MinGW` installiert.

   Das Installationsprogramm holt sich die benötigten Dateien von sourceforge.net, wobei es gegebenenfalls mehrfach aufgerufen werden muss bis alle Dateien heruntergeladen wurden.
3. Tragen Sie nun `C:\MinGW\bin` in `PATH` ein.
4. Testen Sie die Installation in einer Shell mit:

   ```
   gcc --version
   ```

   Die Ausgabe sollte `gcc (GCC) 3.4.5` oder neuer sein.
5. Anschließend wird Distutils für MinGW konfiguriert. Hierzu erstellen Sie die Datei `distutils.cfg` in `C:\Python24\Lib\distutils` mit folgendem Inhalt:

   ```
   [build]
   compiler=mingw32
   ```

Libxml- und Libxslt-Python-Bindings

1. Die Bindings können heruntergeladen werden von http://users.skynet.be/sbi/libxml-python/binaries/libxml2-python-2.7.7.win32-py2.6.exe.

Python Imaging Library (PIL)

Tragen Sie in der Buildout-Konfigurationsdatei statt `PIL` bitte Pillow ein, also:

```
[instance]
...
eggs =
    Pillow
    Plone
    ...
```

Windows-Service

Nun können Sie ein Buildout-Projekt erstellen, wobei jedoch im Pfad keine Leer- oder Sonderzeichen enthalten sein dürfen.

Anschließend können Sie die Zope-Instanz als Windows-Service installieren, indem Sie in der Shell folgendes angeben:

```
> bin\instance.exe install
```

Anschließend lässt sich die Instanz starten mit:

```
> bin\instance.exe start
```

Soll der Service wieder entfernt werden, geben Sie einfach folgendes an:

```
> bin\instance.exe remove
```

Buildout-Konfiguration

Die `base.cfg`-Datei gliedert sich in die folgenden Abschnitte:

`[buildout]` Hier werden die globalen Einstellungen für diesen Buildout angegeben.

`parts` Die in dieser Konfiguration angegebenen Abschnitte, die in der angegebenen Reihenfolge durchlaufen werden:

```
parts =
    instance-base
    zopepy
    i18ndude
    zopeskel
```

`extends` Eine bestehende Buildout-Konfiguration wird erweitert, nämlich `http://dist.plone.org/release/4.3/versions.cfg` und `versions.cfg`.

`find-links` URLs, Datei- oder Verzeichnisnamen, in denen Buildout nach Links zu Distributionen suchen soll.

`extensions` Erweiterungen, die den Funktionsumfang von Buildout vergrößern:

- mr.developer
- buildout.threatlevel
- jarn.setuptoolsfixer

`allow-picked-versions` Mit dem Wert `false` kann gewährleistet werden, dass alle Versionen festgeschrieben wurden.

`versions` Es wird auf einen Abschnitt verwiesen, in dem die Versionen der Python-Packages festgeschrieben werden können, in unserem Fall ist der Abschnitt `versions` genannt worden. Weiter Hinweise zur Verwaltung von Versionen erhalten Sie in Aktualisierung und Versionierung.

`unzip` Üblicherweise packt Buildout keine gezippten Python Eggs aus.

`[instance-base]` Dieser Abschnitt erstellt und konfiguriert eine Zope-Instanz unter Verwendung von plone.recipe.zope2instance:

```
[instance-base]
recipe = plone.recipe.zope2instance
user = admin:admin
http-address = 8080
debug-mode = on
verbose-security = on
blob-storage = var/blobstorage

eggs =
    Plone
    Pillow
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
zcml =

environment-vars =
    PTS_LANGUAGES en de
    zope_i18n_allowed_languages en de
    zope_i18n_compile_mo_files true
```

eggs Hier können zusätzliche Python-Eggs angegeben werden, wobei `elementtree` von Plone benötigt wird. Auf diese Eggs wird später in `[instance]` verwiesen.

Es können auch spezifische Versionen angegeben werden. Soll z.B. `SQLAlchemy 0.3` installiert werden, sieht der Eintrag so aus:

```
eggs =
    ...
    SQLAlchemy>=0.3,<0.4dev
```

Auch die in diesem Projekt entwickelten Eggs werden hier angegeben:

```
eggs =
    ...
    my.package

develop =
    src/my.package
```

verbose-security Damit lässt sich angeben,

- Für welche Objekte wurde der Zugriff verweigert?
- Welche Rechte sind notwendig?
- Welche Rollen und Rechte sind den Objekten und Eigentümern zugeteilt?
- Welches sind die wirksamen Proxy-Rollen?

Damit wir alle `unauthorized`-Ereignisse auch im Fehlerprotokoll angezeigt bekommen, müssen wir später im *Site Error Log* im Zope Management Interface (ZMI) der Plone-Site `Unauthorized` aus der Liste *Ignored exception types* entfernen.

eggs Hier geben wir die der Instanz zur Verfügung stehenden Eggs an. In unserer Konfiguration werden die in den `buildout`- und `plone`-Abschnitten angegebenen Eggs verwendet.

zcml Da die Konfigurationsdateien nicht automatisch für ältere Eggs oder Pakete geladen werden, kann Buildout angewiesen werden, einen sog. *ZCML slug* in `parts/instance/etc/package-includes` zu erstellen, indem die entsprechenden Pakete unter dieser Option aufgelistet werden:

```
zcml =
    my.package
```

Es kann auch explizit angegeben werden, welche Art von *ZCML slug* erstellt werden soll, z.B.:

```
zcml =
    my.package-overrides
    my.package-meta
```

overrides Dies erstellt eine `*-overrides.zcml`-Datei in `myproject/parts/instance/etc/package-includes/`, mit der sich eine per `zcml` angegebene Konfiguration wieder überschrieben wird.

Anschließend wird in der `configure.zcml`-Datei von `my.package` eine `overrides`-Konfigurationsdatei eingefügt:

```
<includeOverrides file="overrides.zcml" />
```

Diese `overrides.zcml`-Datei enthält dann die Ersetzung einer bestehenden Konfiguration.

meta Dies erstellt eine `*-meta.zcml`-Datei in `myproject/parts/instance/etc/package-includes/`, die gewährleistet, dass die gesamte Konfiguration dieses Pakets zur Verfügung steht bevor die weiteren `zcml`-Anweisungen abgearbeitet werden.

Weitere Konfigurationsoptionen von `plone.recipe.zope2instance` sind:

default-zpublisher-encoding Liefert ein Request eine Unicode-Antwort und ist für `ZPublisher.HTTPResponse` kein spezifischer Zeichensatz angegeben, dann wird der Unicode-String mit dem `default-zpublisher-encoding` kodiert.

Der Standardwert ist `utf-8`.

zope-conf Ein relativer oder absoluter Pfad zu einer Zope-Konfigurationsdatei. Eine `zope.conf`-Datei wird dann mit den Angaben in diesem Abschnitt generiert in `parts/instance/etc/zope.conf`.

zope-conf-additional Sollen nur die Werte einiger Attribute der `zope.conf`-Datei geändert werden, können diese in `zope-conf-additional` angegeben werden. Dabei müssen die nachfolgenden Zeilen eingerückt sein.

environment-vars definiert Umgebungsvariablen zur Laufzeit von Zope, z.B.:

```
environment-vars =
    zope_i18n_compile_mo_files = true
```

Einen vollständigen Überblick über alle Optionen des `[instance]`-Abschnitts erhalten Sie in plone.recipe.zope2instance.

[zopepy] In diesem Abschnitt wird ein Python-Interpreter definiert, der alle Eggs und Pakete, aber keine Zope2-Produkte enthält und sich daher gut zum Debuggen und Testen eignet:

```
[zopepy]
recipe = zc.recipe.egg
eggs = ${instance:eggs}
interpreter = zopepy
extra-paths = ${zope2:location}/lib/python
scripts = zopepy
```

Mit dem Rezept wird das `./bin/zopepy`-Skript erstellt und sowohl die Eggs aus dem `[instance]`-Abschnitt als auch die Zope-Module aus `parts/zope2/lib/python` der Zope-Installation eingeschlossen. Es muss also nicht mit jedem neuen Buildout-Projekt auch die `PYTHONPATH`-Umgebungsvariable neu gesetzt werden. Mit `zopepy` sollte sich z.B. auch einfach das Modul `PageTemplates` aus `Products` importieren lassen:

```
$ ./bin/zopepy
>>> from Products import PageTemplates
```

Da kein Fehler für den Import angegeben wurde, wird das Modul geladen, und der Python-Interpreter kann mit `Strg-D` (unter Windows `Strg-Z`) wieder verlassen werden.

annotate

Mit der Buildout-Option `annotate` werden alle Abschnitte alphabetisch sortiert angezeigt. Innerhalb jedes Abschnitts werden alle Schlüssel-Wert-Paare zusammen mit der Quelle angezeigt. Eine solche Quelle kann entweder ein Dateiname oder die Variablen `COMPUTED_VALUE`, `DEFAULT_VALUE` oder `COMMAND_LINE_VALUE` sein. Die Ausgabe kann z.B. folgendermaßen aussehen:

```
$ ./bin/buildout -c deploy.cfg  annotate
Setting socket time out to 3 seconds.

Annotated sections
==================

[backup]
enable_snapshotrestore= false
    /home/veit/sandboxes/vs_buildout/deploy.cfg
...
[buildout]
...
develop-eggs-directory= develop-eggs
    DEFAULT_VALUE
directory= /home/veit/vs_buildout
    COMPUTED_VALUE
...
```

`annotate` kann auch genutzt werden um herauszufinden, welche Version aufgrund welcher Konfiguration verwendet wird, z.B.:

```
[versions]
...
six= 1.2.0
    /home/veit/vs_buildout/plone-versions.cfg
...
```

Plone 3.2

Für Plone 3.2 sieht die Buildout-Konfigurationsdatei etwas anders aus. Sie gliedert sich in die folgenden Abschnitte:

```
parts =
    zope2
    productdistros
    instance
    zopepy
```

`[zope2]` Dieser Abschnitt lädt und erstellt Zope 2 aus der im Abschnitt `plone` angegebenen URL:

```
[zope2]
recipe = plone.recipe.zope2install
fake-zope-eggs = true
additional-fake-eggs =
    ZODB3
url = ${versions:zope2-url}
```

Mit dem Rezept wird Zope 2 in `parts/zope2` installiert, die Variable `ZOPE_HOME` ist also `parts/zope2` und die Variable `SOFTWARE_HOME` `parts/zope2/lib/python`.

fake-zope-eggs Falls der Wert auf `true` gesetzt wird, werden Links auf die Zope-3-Bibliotheken gesetzt. Wenn nun ein Egg in seiner `setup.py`-Datei auf ein `zope.*`-Egg verweist, finden die setuptools diese in `/parts/zope2/lib/python/zope/` und installieren nicht erneut Versionen dieser Eggs in womöglich inkompatiblen Versionen. Ab Version 3 ist der Standardwert auf `true` gesetzt.

additional-fake-eggs Hiermit lasst sich eine Liste zusätzlicher *fake eggs* angeben, wobei nur Python-Packages angegeben werden sollten, die sich auch in `PYTHONPATH` befinden. Der Standardwert schließt `Acquisition`, `ClientForm`, `DateTime`, `docutils`, `ExtensionClass`, `mechanize`, `Persistence`, `pytz`, `RestrictedPython`, `tempstorage`, `ZConfig`, `zLOG`, `zodbcode`, `ZODB3`, `zdaemon` und `Zope2` ein.

Die Versionen von `additional-fake-eggs` lassen sich einfach angeben, z.B.:

```
additional-fake-eggs =
    ZODB3 = 3.7.1
    zope.annotation = 3.3.2
```

Wird keine Version für `additional-fake-eggs` angegeben, haben die *faked eggs* immer die Version `0.0`.

skip-fake-eggs Hier kann eine Liste von Packages angegeben werden, für die keine Fake eggs erstellt werden sollen. Somit können neuere Versionen spezifischer Zope-Packages installiert werden auch wenn `fake-zope-eggs = true` gesetzt ist, z.B.:

```
[buildout]
versions = versions

[versions]
zope.app.catalog = 3.5.2
zope.component = 3.5.1
zope.i18n = 3.6.0
zope.sendmail = 3.5.1
zope.testing = 3.7.1
five.intid = 0.3.0

[zope2]
fake-zope-eggs = true
additional-fake-eggs =
    ZConfig
    ZODB3
    pytz
skip-fake-eggs =
    zope.component
    zope.i18n
    zope.sendmail
    zope.testing
```

url Die URL, unter der Zope heruntergeladen werden kann, in unserem Fall wird auf die `versions.cfg`-Datei verwiesen und den dort angegebenen Wert für `zope2-url`.

[productdistros] Der Abschnitt kann verwendet werden, um Archive von Produkten herunterzuladen und zu installieren, z.B.:

```
urls =
    http://www.zope.org/Members/shh/DocFinderTab/1.0.2/DocFinderTab-1.0.2.tar.gz
```

nested-packages Archive, die mehrere Zope2-Produkte enthalten.

Im folgenden Beispiel soll PloneLDAP 1.0 installiert werden:

```
[productdistros]
recipe = plone.recipe.distros
urls =
    http://plone.org/products/ploneldap/releases/1.0/PloneLDAP-bundle-1.0.tar.
↪gz
nested-packages =
    PloneLDAP-bundle-1.0.tar.gz
version-suffix-packages =
```

Nach dem Aufruf von `./bin/buildout` finden sich die Produkte `LDAPMultiPlugins`, `LDAPUserFolder` und `PloneLDAP` in `parts/productdistros`.

`version-suffix-packages` Produkte, deren Verzeichnisnamen die Version enthält und die daher vor ihrer Verwendung umbenannt werden müssen.

In den Abschnitten `products` in `[instance]` wird dann auf den Installationsort von `productdistros` verwiesen:

```
${productdistros:location}
```

`[instance]` Dieser Abschnitt erstellt und konfiguriert eine Zope-Instanz unter Verwendung von `plone.recipe.zope2instance`:

```
[instance]
recipe = plone.recipe.zope2instance
zope2-location = ${zope2:location}
...
products =
    ${buildout:directory}/products
    ${productdistros:location}
```

`zope2-location` Es wird das im `zope2`-Abschnitt angegebene Verzeichnis für die Zope2-Installation verwendet.

Plone 3.1

Für Plone 3.1 sieht die `buildout.cfg`-Datei etwas anders aus:

```
[buildout]
parts =
...
plone
```

`index` URL eines Index-Servers. In diesem Index sucht Buildout sofern in den unter `find-links` angegebenen Distributionen nichts gefunden wurde.

Ohne spezifische Angabe für `index` wird der Python Package Index verwendet. Aus Gründen der Stabilität und Performance kann sich jedoch ein anderer Index empfehlen:

```
index = http://download.zope.org/ppix
```

`[plone]` verwendet plone.recipe.plone um die Plone Produkte und Eggs herunterladen:

```
[plone]
recipe = plone.recipe.plone
```

Dabei ist zu beachten, dass immer die aktuellste Version verwendet wird. Soll immer nur ein Plone-3.1.x-Release verwendet wird, wird beim Erstellen des Buildout-Projekts zunächst auf die Frage `Enter plone_version` mit `3.1` geantwortet, und anschließend kann man sich zunutze machen, dass die Versionsnummern des `plone.recipe.plone` immer mit denen von Plone übereinstimmen:

```
recipe = plone.recipe.plone>=3.1,<3.2dev
```

Und für ein bestimmtes Plone-Release sieht die Angabe so aus:

```
recipe = plone.recipe.plone==3.1.7
```

Das `plone`-Rezept gibt jeweils passende Zope-Versionen, Produkte und Eggs an, die in den Abschnitten `[zope2]` und `[instance]` mit den Buildout-Variablen `${plone:zope2-url}`, `${plone:eggs}` und `${plone:products}` referenziert werden.

Aktualisierung und Versionierung

Aktualisierung

Damit Änderungen der buildout.cfg wirksam werden, müssen wir `./bin/buildout` erneut aufrufen. Diese Aktualisierung kann häufig beschleunigt werden, indem buildout im *non-updating*-Modus aufgerufen wird, also:

```
$ ./bin/buildout -N
```

Eine Übersicht über alle für `buildout` verfügbaren Optionen erhalten Sie mit:

```
$ ./bin/buildout -h
```

Versionen festschreiben

zc.buildout < 2.0 verwendet üblicherweise immer die neueste Version eines Eggs. Sollen jedoch nur finale Versionen verwendet werden, kann im buildout-Abschnitt folgendes angegeben werden:

```
prefer-final = true
```

zc.buildout < 2.0 aktualisiert üblicherweise immer auf die neueste Version. Dies unterbleibt jedoch, wenn im buildout-Abschnitt folgendes angegeben wird:

```
newest = false
```

Darüberhinaus kann `buildout` eine Fehlermeldung ausgeben, wenn Versionen noch nicht festgeschrieben wurden mit:

```
allow-picked-versions = false
```

In den oben angegebenen Beispielen wurde gezeigt, wie sich die Versionen für Eggs, Recipes und Products fest vorgeben lassen.

Eine Liste der noch nicht festgeschriebenen Versionen erhalten wir in buildout 2.0 mit:

```
[buildout]
...
show-picked-versions = true
```

In früheren Buildout-Versionen können wir diese Angabe erhalten mit:

```
$ ./bin/buildout -Nv | sed -ne 's/^Picked: //p' | sort | uniq
```

Ab Plone 4.1 werden die benötigten Versionen festgeschrieben indem in der `buildout.cfg`-Datei auf eine externe `versions.cfg`-Datei angegeben wird:

```
[buildout]
...
extends =
   http://dist.plone.org/release/4.1/versions.cfg

versions = versions
```

In der Datei `http://dist.plone.org/release/4.1/versions.cfg` werden dann im `[versions]`-Abschnitt die von Plone benötigten Eggs in definierten Versionen angegeben:

```
[buildout]
extends = http://download.zope.org/zopetoolkit/index/1.0.3/zopeapp-versions.cfg
          http://download.zope.org/Zope2/index/2.13.8/versions.cfg

[versions]
# Buildout infrastructure
mr.developer = 1.17
...

# External dependencies
...

# Plone release
Plone                                = 4.1
Products.ATContentTypes              = 2.1.3
...
```

Falls weitere Eggs festgeschrieben werden sollen, kann nicht in der `buildout.cfg`-Datei ein weiterer `versions`-Abschnitt angegeben werden, es kann jedoch in `extends` eine eigene `versions.cfg`-Datei angegeben werden, die ebenfalls wieder einen `[versions]`-Abschnitt enthält. So kann z.B. die `buildout.cfg`-Datei folgendermaßen aussehen:

```
[buildout]
...
extends =
    http://dist.plone.org/release/4.1/versions.cfg
    versions.cfg

versions = versions
```

Und in der `versions.cfg`-Datei werden dann weitere Versionen festgeschrieben:

```
[versions]
...
```

Plone 3.1

Sollen spezifische, in einem Rezept angegebene Versionen überschrieben werden, wird zunächst im `buildout`-Abschnitt mit der Option `versions` auf einen Abschnitt verwiesen, der die zu verwendenden Versionen enthält:

```
[buildout] ... versions = release1

[release1] plone.locking = 1.0.5
```

Anschließend muss das Egg noch im `[plone]`-Abschnitt angegeben werden, um die Festlegung der Version für dieses Produkt in `plone.recipe.plone` aufzuheben:

```
[plone]
recipe = plone.recipe.plone
eggs =
    plone.locking
```

Siehe auch: Repeatable buildouts: controlling eggs used

buildout.dumppickedversions

buildout.dumppickedversions ist eine `buildout-extension` für zc.buildout < 2.0, die alle Eggs, deren Versionen bisher nicht festgeschrieben wurden, in einer `versions.cfg`-Datei festschreiben kann. Eine Beispielkonfiguration kann z.B. so aussehen:

```
[buildout]
...
extensions =
    buildout.dumppickedversions
dump-picked-versions-file = versions.cfg
overwrite-picked-versions-file = false
```

Sofern noch nicht vorhanden, wird nun beim Aufruf von `./bin/buildout` eine `versions.cfg`-Datei erzeugt, die z.B. so aussehen kann:

```
[versions]
Cheetah = 2.2.1
Paste = 1.7.5.1
PasteScript = 1.7.3
ZopeSkel = 2.19
i18ndude = 3.2.2

#Required by:
#PasteScript 1.7.3
PasteDeploy = 1.3.4

#Required by:
#i18ndude 3.2.2
ordereddict = 1.1
```

Kontrollierte Updates

z3c.checkversions findet neuere Versionen der im Buildout-Projekt verwendeten Python-Pakete.

Es kann folgendermaßen installiert werden:

```
[buildout]
parts =
    ...
    checkversions
...
[checkversions]
recipe = zc.recipe.egg
eggs = z3c.checkversions [buildout]
```

Buildout-Erweiterungen

extends

Im Buildout-Abschnitt können mit `extends` mehrere Konfigurationsdateien eingebunden werden. Auf diese Weise können dann auch umfangreiche Konfigurationen, wie z.B. die Installation der libxml2- und libxslt-Bibliotheken in eine eigene Konfigurationsdatei `lxml.cfg` mit folgendem Inhalt ausgelagert werden:

```
[lxml]
parts =
   staticlxml
   pylxml

[pylxml]
recipe=zc.recipe.egg
interpreter=pylxml
eggs=
    lxml

[staticlxml]
recipe = z3c.recipe.staticlxml
egg = lxml
```

Anschließend kann diese Konfigurationsdatei mit all ihren Abschnitten in die `buildout.cfg`-Datei eingebunden werden mit:

```
[buildout]
extends =
    lxml.cfg

parts =
    ${lxml:parts}
    ...
```

Es kann auch eine URL angegeben werden, also z.B.:

```
[buildout]
extends =
    http://www.plone-entwicklerhandbuch.de/plone-entwicklerhandbuch/
↪entwicklungsumgebung/lxml.cfg
```

Umgekehrt kann auch die `buildout.cfg`-Datei in eine andere Konfiguration übernommen werden, siehe hierzu Buildout für Produktivserver.

Setuptool-Bugfix

Mit jarn.setuptoolsfixer wird ein Bug in den Setuptools behoben, der auftritt sofern die Homepage oder Download-URL eines Pakets nicht erreichbar ist, das Paket jedoch in PyPI zur Verfügung steht.

Shell-Befehle

Mit plone.recipe.command können Sie eigene Shell-Befehle während der Installation oder des Updates durchführen. Somit können Sie zum Beispiel der Zope-Instanz externe Methoden im Verzeichnis `parts/instance/Extension` zur Verfügung stellen:

```
[extensions]
recipe = plone.recipe.command
command =
    ln -sf ${buildout:directory}/Extensions/*  ${instance:location}/Extensions/
update-command =
    ${extensions:command}
```

Python-Skripte

buildout.extensionscripts erlaubt die Verwendung von Python-Skripten als Buildout-Erweiterungen.

Die Buildout-Konfiguration kann dann z.B. so aussehen:

```
[buildout]
extensions =
    ...
    buildout.extensionscripts
...
extension-scripts =
    ${buildout:directory}/buildout-utils.py:patchScriptGeneration
```

Und `buildout-utils.py` kann dann z.B. so aussehen:

```
# Workaround for https://bugs.launchpad.net/zc.buildout/+bug/164629

def patchScriptGeneration(buildout):
    from zc.buildout import easy_install
    if not 'sys.exit(' in easy_install.script_template:
        easy_install.script_template = easy_install.script_template.replace(
            "%(module_name)s.%(attrs)s(%(arguments)s)",
            "sys.exit(%(module_name)s.%(attrs)s(%(arguments)s))")
```

User-crontab

Das Rezept `z3c.recipe.usercrontab` ändert die *crontab*-Einträge des Nutzers. So kann z.B. für den `@reboot`-Eintrag folgendes in der `buildout.cfg`-Datei angegeben werden:

```
[buildout]
...
parts =
    ...
    crontab

[crontab]
recipe = z3c.recipe.usercrontab
times = @reboot
command = ${buildout:directory}/bin/instance start
```

Dabei kann eine Buildout-Konfigurationsdatei auch mehrere *crontab*-Abschnitte enthalten.

Templates verwenden

Mit collective.recipe.template lassen sich Textdateien aus Vorlagen erstellen wobei die `buildout`-Variablen verwendet werden können. Hierzu wird in der `buildout.cfg`-Datei z.B. folgender neuer Abschnitt definiert:

```
[buildout]
parts =
   ...
   logrotate

...
[logrotate]
recipe = collective.recipe.template
input = templates/logrotate.conf
output = ${buildout:directory}/etc/logrotate.conf
```

Und wenn ein Auszug aus `templates/logrotate.conf`-Datei so aussieht:

```
...
${buildout:directory}/var/log/instance.log {
    postrotate
        ${buildout:bin-directory}/instance logreopen
    endscript
}
```

sieht dieser Auszug in der generierten Datei `myproject/etc/logrotate.conf` so aus:

```
...
/home/veit/myproject/var/log/instance.log {
    postrotate
        /home/veit/myproject/instance logreopen
    endscript
}
```

Verzeichnisstruktur

```
vs_buildout
├── README.rst
├── base.cfg
├── bootstrap.py
├── deploy.cfg
├── devel.cfg
├── develop-eggs
│   └── vs.policy.egg-link
├── docs
│   └── HISTORY.rst
├── eggs
│   ├── AccessControl-2.13.10-py2.7-macosx-10.4-x86_64.egg
│   ├── Acquisition-2.13.8-py2.7-macosx-10.4-x86_64.egg
│   ├── ...
├── etc
│   ├── logrotate.conf
│   └── plone.vcl
├── rsync.cfg
├── src
│   ├── README.txt
│   └── vs.policy
├── templates
│   ├── haproxy.conf.in
│   ├── logrotate.conf.in
│   └── plone.vcl.in
├── var
│   ├── blobbackup-blobstorages
│   ├── blobbackup-blobstoragesnapshots
│   ├── blobstorage
│   │   └── tmp
│   ├── filebackup-snapshots
│   ├── filebackups
│   ├── filestorage
│   │   ├── Data.fs
│   │   ├── Data.fs.index
│   │   ├── Data.fs.lock
│   │   └── Data.fs.tmp
│   ├── instance
│   │   └── import
│   ├── instance-base
│   │   └── import
│   └── log
│       └── zeoserver.log
├── versions.cfg
└── vs_buildout.txt
```

bin/ enthält ausführbare Dateien, u.a. `buildout` und das Zope-Kontrollskript `instance`.

bootstrap.py Bootstrap-Skript des Buildout-Projekts.

***.cfg** Konfigurationsdateien von Buildout, siehe Buildout-Konfiguration.

develop-eggs/ Verweise auf Eggs, die in diesem Buildout-Projekt entwickelt werden sollen und die in der `buildout.cfg` angegeben wurden.

downloads/ Rezepte wie `plone.recipe.plone` und `plone.recipe.distros` laden ihre Archive von Produkten und Paketen in diesem Verzeichnis herunter.

eggs/ Eggs, die Buildout automatisch heruntergeladen hat. Aktiviert werden die Eggs explizit in den Kontrollskripten im `bin`-Verzeichnis.

fake-eggs/ In Buildout-Projekten für Plone 3.1 enthält dieses Verzeichnis die sog. `fake-zope-eggs`, wobei in der `*.egg-info`-Datei die Version angegeben werden kann.

.installed.cfg Buildout speichert die aktuellen Konfigurationsdaten in dieser Datei.

parts/ Dateien, die für die jeweiligen Abschnitte der Buildout-Konfiguration verwendet werden.

products/ Für ältere Plone-Versionen lassen sich hier die Zope-Produkte erstellen, die in diesem Buildout-Projekt entwickelt werden.

src/ Eggs, die in diesem Buildout-Projekt entwickelt werden.

var/ var-Dateien der Zope-Instanz: z.B. ZODBs in `filestorage/` Log-Dateien in `logs/` und `zopectlsock` und kompilierte Übersetzungsdateien in `instance-base`.

Ein solches Buildout-Projekt kann anderen Entwicklern in einem Repository zur Verfügung gestellt werden, wobei die Verzeichnisse `bin`, `eggs`, `download`, `var` und `parts` ignoriert werden können, da sie sich mit `bootstrap.py` wiederherstellen lassen.

Ressourcen teilen

Verzeichnisse teilen

Da wir mehrere Projekte mit verschiedenen Buildout-Konfigurationen betreuen, wollen wir die Sourcen zwischen den Projekten teilen.

So ist es möglich, Eggs verschiedener Versionen in einem Verzeichnis bereitzustellen. Ein solches Verzeichnis kann in der Buildout-Konfigurationsdatei `buildout.cfg` im Abschnitt `[buildout]` eingetragen werden, z.B.:

```
[buildout]
eggs-directory = /home/veit/.buildout/eggs
```

Analog teilen wir das `downloads`-Verzeichnis zwischen verschiedenen Buildout-Projekten:

```
download-cache = /home/veit/.buildout/downloads
```

Ab Version 1.4.1 von `zc.buildout` lässt sich auch ein geteiltes Verzeichnis für die heruntergeladenen Dateien angeben:

```
extends-cache = /home/veit/.buildout/cache
```

Um Fehlermeldungen zu vermeiden sollten die zu verwendenden Verzeichnisse vor dem ersten Durchlauf von Buildout erstellt werden, z.B. mit `mkdir -p /home/veit/.buildout/{eggs,downloads,cache}`.

Distributionen finden

Indizes

Üblicherweise sucht Buildout im Python Package Index nach Distributionen. Es können jedoch auch auch weitere Indizes angegeben werden mit:

```
[buildout]
...
index = https://pypi.org/simple/
```

Dieser Index oder falls kein Index angegeben ist `https://pypi.python.org/simple/` wird zur Suche nach der passenden Distribution verwendet. Dabei wird immer die letzte passende Version einer angeforderten Distribution heruntergeladen.

Links

Neben Indizes kann auch die `find-links`-Option angegeben werden um Distributionen zu finden.

Dabei können sowohl URLs als auch Pfadangaben verwendet werden:

```
[buildout]
...
find-links =
    http://download.zope.org/distribution/
    /some/path
    /other/path/someegg-1.0.0-py2.4.egg
```

Globale Variablen für alle Buildout-Projekte

Um diese Konfiguration nicht für jedes Buildout-Projekt erneut angeben zu müssen, kann eine `default.cfg`-Datei im Verzeichnis `~/.buildout` angelegt werden:

```
[buildout]
eggs-directory = /home/veit/.buildout/eggs
download-cache = /home/veit/.buildout/downloads
extends-cache = /home/veit/.buildout/cache
index = https://pypi.python.org/simple
socket-timeout = 3
```

Weitere Entwicklungswerkzeuge

Wollen wir weitere Entwicklungswerkzeuge in einem Buildout-Projekt installieren, geben wir diese einfach in der `buildout.cfg`-Datei an. Folgende Entwicklungswerkzeuge können die Arbeit deutlich vereinfachen:

DocFinderTab Produkt, das alle Klassen und Methoden eines Objekts im Zope Management Interface (ZMI) auflistet.

DocFinderTab kann direkt als Egg in der Instanz angegeben werden:

```
[instance]
...
eggs =
    Products.DocFinderTab
```

pyflakes Pyflakes analysiert Python-Programme und entdeckt verschiedene Fehlerarten. Es ist sehr viel schneller als das Ausführen der Programme.

pyflakes lässt sich einfach mit Buildout installieren:

```
parts =
    ...
    pyflakes

[pyflakes]
recipe = zc.recipe.egg:scripts
eggs = pyflakes
scripts = pyflakes
entry-points = pyflakes=pyflakes.scripts.pyflakes:main
```

plone.app.debugtoolbar Debug-Toolbar, die einfach für eine Plone-Site aktiviert werden kann. Die Installation kann einfach mit Buildout erfolgen:

```
[instance]
...
eggs =
    plone.app.debugtoolbar
```

pylint Pylint analysiert Python-Code in Bezug auf Bugs und geringe Code-Qualität.

Pylint lässt sich einfach mit Buildout installieren:

```
parts =
    pylint
    ...

[pylint]
recipe = zc.recipe.egg
eggs =
    ${instance:eggs}
    pylint
entry-points = pylint=pylint.lint:Run
arguments = sys.argv[1:]
```

DeadlockDebugger Für entsprechende Prozesse wird Debugging möglich, indem ein Traceback aller laufenden Pythonprozesse sowohl zum Eventlog als auch zum Browser geschickt wird.

PDBDebugMode PDBDebugMode erlaubt sog. post-mortem-Debugging für *exceptions* im Debug-Modus, d.h., bei einem Fehler wird der Debugger aufgerufen, der den Traceback ausgibt. Sofern vorhanden, nutzt `PDBDebugMode` `ipdb` statt `pdb`.

Diese Entwicklungswerkzeuge lassen sich einfach angeben mit:

```
[instance]
...
debug-mode = on
eggs =
    Products.PDBDebugMode
    z3c.deadlockdebugger
```

Products.PrintingMailHost Monkey Patch, der MailHost-Nachrichten nicht verschickt, sondern auf der Konsole ausgibt, d.h., Zope versendet damit keine Mails mehr.

roadrunner Testrunner, der die testgetriebene Entwicklung deutlich beschleunigen kann.

`roadrunner` läd vorab das Standard-Zope- und Plone-Environment für PloneTestCase. zur Installation wird einfach folgendes in die `buildout.cfg`-Datei eingetragen:

```
[buildout]
parts =
    ...
    roadrunner

[roadrunner]
recipe = roadrunner:plone
packages-under-test = vs.policy
```

Anschließend kann es wie der reguläre Zope-Testrunner aufgerufen werden:

```
$ ./bin/roadrunner -s vs.policy
```

collective.recipe.grp Rezept, mit dem in Buildout auf `${grp:GROUP}` referenziert werden kann um die Gruppe des aktuellen Nutzers herauszubekommen.

Zusammen mit gocept.recipe.env, das Environment-Variablen in einem Buildout-Abschnitt zur Verfügung stellt, lassen sich hiermit die Eigentümer (*Owner*) der Buildout-Inhalte setzen lassen, z.B. mit:

```
chown -R ${env:USER}:${grp:GROUP} ${buildout:directory}
```

Installieren lassen sich die Pakete mit:

```
[env]
recipe = gocept.recipe.env

[grp]
recipe = collective.recipe.grp
```

IPython Python-Shell, die Ihnen u.a. folgende Vorteile bietet:

- Objekt-Introspektion
- Code- Introspektion
- Dokumentation-Introspektion (mit `%pdoc`)
- Eingabehistorie, persistent auch über Sessions hinweg.

Zur Installation fügen Sie bitte folgendes in Ihrer `devel.cfg`-Datei hinzu:

```
[buildout]
_
parts =
    ...
    ipzope
...
   [ipzope]
   # An IPython Shell for interactive use with Zope running.
   #
   # It requires the `ipy_profile_zope.py` configuration script. Get this from
   # git@github.com:collective/dotipython.git and put it in your profile
   # directory. Depending on your setup, this may be at
   # `$HOME/.ipython/profile_zope/startup`,
   # `$HOME/.config/ipython/profile_zope/startup` (Ubuntu 12.04), or see
   # http://ipython.org/ipython-doc/dev/config/overview.html#configuration-file-
↪location
   # for more details.
   #
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
recipe = zc.recipe.egg
eggs =
    ipython
    ${instance}
initialization =
    import sys, os
    os.environ["INSTANCE_HOME"] = "${instance:location}"
    sys.argv[1:1] = "--profile=zope".split()
scripts = ipython=ipzope
```

Rufen Sie dann zunächst das buildout-Skript auf. Anschließend können Sie dann die IPython-Sell aufrufen:

```
$ ./bin/buildout
$ ./bin/ipzope
```

Beim ersten Aufruf von `ipzope` wird ein neues `IPython`-Profil in Ihrem Home-Verzeichnis erstellt. In *ix-Betriebssystemen finden Sie das entsprechende Verzeichnis unter `$HOME/.ipython/`, in Windows unter `%userprofile%\_ipython`. In dieses Verzeichnis sollten das Profil aus `https://github.com/collective/dotipython/blob/master/ipy_profile_zope.py` legen. Anschließend sollten Sie die IPython-Session mit `Ctrl-d` beenden und erneut starten.

Anschließend lässt sich z.B. `portal.error_log.get` eingeben und durch Drücken der *Tab*-Taste erhalten Sie alle verfügbaren Methoden des *èrror_log`*, die mit `get` beginnen.

Falls Sie Änderungen an Ihrer Plone-Site vorgenommen haben, können Sie diese speichern mit:

```
utils.commit()
```

Und falls auch andere auf der Zope-Instanz arbeiten, sollten Sie gelegentlich die Änderungen übernehmen mit:

```
utils.sync()
```

Weitere Informationen zu iPython erhalten Sie im iPython-Tutorial.

ipdb, iw.debug `ipdb` ist Python-Debugger, der viele Vorteile von IPython nutzt, z.B. automatische Vervollständigung. `iw.debug` erlaubt Ihnen, den `ipdb`- Debugger über jedem veröffentlichten Objekt einer Zope2-Anwendung aufzurufen.

Zum Installieren fügen Sie in Ihrer `devel.cfg`-Datei folgendes hinzu:

```
[buildout]
_
[instance]
eggs +=
    ...
    iw.debug
_
[instance]
_
zcml +=
    iw.debug
```

Anschließend wird das Buildout-Skript aufgerufen und die Instanz im Vordergrund gestartet:

```
$ ./bin/buildout
$ ./bin/instance fg
```

Bemerkung: Wenn in Ihrem Code an irgendeiner Stelle ein `ipdb` oder `pdb` Code enthalten ist, erhalten Sie die Exception `BdbQuit`.

Nun kann der URL eines jeden Objekts der Plone-Site `/ipdb` angehängt werden um eine IPython-Shell für diese Plone-Site zu erhalten:

```
...
--Return--
None
> /Users/veit/.buildout/eggs/iw.debug-0.3-py2.7.egg/iw/debug/pdbview.py(92)pdb()
     91             else:
---> 92                 set_trace()
     93
```

Um die lokalen Variablen zu erhalten, können Sie nun zunächst `ll` eingeben:

```
ipdb> ll
{'request': <HTTPRequest, URL=http://localhost:8080/Plone/ipdb>, 'portal':
↪<PloneSite at /Plone>, 'context': <PloneSite at /Plone>, 'meth': None, 'view':␣
↪None}
ipdb> context
<PloneSite at /Plone>
ipdb> context == portal
True
ipdb> portal.Title()
'Website'
ipdb> portal.portal_quickinstaller.listInstallableProducts()
[{'status': 'new', 'hasError': False, 'id': 'plone.app.dexterity', 'title': u
↪'Dexterity Content Types'}, {'status': 'new', 'hasError': False, 'id': 'plone.
↪app.theming', 'title': u'Diazo theme support'}, {'status': 'new', 'hasError':␣
↪False, 'id': 'plone.app.caching', 'title': u'HTTP caching support'}, {'status':
↪'new', 'hasError': False, 'id': 'Marshall', 'title': 'Marshall'}, {'status':
↪'new', 'hasError': False, 'id': 'plone.app.openid', 'title': u'OpenID␣
↪Authentication Support'}, {'status': 'new', 'hasError': False, 'id': 'plone.app.
↪debugtoolbar', 'title': u'Plone debug toolbar'}, {'status': 'new', 'hasError':␣
↪False, 'id': 'plone.session', 'title': u'Session refresh support'}, {'status':
↪'new', 'hasError': False, 'id': 'plone.resource', 'title': u'Static resource␣
↪storage'}, {'status': 'new', 'hasError': False, 'id': 'CMFPlacefulWorkflow',
↪'title': u'Workflow Policy Support (CMFPlacefulWorkflow)'}, {'status': 'new',
↪'hasError': False, 'id': 'plone.app.iterate', 'title': u'Working Copy Support␣
↪(Iterate)'}, {'status': 'new', 'hasError': False, 'id': 'collective.z3cform.
↪datetimewidget', 'title': u'collective.z3cform.datetimewidget'}]
ipdb>
```

Zope-Neustarts

Früher wurden Änderungen in Templates oder Python-Skripten im Debug-Modus oder mit refresh.txt einfach übernommen, und ein Neustart von Zope war nur selten nötig.

Heute mag es häufig so erscheinen, als ob Zope bei jeder kleinen Änderung neu gestartet werden muss, damit die Änderungen auch übernommen werden. Ich gebe hier nun mal einen Überblick, welche Änderungen wie übernommen werden:

- Wird die Instanz im Debug-Modus gestartet, werden für Änderungen an Page-Templates und Zope-3-Browser-Ressourcen keine Neustarts benötigt.

- Die Generic-Setup-XML-Dateien werden bei jedem Import durch das Generic-Setup-Tool ausgelesen.
- Auch die `Install.py`-Dateien in einem `Extensions`-Verzeichnis, die vom Portal-Quickinstaller verwendet werden, können ohne Neustart verändert werden.
- Und auch alle Dateien im `skins`-Ordner, einschließlich der Python-Skripte, werden ohne Neustart aktualisiert.
- Andere Python-Skripte, also z.B. `setuphandlers.py`, werden jedoch nur bei einem Neustart aktualisiert. Hier schafft sauna.reload Abhilfe, indem es Code und ZCML-Dateien nachlädt. Um `sauna.reload` zu installieren, geben Sie folgendes in der `buildout.cfg`-Datei an:

```
[buildout]
...
eggs =
    ...
    sauna.reload

[instance]
...
zope-conf-additional =
    %import sauna.reload
```

Nach dem Aufruf des buildout-Skripts sollte die Instanz gestartet werden mit:

```
$ RELOAD_PATH=src/ ./bin/instance fg
```

Anschließend können Sie sich im ZMI anmelden und folgende URL aufrufen:

```
http://localhost:8080/@@saunareload
```

- Darüberhinaus helfen pyflakes und PDBDebugMode schon vor einem Neustart die Fehler zu finden.
- Und schließlich wird bei Test Driven Development nicht der Zope-Server selbst sondern nur der Testrunner aufgerufen ;-)

Bemerkung: Achten Sie auch darauf, dass die Resource Registries CSS-, JavaScripts- und KSS-Registry im Debug/development-Modus betrieben werden, damit die zugehörigen Dateien nicht gecachet und entsprechende HTTP-Header ausgeliefert werden.

Bemerkung: Unter Windows muss PyYAML unter Verwendung der Binärdateien installiert werden.

---

mr.developer

mr.developer ist eine Buildout-Erweiterung, mit der sich Projekte, die über mehrere Repositories verteilt sind, verwalten lassen.

`mr.developer` wird in der `extensions`-Option im `[buildout]`-Abschnitt hinzugefügt. Anschließend können folgende weitere Optionen festgelegt werden:

`sources-dir` Das Verzeichnis, in das die Pakete heruntergeladen werden.

Der Standardwert ist `src`.

`sources` Liste der Repository-Informationen der Pakete.

Das Format ist `<kind> <url> [key=value]`.

kind svn, hg oder git

url Die URL des Repository

key=value Hier können Optionen für jedes einzelne Paket angegeben werden.

Es können weder Leerzeichen in key noch in value noch um das Gleichheitszeichen herum verwendet werden.

Im Folgenden einige der gebräuchlichsten Optionen:

path Optionale Angabe des Pfads, in den das Paket ausgecheckt wird, wobei der Name des Pakets dem Pfad angehängt wird.

Wird keine Angabe für path getroffen, wird stattdessen sources-dir verwendet.

full-path Angabe des Pfades, in das ein Paket ausgecheckt wird ohne dass der Paketname angehängt wird.

update spezifiziert, ob ein Paket beim Durchlaufen von Buildout aktualisiert werden soll oder nicht. Die Angabe überschreibt die globale always-checkout-Anweisung.

egg Diese Option erlaubt, die Verwaltung von Paketen, die keine Python-Eggs sind mit egg=false. Dann wird das Paket nicht der develop-Option von Buildout hinzugefügt.

auto-checkout Pakete, die beim initialen Aufruf des buildout-Skripts automatisch ausgecheckt werden. * kann angegeben werden falls alle Pakete in sources ausgecheckt werden sollen.

always-checkout Der Standardwert ist false.

true Alle in auto-checkout angegebenen Pakete, die im *develop mode* sind, werden aktualisiert sobald das buildout-Skript aufgerufen wird.

force Wie bei true, jedoch werden auch die als dirty markierten Pakete ohne Rückfrage aktualisiert.

Hier ein Beispiel für einen solchen Eintrag in die buildout.cfg-Datei:

```
[buildout]
...
extensions = mr.developer
sources = sources
auto-checkout =
    vs.policy
    some.package
    bootstrap
always-checkout = true

[sources]
vs.policy = svn https://svn.veit-schiele.de/svn/vs.policy/trunk
some.package = git git://example.com/git/some.package.git
bootstrap = git git://github.com/twitter/bootstrap.git␣
↪rev=d9b502dfb876c40b0735008bac18049c7ee7b6d2 path=${buildout:directory} egg=false
```

Buildout erzeugt nun ein Skript bin/develop, das verschiedene Aktionen zu den einzelnen Paketen erlaubt, wie z.B. das Auschecken des Quellcodes ohne den Ort des Repositories kennen zu müssen. Für weitere Aktionen geben Sie einfach folgendes ein:

```
$ ./bin/develop help
```

Wird der Quellcode aus einem Paket ausgecheckt, muss buildout erneut durchlaufen werden. Das Paket wird dann automatisch als *develop egg* markiert, und falls es in der Liste der versions-Option festgeschrieben wurde wird dieser Eintrag gelöscht und das *develop egg* verwendet.

Die Liste der *develop eggs* kann mit den `activate`- und `deactivate`-Kommandos von `bin/develop` gesteuert werden.

3.2 Erstellen eines Site-Policy-Produkts

1. Zum Erstellen eines Python-Eggs, das ein Zope2-Produkt enthält, verwenden wir das ZopeSkel-Template `plone`:

```
$ cd src
$ $ ../bin/zopeskel plone_basic
...
Enter project name (or q to quit): vs.policy
Expert Mode? (What question mode would you like? (easy/expert/all)?) ['easy']: all
Namespace Package Name (Name of outer namespace package) ['vs']:
Package Name (Name of the inner namespace package) ['policy']:
Version (Version number for project) ['1.0']:
Description (One-line description of the project) ['']: Policy package for␣
↪demonstration purposes
Register Profile (Should this package register a GS Profile) [False]: True
Long Description (Multi-line description (in ReST)) ['']: Policy package for␣
↪demonstration purposes
Author (Name of author for project) ['']: Veit Schiele
Author Email (Email of author for project) ['']: kontakt@veit-schiele.de
Keywords (List of keywords, space-separated) ['']: Zope Plone
Project URL (URL of the homepage for this project) ['http://svn.plone.org/svn/
↪collective/']: https://github.com/veit/vs.policy
Project License (Name of license for the project) ['GPL']:
Zip-Safe? (Can this project be used as a zipped egg? (true/false)) [False]:
Zope2 Product? (Are you creating a product for Zope2) [True]:
Creating directory ./vs.policy
...
```

   Hiermit werden folgende Dateien erzeugt:

```
vs_buildout/src/vs.policy
├── CHANGES.rst
├── CONTRIBUTORS.rst
├── README.rst
├── bootstrap.py
├── buildout.cfg
├── docs
│   ├── LICENSE.GPL
│   └── LICENSE.txt
├── setup.cfg
├── setup.py
└── src
    └── vs
        ├── __init__.py
        └── policy
            ├── __init__.py
            ├── configure.zcml
            ├── profiles
            │   ├── default
            │   │   └── metadata.xml
            │   └── testing
            │       └── metadata.xml
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            ├── testing.py
            └── tests
                ├── __init__.py
                └── test_example.py
```

setup.py enthält Anweisungen für Setuptools, Distribute oder Buildout, wie die Paketdistribution verwaltet werden soll.

setup.cfg enthält zusätzliche Konfigurationsinformationen, in diesem Fall über das verwendete ZopeSkel-Template.

README.rst Dokumentation des Pakets.

Soll das Paket im PyPI veröffentlicht werden, wird der Inhalt zusammen mit dem Wert für `long_description` in der `setup.py`-Datei als HTML gerendert.

docs/ Enthält zusätzliche Dokumentation einschließlich der Software-Lizenz

CHANGES.rst wird verwendet für die *Change Log*-Einträge im PyPI.

src/vs das Namespace-Package, das in der `__init__.py`-Datei eine Methode für Setuptools und Distribute bereitstellt.

src/vs/policy Das Wurzelverzeichnis des Paketes selbst.

src/vs/policy/__init__.py Datei, die dieses Paket als Zope2-Produkt initiiert.

Ggf. wird diese Datei auch benötigt um Archetypes-Artikeltypen zu erstellen. Siehe hierzu Initialisierung und Hinzufügen-Rechte.

src/vs/policy/configure.zcml Die wesentliche Zope-Konfigurationsdatei für unser Paket. Diese wird automatisch von Plone beim Starten der Instanz geladen.

src/vs/policy/testing.py enthält ein Gerüst für Integrationstests. Wir werden dieses später durch unsere eigenen Tests ersetzen.

2. Nachdem das Python-Egg im `src`-Verzeichnis erstellt worden ist, sollten wir es noch in der `devel.cfg`-Datei eintragen. Der `instance-base`-Abschnitt übernimmt die Eggs:

```
[buildout]
...
develop =
    src/vs.policy
...
[instance-base]
eggs +=
    ...
    vs.policy
```

3. Für Eggs, die in Plone-Sites 3.3 verwendet werden sollen, kann z3c.autoinclude nicht verwendet werden. Dieses Paket erstellt automatisch zwei neue ZCML- Anweisungen: `includeDependencies` und `includePlugins`. Es ist mit dem ZopeSkel-Template bereits in der Datei `src/vs.policy/setup.py` verwendet worden:

```
setup(name='vs.theme',
    ...
    entry_points="""
    # -*- entry_points -*-
    [z3c.autoinclude.plugin]
    target = plone
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """,
    ...
    )
```

In Plone 3.3 muss hingegen noch folgendes in Die Buildout-Konfiguration eingetragen werden:

```
[instance]
...
zcml =
    vs.policy
```

Damit erstellt Buildout in der Instance einen sog. *zcml-slug* z.B. im Verzeichnis `parts/instance/etc/package-includes/` die Datei `001-vs.policy-configure.zcml` mit folgendem Einzeiler:

```
<include package="vs.policy" file="configure.zcml" />
```

Um zu Testen, ob das Python-Egg in der Instanz zur Verfügung steht, rufen wir den Python-Interpreter `zopepy` auf:

```
$ ./bin/zopepy
>>> from vs import policy
```

Da kein Fehler für den Import ausgegeben wurde, scheint das Egg geladen zu werden, und der Python-Interpreter kann mit `Strg-D` (unter Windows `Strg-Z`) wieder verlassen werden.

Alternativ kann auch die Instanz gestartet werden mit `./bin/instance`; anschließend sollten sich im Zope Management Interface → Control_Panel → Products das `vs.policy`-Produkt finden.

3.2.1 Ändern der Site-Konfiguration

In unserem Beispiel ändern wir Titel und Beschreibung der Site. Im ZMI könnten beide geändert werden unter `localhost:8080/mysite/manage_propertiesForm`. Für die programmatische Änderung solcher Einstellungen gibt es seit Plone 2.5 das Generic Setup Tool, das unter `localhost:8080/mysite/portal_setup/manage_workspace` verfügbar ist. Wenn Sie hier auf den *Export*-Reiter klicken, können solche Konfigurationen auch als XML-Dateien exportiert werden.

Erstellen eines `EXTENSION`-Profils

Um ein solches Profil zu erstellen wird `src/vs.policy/vs/policy/configure.zcml` folgendermaßen geändert:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five"
    xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
    i18n_domain="vs.policy">

<five:registerPackage package="." initialize=".initialize" />

<genericsetup:registerProfile
  name="default"
  title="vs.policy"
  directory="profiles/default"
  description="Policies for www.veit-schiele.de"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  provides="Products.GenericSetup.interfaces.EXTENSION"
  />

</configure>
```

Anschließend sind noch die angegebenen Verzeichnisse zu erstellen:

```
$ mkdir src/vs.policy/vs/policy/profiles src/vs.policy/vs/policy/profiles/default
```

Schließlich wird in `src/vs.policy/vs/policy/profiles/default` das Profil `properties.xml` erstellt mit:

```
<?xml version="1.0"?>
<site>
 <property name="title">Veit Schiele</property>
 <property name="description">Welcome to Veit Schiele</property>
</site>
```

3.2.2 Zusätzliche Änderungen der Site-Konfiguration

Leider lässt sich bisher nicht die gesamte Site durch Profile konfigurieren. Es lassen sich jedoch Methoden hinzufügen, die eine solche umfassende Konfiguration erlauben.

1. Hierzu wird zunächst in `vs.policy/vs/policy/configure.zcml` folgendes angegeben:

   ```
   <genericsetup:importStep
       name="vs.policy.various"
       title="vs.policy: miscellaneous import steps"
       description="Various import steps that are not handled by GS import/export␣
   ↪handlers."
       handler="vs.policy.setuphandlers.setupVarious">
   </genericsetup:importStep>
   ```

2. Anschließend wird die Datei `setuphandlers.py` angelegt in `vs.policy/vs/policy/`.

3. Diese Datei enthält zumindest die Methode `setupVarious`, die nur ausgeführt wird, sofern in im Kontext eine Datei `vs.policy_various.txt` vorhanden ist:

   ```
   def setupVarious(context):
       if context.readDataFile('vs.policy_various.txt') is None:
           return
   ```

4. Schließlich wird noch die Datei `vs.policy_various.txt` in `vs.policy/vs/policy/profiles/default` angelegt.

Konfigurieren der HTML-Filter

Die HTML-Filterregeln lassen sich aktuell nicht mit Generic Setup-Profilen konfigurieren. Soll dies mit der `setuphandlers.py`-Datei geschehen, kann diese z.B. so aussehen:

```
import logging
from plone.app.controlpanel.filter import IFilterSchema

logger = logging.getLogger('vs.policy')

def allowTags(site):
    """
    Allows embed, object, param and iframe tags
    """

    adapter = IFilterSchema(site)
    nasty_tags = adapter.nasty_tags
    if 'object' in nasty_tags:
        nasty_tags.remove('object')
    if 'embed' in nasty_tags:
        nasty_tags.remove('embed')

    stripped_tags = adapter.stripped_tags
    if 'object' in stripped_tags:
        stripped_tags.remove('object')
    if 'param' in stripped_tags:
        stripped_tags.remove('param')

    custom_tags = adapter.custom_tags
    if not 'embed' in custom_tags:
        custom_tags.append('embed')
    if not 'iframe' in custom_tags:
        custom_tags.append('iframe')

    adapter.nasty_tags = nasty_tags
    adapter.stripped_tags = stripped_tags
    adapter.custom_tags = custom_tags
    logger.info("Allowing embed, object, param and iframe tags.")

def setupVarious(context):

    if context.readDataFile('vs.policy_various.txt') is None:
        return

    site = context.getSite()
    allowIframeTags(site)
```

3.2.3 Tests schreiben

Tests erstellen

Statt der bereits angelegten Datei `src/vs.policy/vs/policy/tests.py` erstellen wir ein eigenes `tests`-Modul:

```
$ rm -rf src/vs.policy/vs/policy/tests.py
$ mkdir src/vs.policy/vs/policy/tests
$ touch src/vs.policy/vs/policy/tests/__init__.py
```

Test-Fixture

Anschließend definieren wir im neu erstellten `tests`-Ordner zunächst ein Test-Fixture, eine gleichbleibende Testumgebung mit der Basisklasse `TestCase`, die an den Layer `VS_POLICY_INTEGRATION` gebunden wird. Hierzu erstellen wir im `tests`-Verzeichnis die Datei `base.py` mit folgendem Inhalt:

```
import unittest2 as unittest

from plone.testing import z2
from plone.app.testing import TEST_USER_NAME
from plone.app.testing import TEST_USER_PASSWORD

from vs.policy.tests import layer

def get_browser(app, loggedIn=True):
    browser = z2.Browser(app)
    if loggedIn:
        auth = 'Basic %s:%s' % (TEST_USER_NAME, TEST_USER_PASSWORD)
        browser.addHeader('Authorization', auth)
    return browser

class TestCase(unittest.TestCase):
    layer = layer.VS_POLICY_INTEGRATION

class FunctionalTestCase(unittest.TestCase):
    layer = layer.VS_POLICY_FUNCTIONAL
```

In `layer.py` werden anschließend die Test-Layer `VS_POLICY_INTEGRATION` und `VS_POLICY_FUNCTIONAL` definiert, die beide auf `VS_POLICY_LAYER` basieren:

```
from plone.app.testing import applyProfile
from plone.app.testing import PloneFixture
from plone.app.testing import PloneSandboxLayer
from plone.app.testing import PloneTestLifecycle
from plone.app.testing import setRoles
from plone.app.testing import TEST_USER_ID
from plone.testing import z2
from zope.configuration import xmlconfig

class VsPolicyFixture(PloneFixture):
    # No sunburst please
    extensionProfiles = ()

VS_POLICY_FIXTURE = VsPolicyFixture()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
class VsPolicyTestLifecycle(PloneTestLifecycle):
    defaultBases = (VS_POLICY_FIXTURE, )

class IntegrationTesting(VsPolicyTestLifecycle, z2.IntegrationTesting):
    pass

class FunctionalTesting(VsPolicyTestLifecycle, z2.FunctionalTesting):
    pass

class VsPolicyLayer(PloneSandboxLayer):
    defaultBases = (VS_POLICY_FIXTURE, )

    def setUpZope(self, app, configurationContext):
        import vs.policy

        xmlconfig.file("configure.zcml", vs.policy,
                       context=configurationContext)
        z2.installProduct(app, 'vs.policy')

    def tearDownZope(self, app):
        z2.uninstallProduct(app, 'vs.policy')

    def setUpPloneSite(self, portal):
        applyProfile(portal, 'vs.policy:default')

        setRoles(portal, TEST_USER_ID, ['Manager'])
        portal.invokeFactory('Folder', 'test-folder')
        setRoles(portal, TEST_USER_ID, ['Member'])

VS_POLICY_LAYER = VsPolicyLayer()
VS_POLICY_INTEGRATION = IntegrationTesting(
    bases=(VS_POLICY_LAYER, ), name="VsPolicyLayer:Integration")
VS_POLICY_FUNCTIONAL = FunctionalTesting(
    bases=(VS_POLICY_LAYER, ), name="VsPolicyLayer:Functional")
```

Tests

Die eigentlichen Tests werden in der Datei `test_test.py` definiert:

```
from vs.policy.tests.base import FunctionalTestCase

class TestTest(FunctionalTestCase):

    def test_test(self):
        self.assertTrue(True)
```

Unit Tests, die auf dem Python unittest-Modul, ZopeTestCase und PloneTestCase basieren, müssen sich an einige Namenskonventionen halten:

- Alle Testdateien müssen mit `test` beginnen, z.B. `test_setup.py`.
- In den Testdateien werden Klassen für Testfälle definiert, die ein oder mehrere Testmethoden enthalten können, die ebenfalls mit `test` beginnen müssen, z.B. `test_portal_title`.
- Zunächst wird die Basisklasse importiert, dann die Klassen für die Testfälle und schließlich die Test Suite selbst

definiert.

- Jede Testsuite kann aus mehreren Testklassen bestehen. Wird die Testsuite ausgeführt, werden alle Testmethoden aller Testklassen der Test-Suite ausgeführt.
- Innerhalb einer Testklasse kann die `afterSetUp()`-Methode unmittelbar vor jedem Test aufgerufen werden um Testdaten für diesen Test anzugeben. Nachdem der Test durchgeführt wurde, werden die Transaktionen zurückgenommen, so dass normalerweise keine Artefakte zurückbleiben.
- Werden jedoch Änderungen außerhalb von Zope vorgenommen, müssen diese mit der Methode `beforeTearDown()` aufgeräumt werden.
- Die in einer Testklasse verwendeten Methoden wie `self.assertEqual()` oder `self.failUnless()` sind Assertion-Methoden, und wenn eine von ihnen fehlschlägt, gilt der ganze Test als fehlgeschlagen.

Test- und Hilfsmethoden

Testmethoden überprüfen, ob etwas wahr oder falsch ist. Daher kann aus den Tests auch herausgelesen werden, wie sich Ihr Produkt verhalten soll, welche Fähigkeiten es enthält. Die Liste der Testmethoden ist ausführlich in der Python-Dokumentation für unittest.TestCaseObjects enthalten. Die häufigsten sind:

`failUnless(expr)` stellt sicher, dass der Ausdruck `expr` wahr ist.

`assertEqual(expr1, expr2)` stellt sicher,dass `expr1` gleich `expr2` ist.

`assertRaises(exception, callable, ...)` stellt sicher, dass beim Aufruf von `callable` die Fehlermeldung `exception` ausgegeben wird.

Hinweis: `callable` sollte der Name einer Methode oder ein aufrufbares Objekt sein, nicht ein aktueller Aufruf, z.B.:

```
self.assertRaises(AttributeError, myObject.myMethod, someParameter)
```

`fail()` Dies ist sinnvoll, wenn ein Test noch nicht fertiggestellt ist oder in einem `if`-Statement, das deutlich macht, dass der Test fehlgeschlagen ist.

ZopeTestCase und PloneTestCase fügen zu den Assertion-Methoden noch weitere hilfreiche Methoden und Variablen hinzu, die mit Zope interagieren. Hier nur kurz die wesentlichen Variablen:

`self.portal` Die PloneSite, in der der Test ausgeführt wird.

`self.folder` Der `member`-Ordner des Mitglieds, als der die Tests ausgeführt werden.

Und hier die wesentlichen Hilfsmethoden:

`self.logout()` abmelden, d.i. die Rolle `anonymous` bekommen;

`self.login()` sich erneut anmelden; wird ein Nutzername mit übergeben, erfolgt die Anmeldung als dieser Nutzer.

`self.setRoles(roles)` durchläuft eine Liste von Rollen, die angenommen werden sollen.

`self.setRoles((Manager,))` lässt Sie beispielsweise die Rolle des Managers für eine bestimmte Zeit annehmen.

`self.setPermissions(permissions)` analog können auch Berechtigungen für den Testnutzer in `self.folder` angegeben werden;

`self.setGroups(groups)` eine Liste von Gruppen, der der aktuelle Nutzer angehören soll.

Mehr über Unit Tests in Python erfahren Sie in der unittest-Python-Dokumentation.

Testen

Der Testrunner kann nun gestartet werden mit:

```
$ ./bin/test -s vs.policy
```

Wären die Tests geschrieben worden, bevor die Profile erstellt wurden, hätten beide Tests fehlschlagen müssen und der Testrunner folgendes ausgeben:

```
 AssertionError:"Welcome to Veit Schiele != ''
...
AssertionError:'Veit Schiele != 'Plone site'
Ran 2 tests with 2 failures and 0 errors
```

Nachdem die Profile angelegt wurden, sollte jedoch keiner der Tests fehlschlagen:

```
Ran 2 tests with 0 failures and 0 errors.
```

Filter

-s my.package, --package my.package, --dir my.package durchsucht die angegebenen Verzeichnisse nach Tests.

-m test_setup, --module test_setup spezifiziert ein Testmodul als regulären Ausdruck, z.B.:

```
$ ./bin/test -s my.package  -m 'test_setup'
```

-t '.*installed.*', --test test_theme_installed spezifiziert einen Testfilter als regulären Ausdruck, z.B.:

```
$ ./bin/test -s vs.policy -m '.*setup.*' -t '.*installed.*'
```

Hiermit werden im Paket `vs.policy` alle, mit `installed` endenden, Methoden in allen Testmodulen, die auf `setup` enden, durchlaufen.

-u, --unit durchläuft ausschließlich Unit tests und ignoriert andere `layer`-Optionen.

-f, --non-unit durchläuft alle Tests, die keine Unit Tests sind

Report

-v, --verbose führt zu ausführlicherer Ausgabe

--ndiff falls ein Doctest fehlschlägt, wird `ndiff.py` zur Darstellung der Unterschiede verwendet

--udiff falls ein Doctest fehlschlägt, wird Unified Diff zur Darstellung der Unterschiede verwendet

--cdiff falls ein Doctest fehlschlägt, wird Context Diff zur Darstellung der Unterschiede verwendet

Analyse

-d, post-mortem stoppt die Ausführung nach dem ersten nicht-bestandenen Test und ermöglicht *post-mortem*-Debugging, d.h. die Debug-Session wird nur gestartet, wenn ein Test fehlschlägt.

Setup

--path src/my.package fügt einen Pfad zu Pythons Suchpfad hinzu, wobei die Option mehrfach angegeben werden kann.

Weitere Optionen

Diese erhalten Sie mit:

```
$ ./bin/test --help
```

Wenn die relevanten Tests erfolgreich verliefen, sollten schließlich noch alle Tests durchgeführt werden um sicherzustellen, dass nicht an anderer Stelle etwas gebrochen ist. Wenn alle Tests erfolgreich durchlaufen wurden, erscheint eine Meldung:

```
Ran 10 tests with 0 failures and 0 errors in 4.830 seconds.
```

Falls nicht alle Tests erfolgreich durchlaufen wurden, ändert sich die Meldung:

```
Ran 10 tests with 2 failures and 3 errors in 9.688 seconds.
```

Dabei wurden dann zwei Tests nicht bestanden und drei Tests enthielten Fehler.

roadrunner

roadrunner ist ein Testrunner für Plone 2.5 bis 3.1, der die testgetriebene Entwicklung deutlich beschleunigen kann, da er vorab das Standard-Zope- und Plone-Environment für PloneTestCase läd. zur Installation wird einfach folgendes in die `devel.cfg`-Datei eingetragen:

```
[buildout]
parts =
    ...
    roadrunner

[roadrunner]
recipe = roadrunner:plone
packages-under-test = vs.policy
```

Anschließend kann es wie der reguläre Zope-Testrunner aufgerufen werden:

```
$ ./bin/roadrunner -s vs.policy
```

Tipps & Tricks

- Übernehmen Sie Tests z.B. aus Plone wenn diese Ihren eigenen Absichten entsprechen.
- Dummy-Implementierungen sind häufig der einzige Weg um bestimmte Funktionen zu testen. Siehe auch CMFPlone/tests/dummy.py für einige Dummy-Objekt- Beispiele.
- Tests können auch verwendet werden um Dinge auszuprobieren – sie sind eine sichere Umgebung.
- Während des Debugging können `print`-Statements in den Test eingefügt werden um nachvollziehbare Hinweise im Terminal zu erhalten.
- Es kann jedoch auch gleich der Python-Debugger in die Testmethoden importiert werden mit:

```
import pdb; pdb.set_trace()
```

  Anschließend können Sie mit `r` schrittweise durch den Testkode gehen.

  Mehr zum Python-Debugger erfahren Sie in Debugging und in der Python-Dokumentation.

3.2.4 Installation

Zur Installation des Produkts in der Plone-Site gehen wir in Konfiguration → Zusatzprodukte, markieren vs.policy und klicken auf *Installieren*.

Anschließend sollten Sie in Ihrem Webbrowser unter der URL `localhost:8080/mysite/folder_contents` Titel und Beschreibung sehen können.

3.3 Zusatzprodukte

Finden, evaluieren und installieren von Zusatzprodukten.

3.3.1 Finden

Die meisten Produkte befinden sich mittlerweile auf Github: https://github.com/collective.

3.3.2 Evaluieren

Um nun zu überprüfen, ob das Produkt wirklich passend ist, können Sie verschiedene Schritte durchführen:

1. Zunächst sollten Sie überprüfen, wie sich das Produkt selbst präsentiert:
   - Ist es hinreichend gut dokumentiert?
   - Gibt es einen Bug Tracker?

     Wie viele offene und geschlossene Bugs gibt es?

     Wie schnell wurden die Bugs behoben?
   - In welcher Version liegt das Produkt vor?

     Eine 0.* oder alpha-Version ist voraussichtlich weniger stabil als ein *final release*.

- Gibt es eine Roadmap?

  Sie gibt Ihnen Hinweise, wie die Planung für die weitere Entwicklung aussieht und wie zukunftssicher das Produkt ist.

2. Wie umfangreich und mit welchen Erfahrungen wird das Produkt eingesetzt?

   Fragen Sie in einer Mailingliste nach, welche Erfahrungen mit dem Produkt gemacht wurden.

3. Dann sollten Sie das Produkt ausführlich in einer Testumgebung testen.

   - Kopieren Sie gegebenenfalls die ZODB aus Ihrem Produktivsystem in Ihr Testsystem.

     Achten Sie darauf, dass die Buildout-Konfigurationen beider Systeme identisch sind.

   - Wie hoch ist die Testabdeckung des Produkts?

     In Testabdeckung (Code Coverage) erhalten Sie weitergehende Informationen.

   - Durchlaufen Sie die automatisierten Tests sowohl Ihrer eigenen als auch des neuen Produkts.

   - Schließlich sollten Sie die Funktionalität und das User Interface auch in Ihrem Browser testen.

Allgemeinere und umfassendere Informationen zur Evaluation von OpenSource- Software erhalten Sie im Artikel Software-Evaluation.

3.3.3 Installation und Aktivierung

1. Zusatzprodukte können unter `install_requires` in der setup.py``-Datei angegeben werden, also z.B.:

```
install_requires=[
    'setuptools',
    'Plone',
    'vs.event',
],
```

2. Anschließend wird das Zusatzprodukt noch in der Instanz registriert in der `configure.zcml`-Datei:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five"
    xmlns:i18n="http://namespaces.zope.org/i18n"
    xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
    i18n_domain="vs.policy">
    <includeDependencies package="." />
    ...
</configure>
```

   `<includeDependencies package="." />` fügt alle Pakete der `install_requires`-Liste einer Site hinzu.

   Sollen die Pakete explizit angegeben werden, kann dies z.B. erfolgen mit:

```
<include package="vs.event" />
```

3. Damit unser Zusatzprodukt beim Aufsetzen einer neuen Plone-Site mit unserem `vs.policy`-Produkt automatisch aktiviert wird, sollten wir es noch in der `metadata.xml`-Datei eintragen:

```
<?xml version="1.0"?>
<metadata>
  <version>1.0</version>
  <dependencies>
    <dependency>profile-vs.event:default</dependency>
  </dependencies>
</metadata>
```

4. Schließlich sollten diese Änderungen noch in unserem Buildout-Projekt übernommen werden. Hierzu wird das Buildout-Skript erneut aufgerufen:

```
$ bin/buildout
```

3.3.4 Tests

1. Zunächst müssen wir sicherstellen, dass ``vs.event´´ in unseren Tests zur Verfügung steht:

```
class VsPolicy(PloneSandboxLayer):
    defaultBases = (PLONE_FIXTURE,)
    def setUpZope(self, app, configurationContext):
        # Load ZCML
        import vs.policy
        xmlconfig.file(
            'configure.zcml',
            vs.policy,
            context=configurationContext
        )
        # Install products that use an old-style initialize()
        # function
        z2.installProduct(app, 'vs.event')

def tearDownZope(self, app):
    # Uninstall products installed above
    z2.uninstallProduct(app, 'vs.event')

def setUpPloneSite(self, portal):
    applyProfile(portal, 'vs.policy:default')
```

2. Nun fügen wir den eigentlichen Test hinzu:

```
def test_vs_event_installed(self):
    portal = self.layer['portal']
    portal_types = getToolByName(portal, 'portal_types')
    self.assertTrue("VSEvent" in portal_types)
```

3. Schließlich führen wir diesen Test aus:

```
$ ./bin/test
```

Häufig verwendete Zusatzprodukte

Plone 5

Zusatzprodukte für Plone 5 finden Sie im Python Package Index (PyPI): Framework: Plone: 5.0.

Plone 4

Kommunikation

PloneFormGen erlaubt das web-basierte Erstellen von Formularen.

> Mit uwosh.pfg.d2c gibt es auch eine Erweiterung für PloneFormGen, mit der sich aus PloneFormGen-Formularen Artikel erstellen lassen.

PloneBoard ist ein einfach zu verwendendes Web-Forum, das gut in die Plone-Oberfläche integriert ist.

collective.blog.star Blogging-Tool für Plone.

vs.jquerybookmarks integriert das jQuery-Plugin bookmark.

EasyNewsletter erlaubt das einfache Erstellen von Newslettern aus Inhalten einer Plone- Site.

Spam-Schutz

plone.formwidget.captcha für Captcha Spam-Schutz

plone.formwidget.recaptcha für ReCaptcha-Spam-Schutz

collective.akismet für Akismet-Spam-Schutz

collective.z3cform.norobots bietet ein Widget basierend auf einer Liste von Fragen und Antworten

Kalender und Termine

plone.app.event ist ein Termin-Artikeltyp mit Unterstützung für sich wiederholende Termine und Zeitzonen, RFC5545-Kalender-Export etc.

Solgema.fullcalendar ändert einen Ordner in einen Kalender mit verschiedenen Ansichten: Monat, Woche, Tag.

Layout

collective.cover erlaubt das einfache Ändern von Layouts einer Plone-Site.

collective.plonetruegallery Gallerie und Slideshow, die auch Bilder aus Picase und Flickr anzeigen können.

collective.easyslider erlaubt das Hinzufügen eines Sliders zu jeder Plone-Seite. Darüberhinaus gibt es auch eine Slider-Ansicht für Ordner und Kollektionen.

Erschließen von Inhalten

Document Viewer OCR und Ansicht von PDF-, Word- und Excel-Dokumenten

docsplit kann ggf. im Userspace installiert werden mit:

```
$ gem install docsplit --user-install
```

Anschließend kann der Pfad in die `~.bashrc` eingetragen werden, z.B.:

```
export PATH=$HOME/.gem/ruby/1.9.1/bin:$PATH
```

und anschließend das Terminal aktualisert werden mit:

```
$ source ~/.bashrc
```

Plone Glossary erlaubt das Erstellen mehrerer Glossare in einer Plone-Site.

AT Vocabulary Manager ermöglicht das Erstellen mehrerer Vokabularien in Plone, die entweder flach, hierarchisch oder VDEX-basiert sein können.

plone.app.multilingual erlaubt mehrsprachige Plone-Sites.

Datei-Handling

wildcard.foldercontents bietet eine bessere Ansicht von Ordnerinhalten, das sowohl die Sortierung erleichtert als auch das Hochladen mehrerer Dateien.

collective.zipfiletransport ermöglicht das Hoch- und Herunterladen von Zip-Dateien.

Reflecto stellt Teile des Dateisystems in einer Plone-Site dar.

Rechteverwaltung

FacultyStaffDirectory bietet persönliche Verzeichnisse, Arbeitsräume etc. wobei es einerseits gut in Plones Nutzer- und Gruppenverwaltung integriert ist und andererseits leicht für spezifische Anforderungen erweitert werden kann.

Artikeltypen

Smart Link ist ein Link-Artikeltyp mit Bild und veränderbarem Icon.

PressRoom erstellt einen Presseraum mit Presseveröffentlichungen, Mitteilungen und Pressekontakten.

Plone Help Center wurde entwickelt für die Dokumentation von Plone, kann jedoch auch gut für die Dokumentation anderer Open-Source-Produkte verwendet werden.

LinguaPlone

Um LinguaPlone ordnungsgemäß in einer Plone-Site aktivieren zu können, muss folgende Reihenfolge eingehalten werden:

1. Zunächst muss die Sprachauswahl im *Plone Language Tool* angegeben werden.
2. Erst im Anschluss daran kann LinguaPlone installiert werden.
3. Und schließlich wird der `@@language-setup-folders`-View von LinguaPlone aufgerufen um auch das Site-Root-Objekt mehrsprachig darzustellen.

Programmatisch lässt sich dies realisieren, indem in `vs.policy/vs/policy/setuphandlers.py` zunächst die Spracheinstellungen gesetzt werden, anschließend LinguaPlone aktiviert und der View `@@language-setup-folders` aufgerufen wird:

```
def installLinguaPlone(site):

     # LP must be installed a last step in order to deal with
     # several strange annoyances and expecations that LP relies on.

    pl = getToolByName(site, 'portal_languages')
    pl.supported_langs = ('de', 'en')

    qi = getToolByName(site, 'portal_quickinstaller')
    qi.installProducts(['Products.LinguaPlone'])

    transaction.savepoint(1)#
    site.restrictedTraverse('@@language-setup-folders')()

...

def setupVarious(context):
    if context.readDataFile('ise.policy_various.txt') is None:
        return
    ...
    installLinguaPlone(site)
```

Tests

In `test_language_settings.py` können folgende Tests geschrieben werden:

```
import unittest2
from base import TestBase

class LanguageTests(TestBase):
    def testInstalledProducts(self):
        ...
        self.assertEqual('LinguaPlone' in installed, True)

    def testLanguageSettings(self):
        lang_tool = self.portal.portal_languages
        default_language = self.portal.portal_languages.getDefaultLanguage()
        self.assertEqual(default_language == 'de', True)
        # return [(country code, countryname), ...]
        supported_languages = [r[0] for r in self.portal.portal_languages.
↪listSupportedLanguages()]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        self.assertEqual('en' in supported_languages, True)
        self.assertEqual('de' in supported_languages, True)
        self.assertEqual(lang_tool.use_cookie_negotiation, True)
        self.assertEqual(lang_tool.use_request_negotiation, True)
        self.assertEqual(lang_tool.use_content_negotiation, True)

def test_suite():
    from unittest2 import TestSuite, makeSuite
    suite = TestSuite()
    suite.addTest(makeSuite(LanguageTests))
    return suite
```

TinyMCE-Erweiterungen

Sollen sprachneutrale Inhalte wie z.B. Bilder erstellt werden, sollte TinyMCE so gepatcht werden, dass er über die sprachspezifischen Ordner hinaus referenzieren kann. Dies lässt sich am einfachsten realisieren mit collective.monkeypatcher.

Anschließend erweitern wir unsere `configure.zcml`-Datei um `patches.zcml`:

```
<include file="patches.zcml" />
```

Nun legen wir `patches.zcml` mit folgendem Inhalt an:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:monkey="http://namespaces.plone.org/monkey"
    i18n_domain="vs.policy">

    <include package="collective.monkeypatcher" />

    <monkey:patch
        description="TinyMCE JSON Folder listing should ignore INavigationRoot"
        class="Products.TinyMCE.adapters.JSONFolderListing.JSONFolderListing"
        original="getListing"
        replacement=".patches.getListing"
        />

    <monkey:patch
        description="Navigation support RefBrowserWidget across INavigationRoot"
        class="archetypes.referencebrowserwidget.browser.view.ReferenceBrowserPopup"
        original="breadcrumbs"
        replacement=".patches.breadcrumbs"
        />

   <monkey:patch
        description="Unrestrict TinyMCE image search"
        class="Products.TinyMCE.adapters.JSONSearch.JSONSearch"
        original="getSearchResults"
        replacement=".patches.getSearchResults"
        />
</configure>
```

Schließlich schreiben wir noch die Datei `patches.py`, die die Originalklassen mit den entsprechenden Änderungen enthält.

3.4 Erscheinungsbild

3.4.1 Plone3-Theme-Package

Erstellen des Eggs

```
$ cd src
$ paster create -t plone3_theme

Enter namespace_package (Namespace package (like plonetheme)) ['plonetheme']: vs
Enter package (The package contained namespace package (like example)) ['example']:␣
↪theme
Enter skinname (The skin selection to be added to 'portal_skins' (like 'My Theme')) ['
↪']: vs.theme
Enter skinbase (Name of the skin selection from which the new one will be copied) [
↪'Plone Default']:
Enter empty_styles (Override default public stylesheets with empty ones?) [True]:␣
↪False
Enter include_doc (Include in-line documentation in generated code?) [False]:
Enter zope2product (Are you creating a Zope 2 Product?) [True]:
...
Enter zip_safe (True/False: if the package can be distributed as a .zip file) [False]:
```

Enter empty_styles Wenn die Skin-Anpassungen sehr umfangreich sind oder es sich um eine performance-kritische Anwendung handelt empfehle ich, den Skin vollständig neu aufzusetzen und Enter `empty_styles` mit `True` anzugeben.

Enter skinbase In Plone 4 kann hier zwischen folgenden beiden Skins gewählt werden:

Sunburst Theme Ein neuer Skin aus dem plonetheme.sunburst-Egg.

Sunburst ist der Standard-Skin für neu erstellte Plone-4-Sites.

Plone Classic Theme Der aus Plone 3 bekannte Plone Default-Skin. Er ist nun im `plonetheme.classic`-Egg zu finden.

Plone Default Der Plone Default-Skin ist in Plone 4 nur noch ein minimalistischer Skin, der ideal geeignet ist für die nachgelagerte Gestaltung einer Plone-Site mit XDV oder Deliverance.

In Plone 3 besteht üblicherweise die Wahl zwischen zwei Skins:

Plone Default Der Standard-Skin.

NuPlone Ein modernerer Skin für Plone 3.

Verzeichnisübersicht des `vs.theme`-Produkts

Skin

Eigenen Skin erstellen

Um nun das Plone Skin Tools für unsere Bedürfnisse anzupassen, wurde die Datei `src/vs.theme/vs/theme/profiles/default/skins.xml` folgendermaßen erstellt:

```
<?xml version="1.0"?>
<object name="portal_skins" allow_any="False" cookie_persistence="False"
        default_skin="vs.theme">
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <object name="vs_theme_custom_images"
            meta_type="Filesystem Directory View"
            directory="vs.theme:skins/vs_theme_custom_images"/>
    <object name="vs_theme_custom_templates"
            meta_type="Filesystem Directory View"
            directory="vs.theme:skins/vs_theme_custom_templates"/>
    <object name="vs_theme_styles"
            meta_type="Filesystem Directory View"
            directory="vs.theme:skins/vs_theme_styles"/>

    <skin-path name="vs.theme" based-on="Plone Default">
        <layer name="vs_theme_custom_images"
               insert-after="custom"/>
        <layer name="vs_theme_custom_templates"
               insert-after="vs_theme_custom_images"/>
        <layer name="vs_theme_styles"
               insert-after="vs_theme_custom_templates"/>
    </skin-path>

</object>
```

Damit werden die drei Verzeichnisse `vs_theme_custom_images`, `vs_theme_custom_templates` und `vs_theme_styles` registriert und ein neuer Skin *vstheme*, der auf *Plone Default* basiert und zudem die drei oben genannten Layer enthält, als Standard-Skin angegeben.

Bemerkung: Sollen die Layer allen Skins zugewiesen werden, kann dies einfach so angegeben werden:

```
<skin-path name="*">
    ...
</skin-path>
```

Entfernen von Layern

Layer können auch einfach wieder entfernt werden mit:

```
<object name="vs_theme_custom_templates" remove="True "/>
```

Dies kann z.B. für ein `uninstall`-Profil verwendet werden.

Elemente in einem Skin-Layer überschreiben

Wollen Sie z.B. das Logo durch ein eigenes im gif-Format ersetzen, sollten Sie zunächst in der Datei `src/vs.theme/vs/theme/skins/vs_theme_styles/base_properties.props` die Angabe für `logoName` ändern:

```
logoName:string=logo.gif
```

Anschließend können Sie Ihr Logo in `src/vs.theme/vs/theme/skins/vs_theme_custom_images/` einfügen.

Bemerkung: Um die DTML-Variable `fontFamily` in einer CSS-Datei verwenden zu können, darf sie nicht mit `&dtml-fontFamily;` eingebunden werden sondern mit `<dtml-var fontFamily>;`, da ansonsten das Zeichen " als `&quot;` interpretiert würde.

Bemerkung: Sollen PageTemplates überschrieben werden, die auch Meta-Angaben in einer `.metadata`-Datei enthalten, dann sollte auch diese Datei mitkopiert werden.

Resourcen registrieren

Um die CSS-Datei `src/vs.theme/vs/theme/browser/stylesheets/main.css` am *CSS Registry Tool* zu registrieren, wird die Datei `src/vs.theme/vs/theme/profiles/default/cssregistry.xml` folgendermaßen geändert:

```
<?xml version="1.0"?>
<object name="portal_css">

 <stylesheet title=""
    id="++resource++vs.theme.stylesheets/main.css"
    media="screen"
    rel="stylesheet"
    rendering="import"
    cacheable="True"
    compression="safe"
    cookable="True"
    enabled="1"
    expression=""/>

</object>
```

id ergibt sich daraus, dass das Verzeichnis `src/vs.theme/vs/theme/browser/stylesheets`, das die `main.css`-Datei enthält, als `resourceDirectory` in `src/vs.theme/vs/theme/browser/configure.zcml` registriert ist.

Title Durch die Angabe eines Titels zusammen mit `rel=stylesheet` wird ein Stylesheet-Dokument vor anderen bevorzugt.

expression Die Bedingung, unter der das Stylesheet ausgeführt werden soll, als TALES-Ausdruck.

Im Folgenden einige der häufigsten Bedingungen:

- nach Artikeltyp:

```
expression = "python:object.meta_type == 'ATFolder'"
```

- nach View:

```
expression="object/@@registration_view/enabled | nothing"
```

  - nach globalen Views:

```
expression="python:portal.restrictedTraverse ('@@plone_portal_state').is_
↪rtl()"
```

- nach Rollen:

```
expression="not: portal/portal_membership/isAnonymousUser"
```

oder:

```
expression="python: not here.restrictedTraverse
                        ('@@plone_portal_state').anonymous()"
```

media Das Medium, für das das Stylesheet gilt:

`all`, `aural`, `braille`, `embossed`, `handheld`, `print`, `projection`, `screen`, `tty` und `tv`.

rel mögliche Werte sind `stylesheet` und `alternate stylesheet`. Der Standardwert ist `stylesheet`.

rendering Angabe, wie das Stylesheet in die HTML-Seiten eingebunden werden soll. Mögliche Werte sind `import`, `link` und `inline`.

compression Angabe, ob und wie das Stylesheet komprimiert werden darf. Mögliche Werte sind `none`, `safe` oder `full`.

enabled Angabe, ob das Stylesheet aktiv ist.

cookable Angabe, ob das Zusammenfügen mit anderen Stylesheets erlaubt wird.

cacheable Angabe, ob das Caching des Stylesheets erlaubt wird.

conditionalcomment Ab der Plone-Version 3.3 kann mit der *CSS-Registry* das Einbinden der CSS-Datei auch in sog. *Conditional Comments* erfolgen, also z.B.:

```
<!--[if IE]>
<style type="text/css" media="all">@import url(http://localhost:8080/mysite/
↪portal_css/vs.theme/iefixes-cachekey7904.css);</style>
<![endif]-->
```

Hierzu wird in der `cssregistry.xml`-Datei folgendes angegeben:

```
<stylesheet title=""
            ...
            conditionalcomment="IE"
            id="iefixes.css"/>
```

Weitere Hinweise zu *Conditional Comments* erhalten Sie in About Conditional Comments.

Bemerkung: Zum Entwickeln von Stylesheets empfiehlt sich, entweder die Instanz im Debug-Modus zu starten oder im ZMI der CSS Registry *Debug/development mode* zu aktivieren und damit das Caching von CSS-Dateien im Browser zu verhindern.

Entfernen von css-Dateien

CSS-Dateien können entfernt werden mit:

```
<stylesheet id="++resource++vs.theme.stylesheets/main.css" remove="True"/>
```

Dies kann z.B. für ein `uninstall`-Profil verwendet werden.

Browser Views

Browser View überschreiben

Zope 3 Browser Views werden üblicherweise als Page Templates definiert. Meist liegen diese Page Templates in einem Unterpaket namens `browser`. Um ein solches Page Template ändern zu können, wird zunächst in `browser/configure.zcml` der Skin an unser *vs.theme* gebunden:

```
<interface
   interface=".interfaces.IThemeSpecific"
   type="zope.publisher.interfaces.browser.IBrowserSkinType"
   name="vs.theme"
   />
```

Um herauszufinden, wo sich das gesuchte Page Template im Dateisystem befindet, kann im ZMI *Plone View Customizations* verwendet werden. Sobald sich der Cursor über dem Namen des entsprechenden Templates befindet, wird der Pfad im Dateisystem angezeigt. Alternativ kann auch nach `.zcml`-Dateien gesucht werden, die den Namen der Ansicht enthalten.

Suchen wir z.B. nach *Dashboard*, so finden wir zwei Ergebnisse:

`plone.app.layout.dashboard`

```
<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
    name="dashboard"
    permission="plone.app.portlets.ManageOwnPortlets"
    class=".dashboard.DashboardView"
    template="dashboard.pt"
    />
```

`plone.app.portlets.browser`

```
<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
    class=".manage.ManageDashboardPortlets"
    name="manage-dashboard"
    template="templates/manage-dashboard.pt"
    permission="plone.app.portlets.ManageOwnPortlets"
    />
```

Um diese beiden Views zu ändern, tragen wir folgendes in `vs.theme/vs/theme/browser/configure.zcml` ein:

```
<include package="plone.app.portlets" />

<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
    name="dashboard"
    permission="plone.app.portlets.ManageOwnPortlets"
    class="plone.app.layout.dashboard.dashboard.DashboardView"
    template="templates/dashboard.pt"
    layer=".interfaces.IThemeSpecific"
    />

<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    name="manage-dashboard"
    permission="plone.app.portlets.ManageOwnPortlets"
    class="plone.app.portlets.browser.manage.ManageDashboardPortlets"
    template="templates/manage-dashboard.pt"
    layer=".interfaces.IThemeSpecific"
    />
```

Um die Templates überschreiben zu können, ist die Angabe für `layer` wesentlich.

Anschließend wird in `browser` der Ordner `templates` erstellt und dann die beiden Templates `dashboard.pt` und `manage-dashboard.pt` dahin kopiert und geändert.

Plone 3.x

Sollen z.B. die Portlets der linken und rechten Spalte angezeigt werden, müssen nur die entsprechenden Zeilen auskommentiert werden, also:

```
<!-- <metal:left fill-slot="column_one_slot" /> -->
<!-- <metal:right fill-slot="column_two_slot" /> -->
```

Plone 4

Ab Plone 4.0 lassen sich die Portlet-Manager besser kontrollieren. So finden Sie nun z.B. im Page Template `dashboard.pt` folgende Anweisung:

```
<head>
    <metal:block fill-slot="top_slot"
                 tal:define="dummy python:request.set('disable_border',1);
                             disable_column_one python:request.set('disable_plone.
↪leftcolumn',1);
                             disable_column_two python:request.set('disable_plone.
↪rightcolumn',1);" />
</head>
```

- Dies sorgt dafür, dass die linke und rechte Spalte üblicherweise nicht im Dashboard angezeigt werden.
- Setzen wir nun die Anweisung auf `0`, so werden der linke und rechte Portlet-Manager wieder angezeigt.
- Werden die Anweisungen im Template weggelassen, so greift die übliche Logik, bei der unterschieden wird, ob Portlets zugewiesen sind oder nicht.

Bemerkung: Eine Anleitung zu Page Templates erhalten Sie in Zope Page Templates (ZPT)

Globale Variablen und Hilfsansichten

Globale Variablen

In Plone 3.x mussten einige Variablen nicht explizit in einem Template definiert werden, da sie bereits als `global_defines` in das `main_template` eingefügt wurden. Ab Plone 4 müssen auch diese Variablen wieder explizit definiert werden. Hier ein Überblick über die gebräuchlichsten dieser globalen Variablen:

`portal` Das Plone-Site-Root-Objekt

`portal_url` Die URL des Plone-*site root*-Objekts

`member` Der aktuell angemeldete Nutzer

`checkPermission` Funktion, die überprüft ob der aktuell angemeldete Nutzer im aktuellen Kontext eine bestimmte Berechtigung hat. Hier ein Beispiel aus `parts/plone/CMFPlone/skins/plone_forms/folder_rename_form.cpt`:

```
tal:define="canModifyItem python:checkPermission('Modify portal content', obj);"
```

`isAnon` `True` wenn der aktuelle Nutzer nicht angemeldet ist

`is_editable` `True` wenn der aktuelle Nutzer den aktuellen Kontext editieren darf

`isLocked` `True` wenn das aktuelle Objekt für das Editieren gesperrt ist

Einen vollständigen Überblick über die verfügbaren Variablen erhalten Sie im Docstring der `globalize()`-Methode in `Products.CMFPlone.browser.interfaces.IPlone`.

Hilfsansichten

In `plone.app.layout.globals` sind Hilfsansichten definiert, mit denen Sie sich häufig genutzte Informationen anzeigen lassen können:

`@@plone_tools` Zugang zu den gebräuchlichsten *CMF Tools*

`@@plone_context_state` Informationen des aktuellen Kontexts, wie URL, Pfad, Status und Editierbarkeit

`@@plone_portal_state` Informationen über die aktuelle Plone-Site, wie URL der Site, aktueller Nutzer und ob er anonym ist

Ein Vorteil dieser Hilfsansichten ist, dass ihre Methoden gecached werden, sodass sie nur bei der ersten Anfrage berechnet werden müssen.

Wie diese Hilfsansichten in Viewlets verwendet werden, können Sie z.B. in `plone/app/layout/viewlets/content.py` sehen:

```
self.portal_state = getMultiAdapter((self.context, self.request),
                                    name=u'plone_portal_state')
self.portal_url = self.portal_state.portal_url()
```

Als Beispiel, wie diese Hilfsansichten in Page Templates verwendet werden können, hier ein Auszug aus `plone/app/portlets/portlets/login.pt`:

```
tal:attributes="value context/@@plone_context_state/current_page_url"
```

Eine vollständige Übersicht über die verfügbaren Interfaces finden Sie in `plone.app.layout.globals.interfaces`.

Viewlets

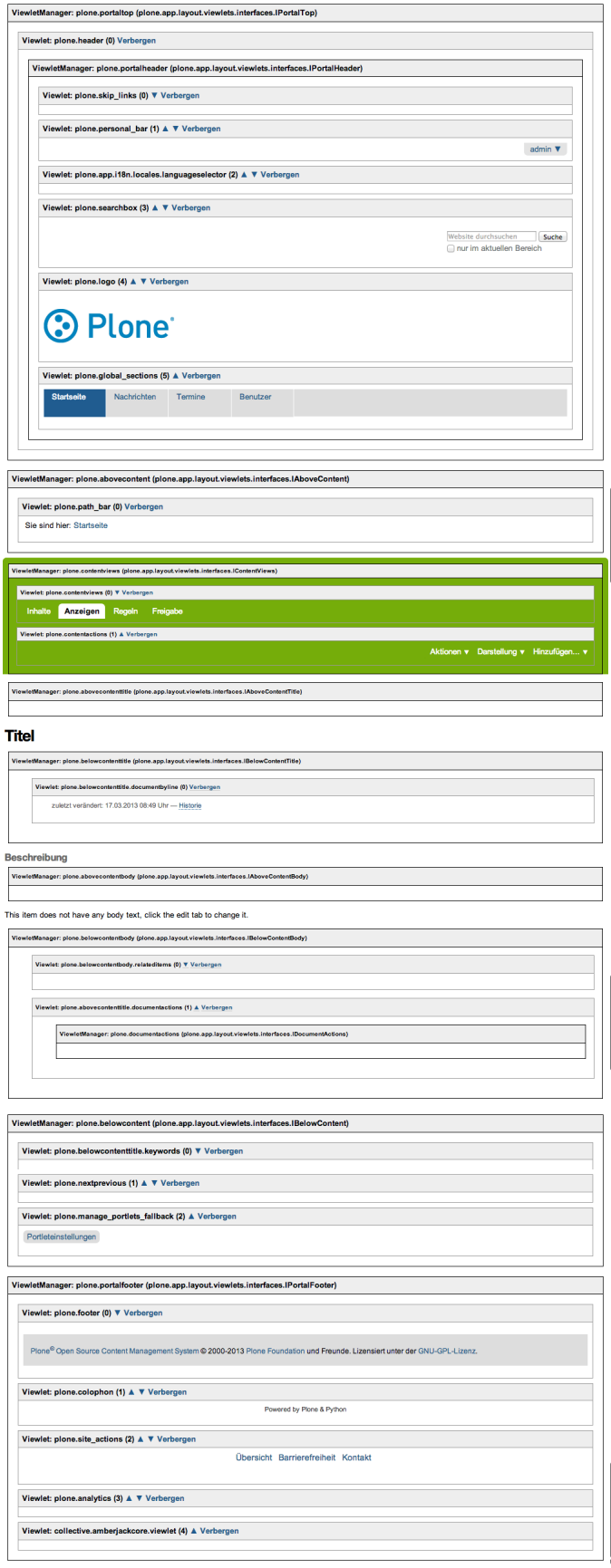
Viewlets und Viewlet-Manager konfigurieren

Um sich einen Überblick über die verschiedenen Viewlet-Manager und Viewlets zu verschaffen, empfiehlt sich im Webbrowser der Aufruf der URL `http://localhost:8080/mysite/@@manage-viewlets`.

Wie die Anzeige und Reihenfolge der Viewlets programmatisch verändert werden kann, können Sie sich in `src/vs.theme/vs/theme/profiles/default/viewlets.xml` anschauen:

```
<?xml version="1.0"?>
<object>
    <order manager="plone.portaltop" skinname="vs.theme" based-on="Plone Default">
        <viewlet name="plone.header" />
    </order>

    <hidden manager="plone.portaltop" skinname="vs.theme">
        <viewlet name="plone.app.i18n.locales.languageselector" />
        <viewlet name="plone.path_bar" />
        <viewlet name="plone.personal_bar" />
    </hidden>

    <order manager="plone.portalheader" skinname="vs.theme" based-on="Plone Default">
        <viewlet name="plone.skip_links" />
        <viewlet name="plone.site_actions" />
        <viewlet name="plone.logo" />
        <viewlet name="plone.global_sections" />
    </order>

    <hidden manager="plone.portalheader" skinname="vs.theme">
        <viewlet name="plone.searchbox" />
    </hidden>

    <order manager="plone.contentviews" skinname="vs.theme" based-on="Plone Default">
        <viewlet name="vs.path_bar" />
        <viewlet name="vs.personal_bar" />
        <viewlet name="plone.contentviews" />
        <viewlet name="plone.contentactions" />
    </order>

    <hidden manager="plone.belowcontenttitle" skinname="vs.theme">
        <viewlet name="plone.belowcontenttitle.documentbyline" />
    </hidden>

    <hidden manager="plone.abovecontentbody" skinname="vs.theme">
        <viewlet name="plone.presentation" />
    </hidden>

    <order manager="plone.portalfooter" skinname="vs.theme" based-on="Plone Default">
        <viewlet name="vs.footer" />
        <viewlet name="plone.colophon" />
    </order>

    <hidden manager="plone.portalfooter" skinname="vs.theme">
        <viewlet name="plone.footer" />
    </hidden>
```

(Fortsetzung auf der nächsten Seite)


ViewletManager: plone.portaltop (plone.app.layout.viewlets.interfaces.IPortalTop)
Viewlet: plone.header (0) Verbergen
ViewletManager: plone.portalheader (plone.app.layout.viewlets.interfaces.IPortalHeader)
Viewlet: plone.skip_links (0) ▼ Verbergen
Viewlet: plone.personal_bar (1) ▲ ▼ Verbergen
admin ▼
Viewlet: plone.app.i18n.locales.languageselector (2) ▲ ▼ Verbergen
Viewlet: plone.searchbox (3) ▲ ▼ Verbergen
Website durchsuchen
Suche
nur im aktuellen Bereich
Viewlet: plone.logo (4) ▲ ▼ Verbergen
Plone
Viewlet: plone.global_sections (5) ▲ Verbergen
Startseite
Nachrichten
Termine
Benutzer
ViewletManager: plone.abovecontent (plone.app.layout.viewlets.interfaces.IAboveContent)
Viewlet: plone.path_bar (0) Verbergen
Sie sind hier: Startseite
ViewletManager: plone.contentviews (plone.app.layout.viewlets.interfaces.IContentViews)
Viewlet: plone.contentviews (0) ▼ Verbergen
Inhalte
Anzeigen
Regeln
Freigabe
Viewlet: plone.contentactions (1) ▲ Verbergen
Aktionen ▼
Darstellung ▼
Hinzufügen... ▼
ViewletManager: plone.abovecontenttitle (plone.app.layout.viewlets.interfaces.IAboveContentTitle)
Titel
ViewletManager: plone.belowcontenttitle (plone.app.layout.viewlets.interfaces.IBelowContentTitle)
Viewlet: plone.belowcontenttitle.documentbyline (0) Verbergen
zuletzt verändert: 17.03.2013 08:49 Uhr — Historie
Beschreibung
ViewletManager: plone.abovecontentbody (plone.app.layout.viewlets.interfaces.IAboveContentBody)
This item does not have any body text, click the edit tab to change it.
ViewletManager: plone.belowcontentbody (plone.app.layout.viewlets.interfaces.IBelowContentBody)
Viewlet: plone.belowcontentbody.relateditems (0) ▼ Verbergen
Viewlet: plone.abovecontenttitle.documentactions (1) ▲ Verbergen
ViewletManager: plone.documentactions (plone.app.layout.viewlets.interfaces.IDocumentActions)
ViewletManager: plone.belowcontent (plone.app.layout.viewlets.interfaces.IBelowContent)
Viewlet: plone.belowcontenttitle.keywords (0) ▼ Verbergen
Viewlet: plone.nextprevious (1) ▲ ▼ Verbergen
Viewlet: plone.manage_portlets_fallback (2) ▲ Verbergen
Portleteinstellungen
ViewletManager: plone.portalfooter (plone.app.layout.viewlets.interfaces.IPortalFooter)
Viewlet: plone.footer (0) ▼ Verbergen
Plone® Open Source Content Management System © 2000-2013 Plone Foundation und Freunde. Lizensiert unter der GNU-GPL-Lizenz.
Viewlet: plone.colophon (1) ▲ ▼ Verbergen
Powered by Plone & Python
Viewlet: plone.site_actions (2) ▲ ▼ Verbergen
Übersicht
Barrierefreiheit
Kontakt
Viewlet: plone.analytics (3) ▲ ▼ Verbergen
Viewlet: collective.amberjackcore.viewlet (4) ▲ Verbergen

(Fortsetzung der vorherigen Seite)

```
</object>
```

<order ...> gibt die Reihenfolge der Viewlets an:

skinname beschränkt die angegebene Reihenfolge auf einen spezifischen Skin.

Soll der entsprechende Manager für alle Skins verwendet werden, kann auch folgendes angegeben werden:

```
skinname="*"
```

based on gibt an, auf welchem Skin die Reihenfolge basiert.

Für die einzelnen Viewlets stehen auch noch die folgenden drei Angaben zur Verfügung:

<viewlet name="my.viewlet" insert-before="another.viewlet" /> Das Viewlet wird unmittelbar vor einem spezifischen Viewlet eingefügt.

<viewlet name="my.viewlet" insert-before="*" /> Das Viewlet wird vor allen anderen Viewlets eingefügt.

<viewlet name="my.viewlet" insert-after="another.viewlet" /> Das Viewlet wird unmittelbar nach einem spezifischen Viewlet eingefügt.

<hidden ...> die hier angegebenen Views werden in diesem Viewlet-Manager nicht angezeigt.

Viewlet-Manager definieren

Viewlet-Manager werden in `zcml`-Dateien angegeben. Wie dies geschieht, lässt sich leicht in `plone/app/layout/viewlets/configure.zcml` nachvollziehen, z.B. für `plone.portaltop`:

```
<browser:viewletManager
    name="plone.portaltop"
    provides=".interfaces.IPortalTop"
    permission="zope2.View"
    class="plone.app.viewletmanager.manager.OrderedViewletManager"
    />
```

Dabei ist das Interface `IPortalTop` definiert in `plone/app/layout/viewlets/interfaces.py`. Allgemein dienen solche Interfaces dazu, Viewlets bestimmten Viewlet-Managern zuzuweisen.

In PageTemplates lassen sich Viewlet-Manager aufrufen z.B. mit:

```
<div tal:replace="structure provider:plone.portaltop" />
```

Viewlets anderen Viewlet-Managern zuweisen

In unserem Beispiel soll die *personal bar* vom `plone.portaltop`- zum `plone.contentviews`-Manager verschoben werden. Hierzu wird in `src/vs.theme/vs/theme/browser/configure.zcml` folgendes angegeben:

```
<browser:viewlet
    name="vs.personal_bar"
    manager="plone.app.layout.viewlets.interfaces.IContentViews"
    layer=".interfaces.IThemeSpecific"
    class="plone.app.layout.viewlets.common.PersonalBarViewlet"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    permission="zope2.View"
    />
```

`personal_bar` erhält einen anderen Namen, einen neuen Viewlet-Manager und wird mit `layer` an unser spezifisches Theme gebunden. Dieses `IThemeSpecific`-Interface wurde bereits vom `plone3_theme`-Template in `browser/configure.zcml` konfiguriert:

```
<interface
   interface=".interfaces.IThemeSpecific"
   type="zope.publisher.interfaces.browser.IBrowserSkinType"
   name="vs.theme"
   />
```

Schließlich wird noch in `src/vs.theme/vs/theme/profiles/default/viewlets.xml` angegeben, dass das alte Viewlet nicht mehr und das neue in `plone.contentviews` angezeigt werden sollen:

```
<hidden manager="plone.portaltop" skinname="vs.theme">
    ...
    <viewlet name="plone.personal_bar" />
    ...
</hidden>
...
<order manager="plone.contentviews" skinname="vs.theme" based-on="Plone Default">
    <viewlet name="vs.personal_bar" />
    ...
</order>
```

Neue Viewlets erstellen

Neue Viewlets werden in `browser.configure.zcml` registriert, z.B. mit:

```
<browser:viewlet
    name="vs.footer"
    manager="plone.app.layout.viewlets.interfaces.IPortalFooter"
    layer=".interfaces.IThemeSpecific"
    template="templates/footer.pt"
    permission="zope2.View"
    />
```

Mit der Angabe für `layer` wird der View an unser spezifisches Theme gebunden und mit `template` lässt sich auch ein eigenes Page Template angeben.

Viewlets überschreiben

Die Pfadleiste (*breadcrumbs*) ist z.B. ein Viewlet. Um es ändern zu können, muss es zunächst in der Datei `browser/configure.zcml` registriert werden:

```
<browser:viewlet
    name="vs.path_bar"
    manager="plone.app.layout.viewlets.interfaces.IContentViews"
    layer=".interfaces.IThemeSpecific"
    class=".viewlets.PathBarViewlet"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    permission="zope2.View"
    />
```

Der Name des Viewlets wird angegeben mit `vs.path_bar` und der Layer bindet das Viewlet an unser spezifisches Theme. Nun definieren wir noch die angegebene Klasse `PathBarViewlet`, indem wir die Datei `browser/viewlets.py` mit folgendem Inhalt anlegen:

```
from zope.component import getMultiAdapter
from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile
from plone.app.layout.viewlets import common

class PathBarViewlet(common.PathBarViewlet):
    render = ViewPageTemplateFile('templates/path_bar.pt')
```

Hier wird verwiesen auf ein PageTemplate `browser/templates/path_bar.pt`.

content-Macros

Plone 4 kommt mit den folgenden Slots:

`content-title` Slot mit dem View des Titels

Der Inhalt wird in `main_template.pt` generiert.

`content-description` Slot mit dem View der Beschreibung.

Der Inhalt wird in `main_template.pt` generiert.

`content-core` Slot, der in `main_template.pt` unterhalb von `content-description` angezeigt wird. Er zeigt den Inhalt einer Seite, die Liste eines Ordners o.ä. an.

`main` Dieser Slot ist weiterhin in Plone 4 vorhanden um rückwärtskompatibel zu bleiben und Views zu rendern, die ohne Viewlet-Manager oder modifizierte Titel und Beschreibungen auskommen.

Viewlet-Manager

Alle Viewlet-Manager für `content` werden nicht mehr wie in Plone 3 in eigenen Templates verwaltet sondern immer im `main_template.pt`. Hierdurch werden die Templates für Artikeltypen nochmals deutlich vereinfacht, sehen Sie z.B. `document_view`:

```
<metal:content-core fill-slot="content-core">
    <metal:content-core define-macro="content-core">
        <metal:field use-macro="python:context.widget('text', mode='view')">
            Body text
        </metal:field>
    </metal:content-core>
</metal:content-core>
```

Portlets ändern, zuweisen, blockieren und entfernen

Portlet ändern

Um eine angepasste SearchBox in der linken Spalte anzeigen, wird zunächst in `browser/templates/` die Datei `search_portlet.pt` mit den gewünschten Anpassungen angelegt. Anschließend wird in `browser/configure.zcml` dieses Template angegeben:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:browser="http://namespaces.zope.org/browser"
    xmlns:plone="http://namespaces.plone.org/plone"
    i18n_domain="vs.theme">

<include package="plone.app.portlets" />
...
<plone:portletRenderer
    portlet="plone.app.portlets.portlets.search.ISearchPortlet"
    layer=".interfaces.IThemeSpecific"
    template="templates/search_portlet.pt"
    />
...
</configure>
```

Portlets zuweisen

Plone 3.0

In `setuphandlers.py` lässt sich das Search-Portlet der linken Spalte zuweisen:

```
from zope.component import getMultiAdapter
from zope.component import getUtility
from plone.portlets.interfaces import IPortletAssignmentMapping
from plone.app.portlets import portlets

class Generator:
    def setupPortlets(self, portal):
        leftColumn = getUtility(IPortletManager, name=u'plone.leftcolumn',␣
↪context=portal)
        rightColumn = getUtility(IPortletManager, name=u'plone.rightcolumn',␣
↪context=portal)

        left = getMultiAdapter((portal, leftColumn,), IPortletAssignmentMapping,␣
↪context=portal)
        right = getMultiAdapter((portal, rightColumn,), IPortletAssignmentMapping,␣
↪context=portal)

        if u'search' not in left:
            left[u'search'] = portlets.search.Assignment(enableLivesearch=True,)

def setupVarious(context):
    if context.readDataFile('vs.theme_various.txt') is None:
        return
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    site = context.getSite()
    gen = Generator()
    gen.setupPortlets(site)
```

Plone 3.1

Hier erfolgt die Zuweisung in `profiles/default/portlets.xml`:

```
<?xml version="1.0"?>
<portlets
    xmlns:i18n="http://xml.zope.org/namespaces/i18n"
    i18n:domain="plone">

<assignment
            manager="plone.leftcolumn"
            category="context"
            key="/"
            type="portlets.Search"
            name="search"
            insert-before="*"
            >

    <property name="enableLivesearch">True</property>

</assignment>
</portlets>
```

`type` (erforderlich) Der Name des Portlets entsprechend dem name-Atribut in der `<portlet />`- Anweisung in `portlets.xml`.

`manager` (erforderlich) Der Name des Portlet-Manager, der verwendet werden soll. Portlet-Manager werden registriert in `portlets.xml`, z.B.:

```
<portletmanager
  name="plone.leftcolumn"
  type="plone.app.portlets.interfaces.ILeftColumn"
  />
```

In Plone verfügbare Portlet-Manager sind:

plone.leftcolumn und plone.rightcolumn für die linke und rechte Spalte

plone.dashboard1 bis plone.dashboard4 für die vier Spalten des Dashboard.

`category` (erforderlich) Die zu verwendende Kategorie. Mögliche Kategorien sind:

- `context`
- `content_type`
- `group`
- `user` (Diese Angabe ist vmtl. nur für die Dashboard-Portlet-Manager sinnvoll.)

`key` (erforderlich) Der Schlüssel, mit dem das Portlet zugewiesen wird.

- Für die context-Kategorie ist der Schlüssel die Angabe des Pfades relativ zur Site-Root, z.B. `/`.

- Für die `content_type`-Kategorie ist der Schlüssel die Angabe des Artikeltyps.
- Für die `group`-Kategorie ist der Schlüssel die Angabe der ID einer bestimmten Gruppe.
- Für die `user`-Kategorie ist der Schlüssel die Angabe der user-ID.

name (optional) Der Name der Zuweisung. Wird keine Name angegeben, wird ein eindeutiger Name erzeugt. Wird ein bereits bestehender Name verwendet, der denselben key, dieselbe Kategorie und denselben Portlet-Manager verwendet, so wird dieser überschrieben.

insert-before (optional) Dieser Parameter kann verwendet werden um die Reihenfolge der Portlets festzulegen.

- Ist der Wert `*`, wird das Porlet an oberster Stelle platziert.
- Ist der Wert der Name eines anderen Portlets, wird das einzufügende Portlet direkt über diesem angezeigt.
- Wird kein Wert angegeben, wird das Porlet zuunterst angezeigt.

Die Portlets werden dabei in der Reihenfolge verarbeitet und eingefügt, wie sie in der `portlets.xml`-Datei angegeben sind.

Hier noch ein Beispiel für das Zuweisen eines Portlets im Dashboard eines Nutzers:

```
<assignment name="quick-links" category="user" key="veit"
   manager="plone.dashboard1" type="plone.portlet.collection.Collection">
 <property name="show_more">True</property>
 <property name="header">Quick links</property>
 <property name="limit">10</property>
 <property
    name="target_collection">/quick-links/quick-links</property>
 <property name="random">False</property>
 <property name="show_dates">False</property>
</assignment>
```

Übernommene Portlets blockieren

Plone 3.0

Von übergeordneten Objekten übernommene Portlets lassen sich in `setuphandlers.py` blockieren, z.B. mit:

```
from plone.portlets.constants import CONTEXT_CATEGORY as CONTEXT_PORTLETS

class Generator:

    def setupPortlets(self, portal):
        rightColumn = getUtility(IPortletManager, name=u'plone.rightcolumn',␣
↪context=portal)
        portletAssignments = getMultiAdapter((members, rightColumn,),␣
↪ILocalPortletAssignmentManager)
        portletAssignments.setBlacklistStatus(CONTEXT_PORTLETS, True)
```

Plone 3.1

Das Blockieren lässt sich hier in `profiles/default/portlets.xml` konfigurieren, z.B.:

```
<blacklist
    manager="plone.rightcolumn"
    location="/Members"
    category="context"
    status="block"
    />
```

manager (erforderlich) Der Name des Portlet-Managers (Spalte), für die die Portlets nicht übernommen werden sollen.

category (erforderlich) Die Kategorie, die geblockt werden soll: context, group oder content_type.

location (erforderlich) Ein relativer Pfad, der den Ordner angibt, in dem die Portlets geblockt werden sollen.

status (erforderlich) Der Status für übernommene Portlets einer Kategorie in einem bestimmten Ordner:

block Portlets des übergeordneten Objekts werden übernommen.

show Portlets der angegebenen Kategorie werden angezeigt.

acquire Portlets des übergeordneten Objekts werden übernommen.

Portlets ausblenden

Portlets lassen sich in setuphandlers.py ausblenden, z.B. mit:

```
from zope.component import getUtility
from zope.component import getMultiAdapter
from plone.portlets.interfaces import IPortletManager
from plone.portlets.interfaces import IPortletAssignmentMapping

class Generator:
    def setupPortlets(self, portal):
        rightColumn = getUtility(IPortletManager, name=u'plone.rightcolumn',␣
↪context=portal)
        right = getMultiAdapter((portal, rightColumn,), IPortletAssignmentMapping,␣
↪context=portal)
        if u'calendar' in right:
            del right["calendar"]

def setupVarious(context):
    if context.readDataFile('vs.theme_various.txt') is None:
        return
    site = context.getSite()
    gen = Generator()
    gen.setupPortlets(site)
```

Portlets entfernen

Plone 3.0

Portlets lassen sich in setuphandlers.py entfernen, z.B. mit:

```
from zope.component import getSiteManager
from zope.component import getUtilitiesFor
from plone.portlets.interfaces import IPortletType
from Products.CMFCore.utils import getToolByName

def removeRegistrantsPortlet(self):
    sm = getSiteManager()
    for name, portletType in getUtilitiesFor(IPortletType):
        if name == "portlets.Registrants":
            sm.unregisterUtility(provided=IPortletType, name=name)
```

Plone 3.1

Hier erfolgt das entfernen von Portlets in `profiles/default/portlets.xml`:

```
<assignment
    remove="true"
    name="calendar"
    category="context"
    key="/"
    manager="plone.rightcolumn"
    type="portlets.Calendar" />
```

Plone 4

In Plone 4 wurde das visible-Attribut zum Ein- oder Ausblenden der Portlets eingeführt:

```
<assignment
    visible="0"
    name="calendar"
    category="context"
    key="/"
    manager="plone.rightcolumn"
    type="portlets.Calendar" />
```

Das Ausblenden aller Portlets in einem bestimmten Kontext erfolgt mit dem purge-Attribut:

```
<assignment
    purge="True"
    manager="plone.rightcolumn"
    category="context"
    key="/Plone/news"
    />
```

Portlet erstellen

Portlet mit *Site Actions* erstellen

Für dieses Portlet erstellen wir ein Unterpaket entsprechend den Konventionen in `plone.app.portlets`. Für unser `Site-Actions`-Portlet wird das Page Template `siteactions.pt` im Ordner `portlets` erstellt:

```
<dl class="portlet portletSiteActions"
    i18n:domain="vs.theme">

    <dt class="portletHeader">
        <span class="portletTopLeft"></span>
        Site Actions
        <span class="portletTopRight"></span>
    </dt>

    <tal:actions tal:define="accesskeys python: {'sitemap' : '3', 'accessibility' : '0
↪', 'contact' : '9'};"
                 tal:condition="view/site_actions">
        <dd class="portletItem"
            tal:repeat="saction view/site_actions"
            tal:attributes="id string:siteaction-${saction/id}">
            <a href=""
               tal:define="title saction/title;
                           id saction/id;
                           accesskey python: accesskeys.get(id, '');"
               i18n:attributes="title"
               i18n:translate=""
               tal:content="title"
               tal:attributes="href saction/url;
                               target saction/link_target|nothing;
                               title title;
                               accesskey accesskey;"
            >Site action</a>
        </dd>
    </tal:actions>
</dl>
```

Dies entspricht dem Aufbau der meisten Plone-Portlets. Dabei ist die Darstellungslogik, welche `siteactions` angezeigt werden, in den View `siteactions.py` ausgelagert worden.

Auch der Aufbau von `siteactions.py` entspricht der üblichen Konvention. Zunächst wird einiges importiert, darunter auch das `base`-Modul von `plone.app.portlets`, das verschiedene Basisklassen wie `Assignment` und `Renderer` zum Erstellen eines neuen Portlets bereitstellt:

```
from zope import schema
from zope.component import getMultiAdapter
from zope.formlib import form
from zope.interface import implements

from plone.app.portlets.portlets import base
from plone.portlets.interfaces import IPortletDataProvider
from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile

from vs.theme import VsThemeMessageFactory as _

class ISiteActionsPortlet(IPortletDataProvider):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """A portlet which shows the available site actions.
    """

class Assignment(base.Assignment):
    implements(ISiteActionsPortlet)


    @property
    def title(self):
        return _(u"Site actions")

class Renderer(base.Renderer):
    render = ViewPageTemplateFile('siteactions.pt')
    title = _('box_siteactions', default=u"Site actions")

    def site_actions(self):
        context_state = getMultiAdapter((self.context, self.request),
                                        name=u'plone_context_state')
        self.siteactions = context_state.actions('site_actions')
        return self.siteactions
```

Schließlich werden noch Klassen für `AddForm` definiert, die Nutzern das Erstellen des Site-Action-Portlets erlauben:

```
class AddForm(base.NullAddForm):
    form_fields = form.Fields(ISiteActionsPortlet)
    label = _(u"Add Site Actions Portlet")
    description = _(u"This portlet lists the available site actions.")

    def create(self):
        return Assignment()
```

Konfigurieren und Registrieren neuer Portlet-Typen

Um neue Portlet-Typen zu konfigurieren, wird die Datei `portlets/configure.zcml` mit folgendem Inhalt erstellt:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:plone="http://namespaces.plone.org/plone">

    <include package="plone.app.portlets" />

    <plone:portlet
        name="vs.theme.SiteActionsPortlet"
        interface=".siteactions.ISiteActionsPortlet"
        assignment=".siteactions.Assignment"
        renderer=".siteactions.Renderer"
        addview=".siteactions.AddForm"
        />

</configure>
```

Damit werden einige Hilfsmethoden, Adapter und Views registriert, Und falls Sie ein editierbares Portlet erstellen wollen, können Sie das `editview`-Attribut hinzufügen und statt `NullAddForm` die `AddForm`-Klasse angeben. Ein solches Portlet ist beschrieben in Portlet erstellen.

Registrieren von Portlets

Ab Plone 3.1 muss das Portlet zusätzlich in `src/vs.theme/vs/theme/profiles/default/portlets.xml` angegeben werden:

```
<?xml version="1.0"?>
<portlets>
    <portlet
       addview="vs.theme.SiteActionsPortlet"
       title="Site Actions"
       description="A portlet which can show the available site actions."
       />
</portlets>
```

Dabei entspricht die Angabe für `addview` dem Namen des Portlets, der in `portlets/configure.zcml` angegeben wurde.

Portlet-Manager hinzufügen

Zunächst wird ein Viewlet erstellt, das auf einen Portlet-Manager verweist – in unserem Fall `vs.abovecontentportlets` – und für den Viewlet-Manager `IContentViews` registriert wird. Anschließend wird für diesen Viewlet-Manager noch ein Management-View erstellt.

Viewlet erstellen

Fügen Sie in `browser/templates` die Datei `abovecontentportlets.pt` mit folgendem Inhalt hinzu:

```
<tal:block replace="structure provider:vs.abovecontentportlets" />
```

Dieses Viewlet wird nun registriert in `browser/configure.zcml`:

```
<browser:viewlet
    name="vs.abovecontentportlets"
    manager="plone.app.layout.viewlets.interfaces.IContentViews"
    layer=".interfaces.IThemeSpecific"
    template="templates/abovecontentportlets.pt"
    permission="zope2.View"
/>
```

Browserlayer registrieren

Das Viewlet steht nur zur Verfügung sofern das Interface `IThemeSpecific` in der Site zur Verfügung steht. Durch das `plone3_theme`-Template sollte dieses Marker-Interface bereits in `browser/interfaces.py` erstellt worden sein:

```
from plone.theme.interfaces import IDefaultPloneLayer

class IThemeSpecific(IDefaultPloneLayer):
    """Marker interface that defines a Zope 3 browser layer.
       If you need to register a viewlet only for the
       "vs.theme" theme, this interface must be its layer.
    """
```

Nun wird dieses Interface als paketspezifischer Browserlayer registriert in `profiles/default/browserlayer.xml`:

```
<?xml version="1.0"?>
<layers>
    <layer name="vs.theme.layer"
           interface="vs.theme.browser.interfaces.IThemeSpecific" />
</layers>
```

Dieser Browserlayer steht nun jenen Sites zur Verfügung, in denen dieses Profil importiert wurde.

Portlet-Manager erstellen

Zunächst wird ein Marker-Interface in `browser/interfaces.py` erstellt:

```
from plone.portlets.interfaces import IPortletManager

class IVsAboveContent(IPortletManager):
    """Portlet manager above the content area.
    """
```

Anschließend wird in der Datei `profiles/default/portlets.xml` der neue Portlet-Manager registriert:

```
<?xml version="1.0"?>
<portlets>
    <portletmanager
         name="vs.abovecontentportlets"
         type="vs.theme.browser.interfaces.IVsAboveContent"
    />
</portlets>
```

Erstellen eines Management-Views für den Portlet-Manager

Um die Portlets des `vs.abovecontentportlets` zu verwalten, wird ein neuer View erstellt und hierfür zunächst folgende Zeilen in `browser/configure.zcml` eingetragen:

```
<browser:page
    for="plone.portlets.interfaces.ILocalPortletAssignable"
    class="plone.app.portlets.browser.manage.ManageContextualPortlets"
    name="manage-vsabove"
    template="templates/managevsabove.pt"
    permission="plone.app.portlets.ManagePortlets"
/>
```

Und falls nicht bereits zu einem früheren Zeitpunkt geschehen, sollte noch das Paket `plone.app.portlets` eingeschlossen werden:

```
<include package="plone.app.portlets" />
```

Schließlich wird dann noch das Page-Template `browser/templates/managevsabove.pt` erstellt:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      xmlns:i18n="http://xml.zope.org/namespaces/i18n"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="plone">
<head>
    <div metal:fill-slot="javascript_head_slot" tal:omit-tag="">
        <link type="text/css" rel="kinetic-stylesheet"
            tal:attributes="href string:${context/absolute_url}/++resource++manage-
↪portlets.kss"/>
    </div>
</head>
<body>
<div metal:fill-slot="main">
  <h1 class="documentFirstHeading">Manage above content portlets</h1>
  <span tal:replace="structure provider:vs.abovecontentportlets" />
</div>
</body>
</html>
```

Nach einem Neustart des Zope-Servers sollten sich nun die Portlets verwalten lassen wenn folgende URL aufgerufen wird:

```
http://localhost/mysite/@@manage-vsabove
```

Zope Page Templates (ZPT)

Zope Page Templates verwenden TAL (Template Attribute Language), z.B.:

```
<title tal:content="context/title">Page Title</title>
```

Dabei ist `tal:content` das TAL-Attribut wobei `tal` den XML-Namensraum angibt und `content` darauf hinweist, dass der Inhalt des `title`-Tags gesetzt werden soll. Der Wert `context/title` schließlich ist ein Ausdruck, der den in den Tag einzufügenden Text liefert und dabei `Page Title` ersetzt.

Ausdrücke

Der Text `context/title` ist ein einfacher Pfadausdruck der *TAL Expression Syntax* (TALES), der die Titel-Eigenschaft des Kontexts aufruft. Andere häufig verwendete Pfadausdrücke sind:

`request/URL` Die URL des aktuellen Web-Requests.

`user/getUserName` Der Login-Name des aktuell angemeldeten Nutzers.

`container/objectIds` Eine Liste aller IDs von Objekten im selben Ordner wie das Template.

Jeder Pfadausdruck startet mit einem Variablenname und kann, getrennt durch einen Schrägstrich (`/`), um den Namen eines Unterobjekts oder eine Eigenschaft spezifiziert werden.

Die Menge der verfügbaren Variablen, wie `request` oder `user`, ist relativ gering und wird später noch vollständig beschrieben werden. Zudem werde ich zeigen, wie eigene Variablen definiert werden können.

Inhalte

Angenommen unser Template hätte die ID `my_page`, und wir wollten Text dynamisch einfügen, so könnten wir dies innerhalb eines `span`-Tags machen:

```
Die URL ist <span tal:replace="request/URL">URL</span>.
```

Beachten Sie, dass der gesamte Tag ersetzt wird durch das Ergebnis der TAL-Anweisung, also:

```
Die URL ist http://localhost:8080/mysite/my_page.
```

Soll ein Tag erhalten bleiben, wird `tal:content` verwendet:

```
<title tal:content="template/title">The Title</title>
```

Aufzählungen

Soll eine ganze Liste von Werten automatisch eingefügt werden, so kann dies mit `tal:repeat` erfolgen, z.B.:

```
<table>
    <tr>
        <th>#</th><th>Id</th><th>Meta-Type</th><th>Title</th>
    </tr>
    <tr tal:repeat="item container/objectValues">
        <td tal:content="repeat/item/number">#</td>
        <td tal:content="item/id">Id</td>
        <td tal:content="item/meta_type">Meta-Type</td>
        <td tal:content="item/title">Title</td>
    </tr>
</table>
```

Der `tal:repeat`-Ausdruck auf einer Tabellenzeile bedeutet, dass diese Zeile für jeden Artikel in diese Container erstellt wird. Dabei wird für jede Zeile ein Artikel der Liste als `item`-Variable verwendet und dessen Werte ausgelesen, wobei statt `item` auch jeder andere Name verwendet werden kann.

`repeat/item/number` ist die fortlaufende Nummerierung der Artikel innerhalb der Aufzählung. Soll die Nummerierung mit `0` beginnen, muss statt `number` `index` verwendet werden, soll die fortlaufende Nummerierung alphabetisch sein, ist das Attribut `letter`.

Selbstverständlich können auch mehrere Aufzählungen ineinander verschachtelt werden.

Bedingungen

Um jeder zweiten Zeile der obigen Tabelle eine andere Hintergrundfarbe zu geben, können entsprechende Bedingungen angegeben werden:

```
<table>
    <tr>
        <th>#</th><th>Id</th><th>Meta-Type</th><th>Title</th>
    </tr>
    <tbody tal:repeat="item container/objectValues">
        <tr bgcolor="#DDE0E8" tal:condition="repeat/item/even">
            <td tal:content="repeat/item/number">#</td>
            <td tal:content="item/id">Id</td>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            <td tal:content="item/meta_type">Meta-Type</td>
            <td tal:content="item/title">Title</td>
        </tr>
        <tr tal:condition="repeat/item/odd">
            <td tal:content="repeat/item/number">#</td>
            <td tal:content="item/id">Id</td>
            <td tal:content="item/meta_type">Meta-Type</td>
            <td tal:content="item/title">Title</td>
        </tr>
    </tbody>
</table>
```

Chameleon

Chameleon ist eine HTML/XML-Template-Engine, die mit der Standardinstallation von Plone 5 mitkommt, jedoch auch als Zusatzprodukt für Plone 4 installiert werden kann.

Chameleon vereinfacht das Einfügen von Variablen erheblich, z.B. kann nun anstat:

```
<a tal:attributes="href href" tal:content="text" />
```

folgendes verwendet werden:

```
<a href="${href}">${text}</a>
```

Zudem lassen sich mit der `__html__`-Methode einfach Strings einfügen ohne dass diese *escaped* werden müssten, z.B.:

```
class Markup(object):
    def __init__(self, s):
        self.s = s

    def __html__(self):
        return s
```

Anschließend kann eine Instanz dieser Klasse verwendet werden um einen String einzufügen:

```
from chameleon.utils import Markup
form.status = Markup('<div class="note">Note</div>')
```

Beachten Sie jedoch, dass Sie nun selbst verantwortlich sind für das *Escaping* unerwünschter Nutzereingaben.

Siehe auch:

- Chameleon documentation

Kinetic Style Sheets

Einführung in Kinetic Style Sheets

In Plone funktionieren alle Anwendungen ohne JavaScript, und JavaScript wird ausschließlich zur Verbesserung des User Interfaces verwendet. Darüberhinaus wird vermieden, JavaScript direkt in eine Seite einzufügen und viele Aufgaben lassen sich ohne selbst JavaScript schreiben zu müssen, erledigen.

Beim Aufruf einer Plone-Seite mit KSS läuft folgende Sequenz ab:

1. KSS-Dateien, die in die Seite mit einem `<link />`-Tag eingebunden sind, werden analysiert.
2. Dabei werden die im KSS definierten Aktionen an nutzerseitige Ereignisse gebunden, z.B. das Klicken auf einen Schalter.
3. Tritt ein solches Ereigns ein, wird die verknüpfte Aktion ausgeführt. Dies kann ein einfacher Effekt auf Nutzerseite sein, häufig jedoch wird eine asynchrone Anfrage an den Server gestellt werden.
4. Eine serverseitige Aktion führt anwendungsspezifische Operationen durch.
5. Die serverseitige Anwendung verknüpft die Antwort mit einem oder mehreren Befehlen und sendet alles gemeinsam an den Client. Befehle werden mit PlugIns verfügbar gemacht, wobei diverse PlugIns bereits mit Plone mitkommen, so z.B. das Ändern von Text oder das Aktualisieren von Portlets.
6. Der Client führt die Befehle aus.

Bemerkung: Mit firekiss.xpi steht eine Erweiterung fpr firebug zur Verfügung, mit der sich KSS-Dateien untersuchen lassen.

Bemerkung: Wie eigene KSS-PlugIns geschrieben werden, wird hier nicht Thema sein. Sie erhalten jedoch weitere Informationen zu KSS unter http://kssproject.org.

Kinetic Style Sheet

1. Zunächst erstellen wir die KSS-Datei `registration.kss`:

```
#confirm-registrant input:click {
    evt-click-preventdefault: true;
    action-server:confirmRegistrant;
    confirm-registrant: kssAttr('confirm');
}
```

   Die erste Zeile identifiziert ein `input`-Feld in einem Knoten mit der ID `confirm-registrant`. Um zu gewährleisten, dass das Formular nicht wie üblich abgeschickt wird, setzen wir `evt-click-preventdefault` auf `true`. Stattdessen soll die serverseitige Aktion `confirmRegistrant` ausgeführt werden wobei der Parameter `confirm` übergeben wird.

2. Anschließend wird die Datei `registration.kss` als Browser-Ressource in `browser/configure.zcml` registriert:

```
<browser:resource
    name="registration.kss"
    file="registration.kss"
    />
```

3. Die KSS-Datei wird nun in die Seite mit folgendem Tag eingebunden:

```
<link rel="kinetic-stylesheet"
      type="text/css"
      href="http://localhost:8080/mysite/++resource++registration.kss" />
```

Häufig empfiehlt es sich jedoch, die KSS-Datei in der *KSS-Registry* anzumelden. Die Anmeldung erfolgt dann im Profil `src/vs.registration/vs/theme/profiles/default/kssregistry.xml` mit:

```
<?xml version="1.0"?>
<object name="portal_kss" meta_type="KSS Registry">
 <kineticstylesheet
     cacheable="True"
     compression="safe"
     cookable="True"
     enabled="1"
     expression=""
     id="registration.kss"/>
</object>
```

Wird nun der *import step* in `portal_setup` durchlaufen oder das `vs.registration`-Produkt neu installiert, sollte der Eintrag mit `++resource++registration.kss` in `portal_kss` eingetragen sein.

Page Template

Im Page Template `src/vs.registration/vs/registration/browser/registration.pt` wird nun ein Formular mit der ID `confirm-registrant` und den entsprechenden `input`-Feldern eingetragen:

```
<tal:registrants condition="view/have_registrants">
    <h2 i18n:translate="title_registration_contents">Registrants</h2>
    <form action="confirmRegistrant">
        <tal:block repeat="registrant view/registrants">
            <dt>
                <a tal:attributes="href registrant/url"
                   tal:content="registrant/title" />
            </dt>
            <dd tal:content="registrant/address" />
            <dd>
                <input class="kssattr-confirm-yes"
                       type="submit"
                       name="vs.registration.confirm.confirm"
                       value="yes"
                       i18n:attributes="value"
                       i18n:attributes="confirm-button"
                       />
                <input class="kssattr-confirm-no"
                       type="submit"
                       name="vs.registration.confirm.reject"
                       value="no"
                       i18n:attributes="value"
                       i18n:attributes="reject-button"
                       />
            </dd>
        </tal:block>
    </form>
</tal:registrants>
```

Serverseitige Logik

In `src/vs.registration/vs/registration/browser/registration.kss` wird die Serveraktion referenziert mit:

```
action-server:confirmRegistrant;
confirm-registrant: kssAttr('confirm');
```

Beim Aufruf der Aktion wird `confirmRegistran` im aktuellen Kontext mit den entsprechenden Parametern aufgerufen. So wird z.B. eine HTTP POST-Anfrage zu der URL `http://localhost:8080/mysite/registration/confirmRegistrant` mit dem Parameter `confirm` gesendet.

Die serverseitige Aktion wird meist als *View* implementiert. Entsprechend geben wir in `browser/configure.zcml` folgendes an:

```
<browser:page
    for="vs.registration.interfaces.IRegistration"
    name="confirmRegistrant"
    class=".confirmations.confirmedRegistrants"
    attribute="confirm_registrant"
    permission="zope2.RequestReview"
    />
```

Und die `confirmedRegistrants`-Klasse in `browser/confirmations` sieht dann so aus:

```
from zope.interface import alsoProvides
from kss.core import kssaction
from plone.app.kss.plonekssview import PloneKSSView
from plone.app.layout.globals.interfaces import IViewView
from Akquisition import aq_inner
from Products.Five.browser import BrowserView
from vs.Registration.interfaces import IConfirmations

class confirmedRegistrants
    @kssaction
    def confirm_registrant
        confirm = confirm.lower()
        if confirm not in ("confirm", "reject"):
            return
```

Tipps

Styles

Zentriertes Design und unterschiedlich gestaltete Bereiche der Website lassen sich in Plone einfach realisieren.

Zentriertes Design

```
#visual-portal-wrapper {
    width: 62em;
    margin-left: auto;
    margin-right: auto;
}
```

Unterschiedlich gestaltete Bereiche der Website

Bereiche

In `parts/plone/CMFPlone/skins/plone_templates/main_template.pt`: wird für den `body`-Tag eine css-Klasse defniniert, die es erlaubt, unterschiedliche Gestaltungen für einzelne Bereiche der Wwbsite anzugeben:

```
<body tal:attributes="class string:${here/getSectionFromURL} .">
```

Dies führt dann z.B. zu folgenden `body`-Tag, je nachdem in welchem Bereich sich die gerade aufgerufene Seite befindet:

```
<body class="section-news">
<body class="section-events">
```

So lassen sich die Bereiche auch gestalterisch unterscheiden:

```
body.section-news {
    ...
}

body.section-events {
    ...
}
```

Templates

Analog lassen sich auch die Gestaltungen für einzelne Templates unterscheiden:

```
<body tal:attributes="class ... template-${template/id}; ...">
```

```
body.template-frontpage_view {
    ...
}
```

Javascripts

Plone bringt einige Javscripts mit, die sich einfach in Templates und Inhalten verwenden lassen.

`form_tabbing.js` Um Reiter in Formularen zu erhalten, erwartet dieses Skript folgendes Markup:

```
<form class="enableFormTabbing">
    <fieldset id="fieldset-id1">
        <legend id="fieldsetlegend-id1">Tab one</legend>
    </fieldset>
    <fieldset id="fieldset-id2">
        <legend id="fieldsetlegend-id2">Tab two</legend>
    </fieldset>
</form>
```

Alternativ kann auch folgendes Markup verwendet werden:

```
<dl class="enableFormTabbing">
    <dt id="fieldsetlegend-id1">tab one</dt>
    <dd id="fieldset-id1">content one</dd>
    <dt id="fieldsetlegend-id2">tab two</dt>
    <dd id="fieldset-id2">content two</dd>
</dl>
```

`table_sorter.js` Um Tabellen zu sortieren kann dieses Javascript einfach in folgendem Markup verwendet werden:

```
<table class="listing" id="my-table">
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last name</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>
                Veit
            </td>
            <td>
                Schiele
            </td>
        </tr>
        <tr>
            <td>
                Sönke
            </td>
            <td>
                Löffler
            </td>
        </tr>
    </tbody>
</table>
```

- Beachten Sie bitte, dass das `table_sorter.js`-Javascript normalerweise nur angemeldeten Nutzern zur Verfügung steht. Wollen Sie es auch anonymen Nutzern zur Verfügung stellen, sollte Ihr `jsregistry.xml`-Profil so aussehen:

```
<?xml version="1.0"?>
<object name="portal_javascripts" meta_type="JavaScripts Registry">
...
<javascript
    cacheable="True"
    compression="safe"
    cookable="True"
    enabled="True"
    expression=""
    id="table_sorter.js"
    inline="False"/>
</object>
```

- Das Skript erwartet die Klasse `listing` und eine eindeutige ID für die Tabelle sowie `<th>`-Tags innerhalb von `<thead>`-Tags.
- Soll eine Tabelle der Klasse `listing` keine sortierbaren Spalten enthlaten, kann der Tabelle eine Klasse `nosort` hinzugefügt werden.
- Soll nur eine Spalte innerhalb einer Tabelle nicht sortiert werden können, so kann dem entsprechenden `<th>`-Tag die Klasse `nosort` zugewiesen werden.

`collapsiblesections.js` Dieses Javascript kannn bei folgendem Markup verwendet werden:

```
<dl class="collapsible">
    <dt class="collapsibleHeader">
        Title
    </dt>
    <dd class="collapsibleContent">
        Content
    </dd>
</dl>
```

Sobald `collapsible` umgeschaltet wurde, erhalt das `dl`-Tag eine zusätzliche Klasse, die zwischen `collapsedBlockCollapsible` und `expandedBlockCollapsible` hin- und herschaltet. Hierfür können Sie dann z.B. folgende CSS-Anweisungen angeben:

```
.expandedBlockCollapsible .collapsibleContent {
    display: block;
}

.collapsedBlockCollapsible .collapsibleContent {
    display: none;
}
```

Wird die `collapsedOnLoad`-Klasse dem `dl`-Tag hinzugefügt, wird die Definitionsliste bereits beim Laden der Seite ausgeklappt.

Wird die `inline`-Klasse für das `dl`-Tag angegeben, wird zwischen `collapsedInlineCollapsible` und `expandedInlineCollapsible` umgeschaltet anstatt zwischen `collapsedBlockCollapsible` und `expandedBlockCollapsible`.

`jQuery` JavaScript-Bibliothek, die die Traversierung und das Event-Handling von HTML-Dokumenten vereinfacht. So lässt sich z.B. in einem Einzeiler angeben, dass alle PDFs in einem neuen Fenster geöffnet werden sollen:

```
jQuery("#content a[ @href $= '.pdf']").attr('target', '_blank');
```

Weitere Informationen zu jQuery erhalten Sie unter:

- http://jquery.com/
- http://docs.jquery.com

Und mit FireQuery gibt es eine Firefox-Extension, die in Firebug integriert ist.

3.5 Archetypes-Artikeltypen

3.5.1 Paket-Layout

Der neue Artikeltyp soll als neues Paket erstellt werden. Entsprechend unserer Anforderung nennen wir es `vs.registration`. Um dieses Paket zu erstellen verwenden wir wieder PasteScript:

```
$ cd src
$ ../bin/zopeskel archetype vs.registration
```

Antworten Sie dabei auf die Frage *Are you creating a Zope 2 Product?* mit *True*.

Anschließend informieren wir die Buildout-Umgebung von unserem neuen Paket. Hierzu ändern wir `buildout.cfg`:

```
[buildout]
...
develop
    src/vs.policy
    src/vs.theme
    src/vs.registration
...
eggs =
    elementtree
    vs.policy
    vs.theme
    vs.registration
...
```

Nun wird das Buildout-Skript erneut aufgerufen:

```
$ ./bin/buildout -o
```

Entgegen dem Policy-Produkt fügen wir keinen neuen *zcml-slug* hinzu, sondern definieren es als Abhängigkeit in `vs.policy`. Deshalb fügen wir in `vs.policy/configure.zcml` folgendes hinzu:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five"
    xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
    i18n_domain="vs.policy">

    <include package="vs.registration" />
    ...
</configure>
```

Interfaces

Interfaces können als formale Dokumentation der Fähigkeiten eines Artikeltyps betrachtet werden.

Folgende Interfaces beschreiben z.B. Registrierungen und die darin enthaltenen registrierten Personen in `myproject/src/vs.registration/vs/registration/interfaces.py`:

```
from zope.interface import Interface
from zope import schema

from zope.app.container.constraints import contains

class IRegistration(Interface):
    """A folder containing registrants
    """
    contains('vs.registration.interfaces.IRegistrants',)

    text = schema.SourceText(title=_(u"Descriptive text"),
                             description=_(u"Descriptive text about this registration
↪"),
                             required=True)

class IRegistrant(Interface):
    """A registrant
    """

    name = schema.TextLine(title=_(u"Registrant name"),
                           required=True)

    address = schema.Text(title=_(u"Address"),
                          description=_(u""),
                          required=True)

    email = schema.TextLine(title=_(u"Email"),
                            description=_(u""),
                            required=True)
```

Im Gegensatz zu `zope.formlib` nutzt Archetypes die angegebenen Titel und Beschreibungen nicht, dennoch sind die Angaben meines Erachtens nützlich um leicht den Funktionsumfang der Artikeltypen erkennen zu können.

Einen Überblick über die verschiedenen Feldtypen erhalten Sie in zope.schema.interfaces.

Damit die Feldattribute übersetzt werden können, wird folgendes in `src/vs.registration/vs/registration/__init__.py` eingetragen:

```
from zope.i18nmessageid import MessageFactory
RegistrationMessageFactory = MessageFactory('vs.registration')
```

Und in `src/vs.registration/vs/registration/interfaces.py`:

```
from vs.registration import RegistrationMessageFactory as _
```

Archetypes

Anmerkung: Eine vollständige Referenz zu Archetypes finden Sie in http://plone.org/documentation/manual/archetypes-developer-manual oder in `Products.Archetypes.atapi`. Für Beispiele empfehlen sich die Quellen von `Products.ATContentTypes`.

Content-Basisklassen

Üblicherweise sind die Content-Klassen von einer der folgenden Klassen abgeleitet:

`BaseContent` ein einfacher, *non-folderish* Artikeltyp, der die Dublin Core-Metaangaben (über die `ExtensibleMetadata`-Mixin-Klasse) enthält.

`BaseFolder` eine `folderish`-Version von `BaseContent`.

`OrderedBaseFolder` Version von `BaseFolder`, der die Sortierung der enthaltenen Objekte erlaubt.

`BaseBTreeFolder` Version von `BaseFolder`, der die Inhalte in einem *binary tree* speichert und damit gut geeignet ist, mehrere tausend Objekte zu enthalten.

Alle vier Klassen sind importierbar von `Products.Archetypes.atapi`.

In unserem Fall sollen *Registration* und *Registrant* jedoch den Plone-Artikeltypen ähnlich sein, z.B. beim Editieren sollen die Felder in verschiedene Reiter kategorisiert werden und auch ein Feld für Verweise soll enthalten sein. Daher werden die Artikeltypen von Plone erweitert, die in `Products.ATContentTypes.content` implementiert sind. *Registrant* erweitert dabei `base.ATCTContent`, in dem das übliche Verhalten aller einfachen (non-folderish) Artikeltypen von Plone bereits definiert ist:

```
class Registrant(base.ATCTContent):
    implements(IRegistrant)

    portal_type = "Registrant"
    _at_rename_after_creation = True
    schema = RegistrantSchema
```

`Registrant` erweitert `base.ATCTContent` und implementiert `IRegistrant`.

`portal_type` ist der eindeutige Name des Artikeltyps.

`at_rename_after_creation` benennt Objekte in die normalisierte Version ihres Titels um.

Schema

Für den obigen Artikeltyp wird das Schema direkt oberhalb der Klasse definiert:

```
RegistrantSchema = schemata.ATContentTypeSchema.copy() + atapi.Schema((

    atapi.StringField('email',
        required=True,
        searchable=True,
        storage=atapi.AnnotationStorage(),
        widget=atapi.StringWidget(label=_(u"Email"),
                                  description=_(u""))
        ),

    ))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
RegistrantSchema['title'].storage = atapi.AnnotationStorage()
RegistrantSchema['title'].widget.label = _(u"Registrant name")
RegistrantSchema['title'].widget.description = _(u"")

RegistrantSchema['description'].storage = atapi.AnnotationStorage()
RegistrantSchema['description'].widget.label = _(u"Address")
RegistrantSchema['description'].widget.description = _("")

RegistrantSchema['email'].storage = atapi.AnnotationStorage()
RegistrantSchema['email'].widget.label = _(u"Email")
RegistrantSchema['email'].widget.description = _(u"")

finalizeATCTSchema(RegistrantSchema, folderish=False, moveDiscussion=False)

class Registrant(base.ATCTContent):
    """Describe a registrant.
    """
    implements(IRegistrant)

    portal_type = "Registrant"
    _at_rename_after_creation = True
    schema = RegistrantSchema

    name = atapi.ATFieldProperty('title')
    address = atapi.ATFieldProperty('description')
    email = atapi.ATFieldProperty('email')

atapi.registerType(Registrant, PROJECTNAME)
```

Zunächst wird das Schema von einem Basistypen, in diesem Fall `ATContentTypeSchema`, kopiert und anschließend das eigene Schema angehängt.

Felder

Im Folgenden eine Liste der gebräuchlichsten Felder:

Feld	dazugehörige Widgets	Beschreibung
StringField	StringWidget, SelectionWidget, PasswordWidget	Eine einzelne Textzeile
TextField	TextAreaWidget, RichWidget	Mehrzeiliges Textfeld, wobei das `RichWidget` den WYSIWYG-Editor einbindet
LinesField	LinesWidget, MultiSelectionWidget, InAndOutWidget	Liste von Zeichenketten, mehrzeilig
IntegerField	IntegerWidget	Ganze Zahl
FixedPointField	DecimalWidget	Dezimalzahl
BooleanField	BooleanWidget	Wahr/falsch-Checkbox
FileField	FileWidget	Feld um Dateien hochzuladen
ImageField	ImageWidget	Feld um Bilder hochzuladen
DateTimeField	CalendarWidget	Feld um ein Datum auszuwählen
ReferenceField	ReferenceWidget, InAndOutWidget	Referenz auf ein anderes Archetypes-Objekt

Werden für SelectionWidgets Vokabularien (Wertelisten) verwendet hängt die Umsetzung des Widgets von der Größe der Werteliste ab. Bei bis zu drei Werten werden Radiobuttons verwendet, ab vier Werten erhält man eine Select-Box.

Die Felder lassen sich mit beliebig vielen Eigenschaften versehen. Im Folgenden nur eine Übersicht über die gebräuchlichsten Eigenschaften:

Feldeigenschaft	Beschreibung
required	Erforderlich, die möglichen Werte sind `true` oder `false`.
searchable	Der Wert `true` schließt das Feld in die `Searchable Text`-Suche ein.
default	Bietet einen Standardwert für dieses Feld.
default_method	Name einer Methode (als Zeichenkette), die aufgerufen wird, um den Standardwert zu liefern.
schemata	Der Name eines Reiters in der Editieransicht. Das Standardschema ist `Default`. Durch den Aufruf von `finalizeATCTSchema()` werden verschiedene Änderungen am Schema vorgenommen, die das Plone-spezifische Erscheinungsbild ermöglichen.
read_permission, write_permission	Die Namen der Berechtigungen, die zum Lesen bzw. Schreiben des Feldes erforderlich sind. Die Standardwerte sind `View` bzw. `Modify portal content`.
vocabulary, vocabulary_factory, enforceVocabulary	Definieren eines Vokabulars für das Auswahlfeld, s.u.
validators	Eine Liste von Feldvalidatoren, s.u.
accessor, edit_accessor, mutator	Überschreibt die Namen der Accessor-, Edit-Accessor- oder Mutator-Methode.
widget	Widget, mit dem das Feld dargestellt werden soll
storage	Speicherabstraktion, die für dieses Feld verwendet werden soll Der Standardwert ist `AttributeStorage` **`AttributeStorage`** die Feldwerte werden in Attributen dieses Objekts gespeichert. Die Attribute haben dabei denselben Namen wie das Feld **`AnnotationStorage`** speichert die Werte in Zope3-Annotations, wodurch das Risiko von Namenskonflikten vermieden wird

Widgets

Widgets werden in `Products.Archetypes.Widget` definiert. Und ähnlich wie für Felder gibt es auch für Widgets eine Reihe von Eigenschaften, wovon die Häufigsten unten aufgeführt sind:

Widget-Eigenschaft	Beschreibung
`label`	Eine Zeichenkette oder übersetzbare Nachricht, die als Etikett des Widgets verwendet wird
`description`	Eine Zeichenkette oder übersetzbare Nachricht, die als Hilfe-Text verwendet wird
`condition`	Ein TALES-Ausfruck, die bestimmt, ob ein Widget angezeigt wird. Die Variablen `object`, `portal` und `folder` sind in diesem Kontext verfügbar
`size`	Die Länge einer Textbox oder die Höhe einer Auswahlbox
`rows`	Höhe einer Textbox
`default_output_type`	Wird von `RichWidget` verwendet um den eingegebenen Text bei der Ausgabe zu verändern **`text/x-html-safe`** nutzt Plone's HTML-Filter-Richtlinien zum Ausfiltern potentiell gefährlicher Tags **`text/html`** kann verwendet werden, wenn Sie Ihren Nutzern trauen

Vokabularien

`vocabulary` spezifisches Vokabular für ein Feld.

`enforceVocabulary` wenn der Wert `True` ist und der eingegebene Wert nicht im Vokabular vorhanden ist, gibt Archetypes einen *validation error* aus.

`vocabulary_factory` erwartet den Namen einer Zope3-`IVocabularyFactory`-Hilfsmethode. Damit kann Archetypes Zope3-Vokabularien nutzen und sie z.B. mit formlib-Formularen teilen.

Das einfachste Vokabular ist eine statische Liste von akzeptierten Werten (ganze Zahlen für das `IntegerField` und Zeichenketten für das `StringField`).

Vokabularien werden im allgemeinen zusammen mit einem `SelectionWidget`, `MultiSelectionWidget` oder `InAndOutWidget` verwendet, wobei nur die Werte des Vokabulars für die Auswahl verfügbar sind. Alternativ kann das `AddRemoveWidget`, das eine flexiblere Handhabung des Vokabulars ermöglicht und über ein separates Egg importiert werden muss. Um das Vokabular zur Verfügung zu haben importieren wir die `config.py` in der `__init__.py`:

```
...
from zope.i18nmessageid import MessageFactory
import config
...
```

Im Beispiel wollen wir für Gebrauchtwaren festhalten, wie es um die Funktionsfähigkeit bestellt ist. Für das Produkt definieren wir eine statische Liste von Tupeln, mit der die Werte „ja, nein, eingeschränkt“ zur Wahl stehen. Die Angaben lauten beispielsweise wie folgt:

```
ARTICLE_USABLE=DisplayList((
   ('yes', _(u'Yes')),
   ('no', _(u'No')),
   ('some', _(u'somewhat usable')),
   ))
```

Für die Internationalisierung des Produkts inklusive des Vokabulars ist die MessageFactory zu importieren und jeder String mit `_(u'')` zu definieren. Siehe auch Erstellen der Übersetzungsdateien.

Dieses Vokabular wird in der `content/vsresale.py` verwendet:

```
...
TypeSchema = schemata.ATContentTypeSchema.copy() + atapi.Schema((
   atapi.StringField('usable',
       vocabulary=config.ARTICLE_USABLE,
       default='yes',
       widget=atapi.SelectionWidget(label=_(u"usable"),
                                    description=(u"is the article usable?"))
       ),
```

Es kann auch gegen eine Liste von `(value, label)`-Tupeln validiert werden, wobei die Etiketten andere Angaben als die Werte des Vokabulars annehmen können. Archetypes transformiert diese Liste in `DisplayList` (s.a. `Products.Archetypes.utils`).

Für dynamische Vokabularien kann für `vocabulary` die Methode eines Objekts, eines übergeordneten Objekts oder ein Skript in einem Skin-Layer angegeben werden. Beim Aufruf erwartet Archetypes eine einfache Werteliste, eine Liste von Tupeln oder eine `DisplayList`. Hier ein Beispiel für eine solche Implementierung in ATTopic (`Products/ATContentTypes/content/topic.py`):

```
LinesField('customViewFields',
            required=False,
            mode="rw",
            default=('Title',),
            vocabulary='listMetaDataFields',
            enforceVocabulary=True,
            write_permission = ChangeTopics,
            widget=InAndOutWidget(
                    label=_(u'label_custom_view_fields', default=u'Table Columns'),
                    description=_(u'help_custom_view_fields',
                                  default=u"Select which fields to display when "
                                          "'Display as Table' is checked.")
                    ),
             ),
...
security.declareProtected(View, 'listMetaDataFields')
def listMetaDataFields(self, exclude=True):
    """Return a list of metadata fields from portal_catalog.
    """
    tool = getToolByName(self, TOOLNAME)
    return tool.getMetadataDisplay(exclude)
```

Die Methode gibt eine `DisplayList` mit Werten und Etiketten zurück.

Schlagworte in einen eigenen Index schreiben

Will man die für einen bestimmten Content-Typen vergebenen Schlagworte für alle gleichartigen Objekte desselben Typs verfügbar machen, stellt man den Vokabelbestand über eine Abfrage des `portal_catalog` bereit. Als Auswahlfeld verwenden wir das AddRemoveWidget, das über die `buildout.cfg` hinzugefügt wird:

```
eggs =
    ...
    Products.AddRemoveWidget
```

Diese Liste wird dann in der `content/vsresale.py` zusammengestellt:

```
atapi.LinesField('category',
       required=True,
       searchable=True,
       vocabulary='getTagsVocab',
       enforceVocabulary=False,
       accessor="Category"
       widget=atapi.AddRemoveWidget(label=_(u"tags"),
                                    description=_(u"")),
       ),
```

Das Vokabular entnehmen wir über über die Methode `getTagsVocab` aus dem Bestand, der in den vorhandenen Objekten desselben Content-Typs angelegt wurde. Die Methoden werden wie folgt definiert:

```
class VsResale(base.ATCTContent):
...
  def getTagsVocab(self):
       """
       Get the available tags as a DisplayList.
       """
       tags = self.getTagsInUse()
       vocab = atapi.DisplayList()
       for t in tags:
           vocab.add(t, t)
       return vocab

  def getTagsInUse(self):
       """
       Get a list of the resale tags in use in this contenttype.
       """
       catalog = getToolByName(self, 'portal_catalog')
       issues = catalog.searchResults(portal_type = 'Resale Goods Type',)
       tags = {}
       result = set()
       for i in issues:
           issue = i.getObject()
           result.update(issue.Category())
       return sorted(result)
```

Darüberhinaus bringt Plone 3 in `plone.app.vocabularies` bereits eine Reihe von häufig verwendeten Vokabularien mit.

Feld- und Objektvalidierung

Wenn im Editierformular eines Archtetypes-Artikeltyps auf *Speichern* geklickt wird, wird die Methode `validate()` von `BaseObject` aufgerufen. Alle Felder bieten einfache Validierungen wie diejenigen, ob auch ein Eintrag in einem als *erforderlich* deklarierten Feld gemacht wurde oder ein Nummernfeld keine Buchstaben enthält. Es lassen sich jedoch auch eigene Validatoren für spezifische Felder schreiben, z.B.:

```
atapi.TextField('text',
    ....
    validators=("isTidyHtmlWithCleanup",),
    ....
    ),
```

Dies weist einem Feld einen oder mehrere Validatoren zu, die in der *validator registry* registriert sein müssen.

Wie solche Validatoren registriert werden können, sehen Sie in `Products.ATContentTypes.lib.validators`:

```
validatorList.append(TidyHtmlWithCleanupValidator('isTidyHtmlWithCleanup', title='',␣
↪description=''))
```

Dies ist jedoch nur notwendig, wenn der Validator für mehrere Felder und Artikeltypen Verwendung finden soll. Wird nur ein feldspezifischer Validator benötigt, so lässt sich dieser einfach in einer Methode `validate_fieldname()` der Klasse dieses Artikeltyps hinzufügen wobei `fieldname` der Name des zu überprüfenden Feldes ist:

```
def validate_text (self, value):
    if "maybe" in value:
        return _(u"You shouldn't be so vague.")
    return None
```

Klassengenerator

```
atapi.registerType(Registrant, PROJECTNAME)
```

registriert den Artikeltyp *Registrant*, indem der Klassengenerator aufgerufen wird und jedem Feld der *Registrant*-Klasse drei Methoden hinzufügt:

Accessor *getter*-Methode

Edit accessor falls der Accessor eine Transformation vornimmt und das Editierfeld anders eingelesen werden muss.

Mutator *setter*-Methode

So werden z.B. für das Feld `email` die Methoden `getEmail()`, `getRawEmail()` und `setEmail()` als Accessor, edit accessor und mutator erzeugt.

Manchmal kann es auch notwendig werden, eigene getter- und setter-Methoden zu schreiben. Entspricht der Name der Methode derjenigen eines Accessors oder Mutators, nimmt Archetypes an, dass diese Methode anstatt einer generierten verwendet werden soll.

Der Name einer solchen Methode kann in den Feldeigenschaften angegeben werden, z.B. in `Products/Archetypes/ExtensibleMetadata.py`:

```
BooleanField(
    'allowDiscussion',
    accessor="isDiscussable",
    mutator="allowDiscussion",
    edit_accessor="editIsDiscussable",
    ...
),
```

Views und browser resources

Nachdem die Artikeltypen mit ihrem jeweiligen Schema erstellt wurden, gehen wir nun zum User-Interface über, dessen Code sich im `browser`-Paket befindet.

Icons und Stylesheet-Dateien

Für jeden Artikeltyp wird ein eigenes Icon definiert in `browser/configure.zcml`, in unserem Fall:

```
<browser:resource
   name="registration_icon.gif"
   image="registration_icon.gif"
   />
```

Dieses Icon kann referenziert werden mit `++resource++registration_icon.gif`. Um das Icon innerhalb eines Page Templates aufzurufen, können Sie folgendes angeben:

```
<img tal:attributes="src context/++resource++registration_icon.gif" />
```

Dem Icon analog lässt sich auch ein Stylesheet-Dokument hinzufügen mit:

```
<browser:resource
   name="registration.css"
   file="registration.css"
   />
```

Diese Datei können Sie mit folgendem Code in ein Page Template einfügen:

```
<metal:css fill-slot="css_slot">
    <style type="text/css" media="all"
           tal:content="string: @import url(${context/++resource++registration.css});
↪"></style>
</metal:css>
```

Views

Auch die Views werden in `browser/configure.zcml` registriert:

```
<browser:page
    for="..interfaces.IRegistration"
    name="view"
    class=".registration.RegistrationView"
    permission="zope2.View"
    />
```

Üblicherweise wird die Standardansicht eines Artikeltyps mit `@@view` aufgerufen.

Sollen Autoren zwischen verschiedenen Ansichten eines Artikeltyps in Plones Ansicht-Menü wählen können, müssen diese Ansichten einerseits in einer Liste im GenericSetup-Profil angegeben werden, andererseits jedoch auch in `browser/configure.zcml` registriert werden:

```
<include package="plone.app.contentmenu" />
...
<browser:menuItem
    for="..interfaces.IRegistration"
    menu="plone_displayviews"
    title="Registration view"
    action="@@view"
    description="Default view of a registration"
    />
```

`action` verweist auf den Namen der Ansicht, wobei der Menüeintrag nur für das `IRegistration`-Interface angezeigt wird.

Die View-Klasse selbst enthält die Methoden `name` und `details`:

```
from Products.Five import BrowserView

class RegistrationView(BrowserView):
   """A view of a Registration object"""
   def name(self):
       return self.context.Title()

   def details(self):
       return self.context.Description()
```

Der Decorator `@memoize` stellt sicher, dass der Aufruf in einer Instanz nur einmal ausgeführt wird – und der zurückgegebene Wert gespeichert wird. Wenn Templates eine Methode mehrfach aufrufen, kann so die Performance deutlich gesteigert werden (s.a. Memoize). Das Template `registration.pt` sieht dann so aus:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      xmlns:i18n="http://xml.zope.org/namespaces/i18n"
      lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="vs.registration">
<body>

<metal:main fill-slot="main">
    <tal:main-macro metal:define-macro="main"
           tal:define="text context/text;">

        <div tal:replace="structure provider:plone.abovecontenttitle" />

        <h1 class="documentFirstHeading">
            <span metal:use-macro="python:context.widget('title', mode='view')" />
        </h1>

        <div tal:replace="structure provider:plone.belowcontenttitle" />

        <div class="documentDescription">
            <span metal:use-macro="python:context.widget('description', mode='view')"␣
↪/>
        </div>

        <div tal:replace="structure provider:plone.abovecontentbody" />

        <p tal:condition="python: not text and is_editable"
           i18n:translate="no_body_text"
           class="discreet">
            This item does not have any body text, click the edit tab to change it.
        </p>

        <div tal:condition="text" metal:use-macro="python:context.widget('text', mode=
↪'view')" />

        <form action="createObject">
            <input name="type_name"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                    type="hidden"
                    value="Registrant"
                    />
            <input class="standalone"
                    value="Registration"
                    type="submit"
                    i18n:attributes="value"
                    />
        </form>

        <tal:registrants condition="view/have_registrants">
            <h2 i18n:translate="title_registration_contents">Registrants</h2>
            <dl>
                <tal:block repeat="registrant view/registrants">
                    <dt>
                        <a tal:attributes="href registrant/url"
                           tal:content="registrant/title" />
                    </dt>
                    <dd tal:content="registrant/address" />
                </tal:block>
            </dl>
        </tal:registrants>

        <div metal:use-macro="context/document_relateditems/macros/relatedItems">
            show related items if they exist
        </div>

        <div tal:replace="structure provider:plone.belowcontentbody" />

    </tal:main-macro>
</metal:main>

</body>
</html>
```

Das Template entspricht weitgehend Plones `document_view.pt`.

Beachten Sie, dass verschiedene Viewlet-Manager angegeben wurden, wie z.B.:

```
<div tal:replace="structure provider:plone.abovecontenttitle" />
```

Im Kapitel Viewlets wird ausführlich auf diese Zope3-Komponenten eingegangen.

Content-Menü

Üblicherweise wird das Content-Menü mit den Menüs *Darstellung, *Hinzufügen* und *Workflow* in Views nicht angezeigt. Falls es dennoch angezeigt werden soll, müssen Sie das `IViewView`-Interface aus `plone.app.layout` erhalten:

```
from zope.interface import implements
from Products.Five.browser import BrowserView
from plone.app.layout.globals.interfaces import IViewView

class MyView(BrowserView):
   implements(IViewView)
```

Inline Editing

In diesem Template wurde auch das Inline-Editing mit Kinetic Style Sheets ermöglicht, z.B. mit:

```
<span metal:use-macro="python:context.widget('title', mode='view')" />
```

Soll ein Feld nicht direkt editiert werden können, genügt ein einfacheres Konstrukt:

```
<span tal:content="context/title" />
```

Mehr über das JavaScript-Framework erfahren Sie im Kapitel *Kinetic Style Sheets*.

Installation und Registrierung

Nachdem die Artikeltypen und die zugehörigen Views erstellt sind, müssen diese Artikeltypen noch installiert werden. Dies geschieht mit einem Generic Setup Extension-Profil in `vs/registration/profiles/default/types.xml`:

```
<object name="portal_types" meta_type="Plone Types Tool">
    <object name="Registration"
            meta_type="Factory-based Type Information with dynamic views"/>
    <object name="Registrant"
            meta_type="Factory-based Type Information with dynamic views"/>
</object>
```

Damit werden zwei verschiedene *factory-based type information* (FTI)-Objekte im `portal_types`-Tool erstellt, wobei die *Dynamic Views* Editoren das Auswählen verschiedener Ansichten im *Darstellung*-Menü von Plone erlauben.

Jede FTI ist detaillierter in der entsprechenden Datei in `profiles/default/types/` definiert. Dabei muss der Dateiname dem Namen des *portal type* entsprechen, wobei Leerzeichen als Unterstriche angegeben werden müssen. Schauen wir uns nun `profiles/default/types/Registration.xml` genauer an:

```
<?xml version="1.0"?>
<object name="Registration"
        meta_type="Factory-based Type Information with dynamic views"
        i18n:domain="vs.registration"
        xmlns:i18n="http://xml.zope.org/namespaces/i18n">
    <property name="title"
              i18n:translate="">Registration</property>
    <property name="description"
              i18n:translate="">A folder which can contain registrants.</property>
    <property name="content_icon">++resource++registration_icon.gif</property>
```

In diesen Zeilen wird dem Artikeltyp Name, Beschreibung und Icon zugewiesen, die Plone auch zur Darstellung nutzt.

```
<property name="content_meta_type">Registration</property>
<property name="product">vs.registration</property>
<property name="factory">addRegistration</property>
<property name="immediate_view">atct_edit</property>
```

Hier wird der Meta-Typ des Artikeltyps angegeben, der meist dem Namen des *portal type* entspricht. Anschließend wird die Methode angegeben, die das neue Artikelobjekt erstellt und initialisiert. `immediate_view` gibt die Ansicht, die unmittelbar nach dem Erstellen des Objekts gezeigt wird, an (auch wenn diese momentan nicht durch Plone unterstützt wird).

```
<property name="global_allow">True</property>
<property name="filter_content_types">True</property>
<property name="allowed_content_types">
    <element value="Registrant" />
</property>
```

Diese Eigenschaften bestimmen das Verhältnis von übergeordneten zu untergeordneten Objekten. `Registration` darf in allen Ordner-Objekten, für die `filter_content_types` auf `False` gesetzt wurde, hinzugefügt werden. Darüberhinaus darf in `Registration` mit `filter_content_types` und `allowed_content_types` nur der Artikeltyp `Registrant` hinzugefügt werden.

```
<property name="allow_discussion">False</property>
```

Diese Eigenschaft bestimmt, ob für diesen Artikeltyp üblicherweise Diskussionen erlaubt sind oder nicht.

```
<property name="default_view">view</property>
<property name="view_methods">
    <element value="view"/>
    <element value="folder_summary_view"/>
    <element value="folder_tabular_view"/>
    <element value="folder_listing"/>
</property>
```

Diese Eigenschaften geben die Standardansicht und die im *Darstellung*-Menü auswählbaren Ansichten (*dynamic views*) an.

```
<alias from="(Default)" to="(dynamic view)"/>
<alias from="edit" to="atct_edit"/>
<alias from="sharing" to="@@sharing"/>
<alias from="view" to="(selected layout)"/>
```

Üblicherweise nutzen die meisten Plone-Artikeltypen diese vier Aliase. Werden keine *dynamic views* FTIs verwendet, müssen die Namen der Views oder Templates für die `(Default)`- und `view`-Aliase angegeben werden.

```
<action title="View"
        action_id="view"
        category="object"
        condition_expr=""
        url_expr="string:${object_url}/"
        visible="True">
    <permission value="View"/>
</action>
<action title="Edit"
        action_id="edit"
        category="object"
        condition_expr=""
        url_expr="string:${object_url}/edit"
        visible="True">
    <permission value="Modify portal content"/>
</action>
```

Schließlich werden noch zwei typspezifische Aktionen definiert, die der Kategorie `object` zugeordnet werden und die oben definierten Aliase nutzen. Beachten Sie, dass bei *folderish*-Artikeltypen für die `view`-Aktion `string:${folder_url}/`, hingegen für *non-folderish*-Artikeltypen `string:${object_url}` verwendet wird.

Initialisierung und Hinzufügen-Rechte

Die oben bereits erwähnte Methode `addRegistration` zum Erstellen eines neuen Objekts des Artikeltyps *Registration* wird automatisch durch Archetypes erstellt, wenn das Produkt initiiert wird. Darüberhinaus werden in der Datei `__init__.py` auch die Rechte zum Hinzufügen der neuen Artikeltypen gesetzt.

```
from vs.registration import config
from Products.Archetypes import atapi
from Products.CMFCore import utils

def initialize(context):
```

Damit wird das Zope-2-Produkt initialisiert.

```
from content import registration, registrant

content_types, constructors, ftis = atapi.process_types(
    atapi.listTypes(config.PROJECTNAME),
    config.PROJECTNAME)
```

Die Artikeltypen aus `content` werden importiert und mit Archetypes `registerType()`-Aufruf registriert.

```
for atype, constructor in zip(content_types, constructors):
    utils.ContentInit("%s: %s" % (config.PROJECTNAME, atype.portal_type),
        content_types      = (atype,),
        permission         = config.ADD_PERMISSIONS[atype.portal_type],
        extra_constructors = (constructor,),
        ).initialize(context)
```

Nun werden die Artikeltypen mit den angegebenen Rechten zum Hinzufügen registriert. Diese Rechte sind in der `config.py`-Datei angegeben:

```
ADD_PERMISSIONS = {
    "Registration" : "vs: Add Registration",
    "Registrant"   : "vs: Add Registrant",
}
```

Damit werden die zwei Artikeltypen entsprechenden Rechten zum Hinzufügen zugeordnet. Zudem sind diese Rechte auch noch definiert in `content/configure.zcml`. Die Rechte erhalten mit `vs` ein Präfix, damit Sie im *Security*-Reiter des ZMI zusammen dargestellt werden.

Die den Hinzufügen-Rechten zugehörigen Rollen sind in `profiles/default/rolemap.xml` definiert:

```
<rolemap>
    <permissions>
        <permission name="vs: Add Registration" acquire="False">
            <role name="Manager" />
        </permission>
        <permission name="vs: Add Registrant" acquire="False">
            <role name="Manager" />
            <role name="Owner" />
            <role name="Contributor" />
        </permission>
    </permissions>
</rolemap>
```

Registrierung der Artikeltypen am *Factory Tool*

Für die meisten Artikeltypen empfiehlt sich die Registrierung am *Factory Tool*, um halbfertige Objekte beim Erstellen zu vermeiden. Die Registrierung geschieht auch hier mit einem Generic-Setup-Profil, nämlich `factorytool.xml`:

```
<object name="portal_factory"
        meta_type="Plone Factory Tool">
    <factorytypes>
        <type portal_type="Registration"/>
        <type portal_type="Registrant"/>
    </factorytypes>
</object>
```

Registrierung der Artikeltypen am *CMFEditions Repository*

Die ATCT-Artikeltypen, mit denen Plone ausgeliefert wird, werden alle automatisch am *CMFEditions Repository* angemeldet, sodass für die entsprechenden Objekte auch die früheren Versionen angezeigt werden können. Um nun unsere beiden Artikeltypen am *CMFEditions Repository* zu registrieren, schreiben wir die Methode `setupEditions`:

```
from StringIO import StringIO
from logging import getLogger

from Products.CMFCore.utils import getToolByName
from Products.Archetypes import atapi

from config import PROJECTNAME

class Generator:

    def setupEditions(self, p, out):


        content_types, constructors, ftis = atapi.process_types(
            atapi.listTypes(PROJECTNAME),
            PROJECTNAME)
        portal_repository = getToolByName(p, 'portal_repository')
        types = portal_repository.getVersionableContentTypes()

        for type in content_types:
            if type.portal_type not in types:
                types.append("%s" %type.portal_type)

        portal_repository.setVersionableContentTypes(types)
        print >> out, " Editions enabled %s \n" % types
```

Diese Methode wird in der `setuphandlers.py`-Datei erstellt und mit `setupVarious` aufgerufen:

```
def setupVarious(context):

    if context.readDataFile('vs.registration_various.txt') is None:
        return
    # Add additional setup code here
    out = StringIO()
    site = context.getSite()
    gen = Generator()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    gen.setupEditions(site, out)
    logger = context.getLogger(PROJECTNAME)
    logger.info(out.getvalue())
```

Entsprechend erstellen wir die Konfigurationsdatei `vs.registration/vs/registration/profiles/default/import_steps.xml` mit folgendem Inhalt:

```
<?xml version="1.0"?>
<import-steps>
    <import-step id="idg.org.various"
                 version="20080617-01"
                 handler="vs.registration.setuphandlers.setupVarious"
                 title="vs.registration: miscellaneous import steps">
        <dependency step="typeinfo" />
            Various import steps that are not handled by GS import/export
            handlers.
    </import-step>
</import-steps>
```

Und schließlich benötigen wir noch die Datei `vs.registration/vs/registration/profiles/default/vs.registration_various.txt`, damit unsere `setupVarious`-Methode auch ausgeführt wird.

Installation und Konfiguration im Policy-Produkt

Schließlich soll das `vs.policy`-Produkt noch so geändert werden, dass es automatisch `vs.registration` bei der Installation in der Plone-Site mitinstalliert. Hierzu wird zunächst in `src/vs.policy/vs/policy/configure.zcml` folgende Zeile eingefügt:

```
<include package="vs.registration" />
```

Für Plone 3.0 wird anschließend `vs.registration` auch noch in die benötigten Produkte in `src/vs.policy/vs/policy/Extensions/Install.py` eingetragen:

```
PRODUCT_DEPENDENCIES = ('vs.theme',
                        'vs.registration')
```

Für Plone 3.1 wird `src/vs.policy/vs/policy/profiles/default/metadata.xml` um folgende Zeile ergänzt:

```
<dependency>profile-vs.registration:default</dependency>
```

ATContenttypes-Konfiguration

Die ATContenttypes liefern in `Products/ATContentTypes/etc/atcontenttypes.conf.in` die Vorlage für eine Konfigurationsdatei. Um diese nun anpassen zu können, kann die Buildout-Konfiguration folgendermaßen geändert werden:

```
[buildout]
...
parts =
    ...
    atct_conf
...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[atct_conf]
recipe = plone.recipe.command
target = ${instance:location}/etc/atcontenttypes.conf
command = ln -s ${buildout:directory}/etc/atcontenttypes.conf ${:target}
```

Schauen wir uns nun `etc/atcontenttypes.conf` genauer an.

mxtidy HTML-Filter-Optionen:

drop_font_tags Sofern auf `yes` gesetzt, verwirft Tidy alle `font`- und `center`-Tags.

drop_empty_paras Sofern auf `yes` gesetzt, werden leere Paragraphen verworfen. Bei `no` werden leere Absätze durch zwei `br`-Elemente ersetzt.

input_xml Sofern auf `yes` gesetzt, verwendet Tidy den XML-Parser und nicht den HTML-Parser.

output_xhtml Sofern auf `yes` gesetzt, generiert Tidy wohlgeformtes HTMLund der Doctpe wird in XHTML geändert.

quiet Sofern auf `yes` gesetzt, gibt Tidy nicht die Anzahl der Fehler und Warnungen aus.

show_warnings Sofern auf ``no``gesetzt, werden Warnungen unterdrückt.

indent_spaces Setzt die Anzahl der Leerzeichen zum Einrücken der Inhalte.

word_2000 Sofern auf `yes` gesetzt, filtert Tidy zusätzliche Anweisungen aus, die Microsoft Word 2000 beim Speichern als *Web pages* einfügt.

wrap Setzt die Anzahl der Zeichen, nach denen spätestens umbrochen wird.

Falls kein automatischer Zeilenumbruch gewünscht ist, kann hier einfach `0` angegeben werden.

tab_size Setzt die Anzahl der Leerzeichen bei der Eingabe eines Tabulators.

char_encoding Bestimmt die Interpretation von Zeichen.

Bei `ascii` akzeptiert Tidy *Latin-1*-Zeichen und Entitäten für alle Zeichen `> 127`.

Eine vollständige Liste der Optionen finden Sie in HTML Tidy Options.

feature swallowImageResizeException Sofern `enable yes` gesetzt ist, werden Fehlermeldungen beim Ändern der Bildgröße nicht ausgegeben.

pil_config Konfiguration der Python Imaging Library (PIL).

quality Qualität, mit der die Bilder berechnet werden. Die Skala reicht von `1` bis `100`. Bei `100` wird jedoch keine JPEG Quantisierung mehr verwendet.

resize_algo Algorithmus, mit dem die Größenänderungen von Bildern berechnet werden.

Mögliche Angaben sind:

- `nearest`
- `bilinear`
- `bicubic`
- `antialias`

archetype Konfiguration der Archetypes-Artikeltypen

contenttypes MIME-Type des ATContenttype.

default Standardwert des MIME-Type

allowed Zulässige MIME-Types

max_file_size Maximale Dateigröße in `byte`, `kb` oder `mb`.

`0` begrenzt die Dateigröße nicht.

max_image_dimension Maximale Breite und Höhe von Bildern.

`0,0` begrenzt weder die Breite noch die Höhe der Originalbilder.

allow_document_upload `yes` erlaubt das Hochladen von Artikeln.

metadate Ein einzelnes Metadata-Element.

Folgende Attribute sind möglich:

- `name`
- `friendlyName`
- `description`
- `enabled`

index Ein einzelnes Metadata-Element.

Folgende Attribute sind möglich:

- `name`
- `friendlyName`
- `description`
- `enabled`
- `criterion`

topic_tool Standardkonfiguration des Topic-Tools.

atct_tool Standardkonfiguration des ATCT-Tools.

ZopeSkel-archetype-Template

Mit dem archetype-Template und dessen local commands lassen sich Artikeltypen noch einfacher erstellen.

Statt des Templates `plone` kann auch das Template `archetype` mit *local commands* verwendet werden um neue Artikeltypen zu erstellen. Hierzu geben wir zunächst folgendes an:

```
paster create -t archetype vs.registration
```

Anschließend wechseln wir in das gerade eben erstelle Verzeichnis und lassen uns die hier verfügbaren *local commands* anzeigen:

```
$ cd vs.registration
$ paster addcontent --list
Available templates:
   atschema:     A handy AT schema builder
   contenttype:  A content type skeleton
   form:         A form skeleton
   formfield:    Schema field for a form
   i18nlocale:   An i18n locale directory structure
   portlet:      A Plone 3 portlet
   view:         A browser view skeleton
   zcmlmeta:     A ZCML meta directive skeleton
```

Weitere *local commands* sind für das `plone_pas`-Projekt verfügbar. Eine Übersicht über alle verfügbaren **local commands** erhält man mit:

```
$ paster addcontent -a
  N anonymous_user_factory_plugin:  A Plone PAS AnonymousUserFactory Plugin
    atschema:                       A handy AT schema builder
  N authentication_plugin:          A Plone PAS Authentication Plugin
  N challenge_plugin:               A Plone PAS Challenge Plugin
    contenttype:                    A content type skeleton
  N credentials_reset_plugin:       A Plone PAS CredentialsReset Plugin
  N extraction_plugin:              A Plone PAS Extraction Plugin
    form:                           A form skeleton
    formfield:                      Schema field for a form
  N group_enumeration_plugin:       A Plone PAS GroupEnumeration Plugin
  N groups_plugin:                  A Plone PAS Groups Plugin
    i18nlocale:                     An i18n locale directory structure
    portlet:                        A Plone 3 portlet
  N properties_plugin:              A Plone PAS Properties Plugin
  N role_assigner_plugin:           A Plone PAS RoleAssigner Plugin
  N role_enumeration_plugin:        A Plone PAS RoleEnumeration Plugin
  N roles_plugin:                   A Plone PAS Roles Plugin
  N update_plugin:                  A Plone PAS Update Plugin
  N user_adder_plugin:              A Plone PAS UserAdder Plugin
  N user_enumeration_plugin:        A Plone PAS UserEnumeration Plugin
  N user_factory_plugin:            A Plone PAS UserFactory Plugin
  N validation_plugin:              A Plone PAS Validation Plugin
    view:                           A browser view skeleton
    zcmlmeta:                       A ZCML meta directive skeleton
```

Die beim Anlegen des Projekts gemachten Angaben finden mit Erläuterungen in der `setup.py`.

Mit den Angaben in der Datei `vs.registration/vs/registration/configure.zcml` wird das Produkt für das Generic Setup-Tool registriert, wenn hierfür der Default-Wert auf „True“ belassen wurde. Titel und Beschreibungstext werden für die Anzeige im Generic Setup-Tool verwendet:

```
<configure
    ...
    i18n_domain="vs.registration">

   <genericsetup:registerProfile
       name="default"
       title="Registration Content-types"
       directory="profiles/default"
       description="Short description of registration content-types"
       provides="Products.GenericSetup.interfaces.EXTENSION"
       />
```

Content-Types Registration und Registrant

Um einen Content-Type zu unserem Projekt hinzuzufügen kann folgendes ausgeführt werden:

```
$ cd vs.registration
$ paster addcontent contenttype
Enter contenttype_name (Content type name ) ['Example Type']: Registration
Enter contenttype_description (Content type description ) ['Description of the␣
↪Example Type']: Container for registrants
Enter folderish (True/False: Content type is Folderish ) [False]: True
Enter global_allow (True/False: Globally addable ) [True]:
Enter allow_discussion (True/False: Allow discussion ) [False]:
 Inserting from README.txt_insert into /home/veit/Projects/IDG/idg_buildout/src/vs.
↪registration/vs/registration/README.txt
 Inserting from config.py_insert into /home/veit/Projects/IDG/idg_buildout/src/vs.
↪registration/vs/registration/config.py
 Recursing into content
   Copying +content_class_filename+.py_tmpl to /home/veit/Projects/IDG/idg_buildout/
↪src/vs.registration/vs/registration/content/registration.py
   Inserting from configure.zcml_insert into /home/veit/Projects/IDG/idg_buildout/src/
↪vs.registration/vs/registration/content/configure.zcml
 Recursing into interfaces
   Copying +content_class_filename+.py_tmpl to /home/veit/Projects/IDG/idg_buildout/
↪src/vs.registration/vs/registration/interfaces/registration.py
   Inserting from __init__.py_insert into /home/veit/Projects/IDG/idg_buildout/src/vs.
↪registration/vs/registration/interfaces/__init__.py
 Recursing into profiles
   Recursing into default
     Inserting from factorytool.xml_insert into /home/veit/Projects/IDG/idg_buildout/
↪src/vs.registration/vs/registration/profiles/default/factorytool.xml
     Copying rolemap.xml_insert to /home/veit/Projects/IDG/idg_buildout/src/vs.
↪registration/vs/registration/profiles/default/rolemap.xml
     Recursing into types
       Creating /home/veit/Projects/IDG/idg_buildout/src/vs.registration/vs/
↪registration/profiles/default/types/
       Copying +types_xml_filename+.xml_tmpl to /home/veit/Projects/IDG/idg_buildout/
↪src/vs.registration/vs/registration/profiles/default/types/Registration.xml
     Inserting from types.xml_insert into /home/veit/Projects/IDG/idg_buildout/src/vs.
↪registration/vs/registration/profiles/default/types.xml
```

i18n-Layer

Damit der Content-Type vollständig internationalisierbar angelegt wird fügen wir noch die Angaben für die Internationalisierung `i18nlocale` hinzu:

```
$ cd vs.registration
$ paster addcontent i18nlocale
Enter language_code (The iso-code of the language) ['']: de
 Inserting from configure.zcml_insert into /home/veit/Projects/IDG/idg_buildout/src/
↪vs.registration/vs/registration/configure.zcml
 Recursing into locales
   Creating /home/veit/Projects/IDG/idg_buildout/src/vs.registration/vs/registration/
↪locales/
   Recursing into +language_iso_code+
     Creating /home/veit/Projects/IDG/idg_buildout/src/vs.registration/vs/
↪registration/locales/de/
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    Recursing into LC_MESSAGES
      Creating /home/veit/Projects/IDG/idg_buildout/src/vs.registration/vs/
↪registration/locales/de/LC_MESSAGES/
      Copying README.txt to /home/veit/Projects/IDG/idg_buildout/src/vs.registration/
↪vs/registration/locales/de/LC_MESSAGES/README.txt
```

Jetzt sollte die Instanz ohne Fehlermeldung gestartet werden können. Unser Content-Type ist beim Hinzufügen einer Plone-Site als „Extension Profile“ mit dem vergebenen Titel „Registration Content-types“ auswählbar.

Die Datei `vs/registration/profiles/default/types.xml` registriert neue Content-Typen am Types-Tool:

```
<object name="portal_types" meta_type="Plone Types Tool">
  <object name="Registration"
          meta_type="Factory-based Type Information with dynamic views"/>
  <object name="Registrant"
          meta_type="Factory-based Type Information with dynamic views"/>
</object>
```

In der neu erstellten Plone-Site lässt sich der Content-Typ global hinzufügen, soweit bei der Erstellung nichts anderes angegeben wurde.

Verwenden der Portal Factory

Bei Content-Typen, die auf Archetypes oder CMF beruhen, wird das Objekt bereits angelegt bevor das zugehörige Formular ausgefüllt wird. Mit den Angaben in `vs/registration/profiles/default/factorytool.xml` wird das Produkt am Factory-Tool angemeldet. Dieses verwaltet neue Objekte nur solange, bis das Formular tatsächlich gespeichert wird und verschiebt das temporäre Objekt beim Speichern an die vorgesehene Stelle. Damit der Mechanismus für unseren Objekte greift sind diese Angaben hinzuzufügen:

```
 <factorytypes>
  ...
      <type portal_type="Registration"/>
      <type portal_type="Registrant"/>
</factorytypes>
```

Struktur der Objekte

Bisher stehen beiden Content-Typen lediglich die Strukturen zur Verfügung, die dem Default von `ATContentType` entsprechen, nämlich die Felder `title` und `description`. Wir ergänzen im folgenden die übernommenen Felder um spezifische Informationen.

Um z.B. die Registrierung mit einen formatierbaren Textfeld zu versehen fügen wir eine entsprechende Definition in `content/registration.py` hinzu. Zunächst wird das Schema von einem Basistypen, in diesem Fall ATFolder-TypeSchema, kopiert und anschließend das eigene Schema angehängt:

```
RegistrationSchema = folder.ATFolderSchema.copy() + atapi.Schema((

  atapi.TextField('text',
      required=True,
      searchable=True,
      storage=atapi.AnnotationStorage(),
      validators=('isTidyHtmlWithCleanup',),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      default_output_type='text/x-html-safe',
      widget=atapi.RichWidget(label=_(u"Body Text"),
                              description=_(u"Text for front page of registration"),
                              rows=25,
                              allow_file_upload=False),
      ),
  ))

RegistrationSchema['title'].storage = atapi.AnnotationStorage()
RegistrationSchema['description'].storage = atapi.AnnotationStorage()
RegistrationSchema['text'].storage = atapi.AnnotationStorage()

schemata.finalizeATCTSchema(
  RegistrationSchema,
  folderish=True,
  moveDiscussion=False
)
...
title = atapi.ATFieldProperty('title')
description = atapi.ATFieldProperty('description')
text = atapi.ATFieldProperty('text')
```

Das zusätzliche Feld erhält erhält den Status Pflichtfeld, wird durchsuchbar und wird auf valides HTML validiert. Der Text kann formatiert werden. Alle Felder verwenden explizit den Speichertyp `AnnotationStorage`.

Um z.B. das Teilnehmer-Objekt um ein Feld für eine Email-Adresse zu ergänzen fügen wir die Angaben für das Feld in `content/registrant.py` hinzu. Zunächst wird wieder das Schema von einem Basistypen, in diesem Fall ATContentTypeSchema, kopiert und anschließend das eigene Schema angehängt:

```
RegistrantSchema = schemata.ATContentTypeSchema.copy() + atapi.Schema((

    atapi.StringField('email',
        required=True,
        searchable=True,
        storage=atapi.AnnotationStorage(),
        widget=atapi.StringWidget(label=_(u"Email"),
                                  description=_(u""))
        ),

    ))

RegistrantSchema['title'].storage = atapi.AnnotationStorage()
RegistrantSchema['title'].widget.label = _(u"Registrant name")
RegistrantSchema['title'].widget.description = _(u"")

RegistrantSchema['description'].storage = atapi.AnnotationStorage()
RegistrantSchema['description'].widget.label = _(u"Address")
RegistrantSchema['description'].widget.description = _("")

RegistrantSchema['email'].storage = atapi.AnnotationStorage()
RegistrantSchema['email'].widget.label = _(u"Email")
RegistrantSchema['email'].widget.description = _(u"")

schemata.finalizeATCTSchema(RegistrantSchema, moveDiscussion=False)

class Registrant(base.ATCTContent):
    """Describe a registrant.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    implements(IRegistrant)

    meta_type = "Registrant"
    _at_rename_after_creation = True
    schema = RegistrantSchema

    name = atapi.ATFieldProperty('title')
    address = atapi.ATFieldProperty('description')
    email = atapi.ATFieldProperty('email')

atapi.registerType(Registrant, PROJECTNAME)
```

Dieses Schema implementiert ein Interface, `IRegistrant`. Die Definition steht im Unterordner `interfaces`. Die Klasse `Registrant` erweitert `base.ATCTContent`. Die Angabe `at_rename_after_creation` benennt Objekte in die normalisierte Version ihres Titels um.

`meta_type` setzt den eindeutigen Namen des Artikeltyps. Alternativ kann dieser auch mit `portal_type` gesetzt werden. Zur Unterscheidung siehe unten.

Um nun z.B. ein Portlet unserem Projekt hinzuzufügen, kann einfach folgendes angegeben werden:

```
$ paster addcontent portlet
Enter portlet_name (Portlet name (human readable)) ['Example portlet']: My portlet
Enter portlet_type_name (Portlet type name (should not contain spaces)) [
↪'ExamplePortlet']: registrants
Enter description (Portlet description) ['']: My portlet
  Recursing into portlets
    Copying +portlet_filename+.pt_tmpl to /home/veit/vs.mytype/vs/mytype/portlets/
↪registrants.pt
    Copying +portlet_filename+.py_tmpl to /home/veit/vs.mytype/vs/mytype/portlets/
↪registrants.py
File '__init__.py' already exists: skipped
    Inserting from configure.zcml_insert into /home/veit/vs.mytype/vs/mytype/portlets/
↪configure.zcml
  Recursing into profiles
    Recursing into default
      Inserting from portlets.xml_insert into /home/veit/vs.mytype/vs/mytype/profiles/
↪default/portlets.xml
  Recursing into tests
File '__init__.py' already exists: skipped
    Copying base_+portlet_filename+.py_tmpl to /home/veit/vs.mytype/vs/mytype/tests/
↪base_registrants.py
    Copying test_+portlet_filename+.py_tmpl to /home/veit/vs.mytype/vs/mytype/tests/
↪test_registrants.py
```

Neben den neu erstellten Dateien `portlets/registrants.py` und `portlets/registrants.pt` wurden auch die Dateien `portlets/configure.zcml` und `profiles/default/portlets.xml` aktualisiert.

Gestaltung

Um z.B. den Ordnern für Registrierungen ein eigenes Icon zu geben kann man einfach diese Angaben in `browser/configure.zcml` hinzufügen:

```
<browser:resource
   name="registration_icon.gif"
   image="registration_icon.gif"
   />
```

Die zugehörige Klasse `class RegistrantView(BrowserView)` wird in `browser/registrant.py` und `browser/registration.py` definiert.

Mit Hilfe des View-Templates lässt sich über Paster auf einfache Art ein neuer Browser View hinzufügen:

```
paster addcontent view
```

Sofern noch nicht vorhanden werden die Dateien `browser/registrantview.py` und `browser/registrationview.py` sowie `browser/registrantview.pt` und `browser/registrationview.pt` erstellt. Die Templates können angepasst werden.

Damit das Icon vom Factory Tool verwendet wird ändert man das default-Icon in `profiles/default/types/Registration.xml`:

```
<object name="Registration"
  ...
  <property name="content_icon">++resource++registration_icon.gif</property>
```

Verwendung einschränken

Der Registrierungs-Ordner ist als Containerobjekt für Teilnehmer vorgesehen. Außerhalb dieser Ordner sollen sich auf der Site keine Teilnehmer hinfügen lassen. Diese Einschränkung wird in der `profiles/default/types/Registration.xml` getroffen, und die Eigenschaft `filter_content_types` auf `True` gesetzt:

```
<object name="Registration"
  ...
  <property name="filter_content_types">True</property>
  <property name="allowed_content_types">
      <element value="Registrant" />
  </property>
```

Damit die Einschränkung übernommen wird und nur Teilnehmer einem Ordner „Registration“ zugeordnet werden können muss bei einer bestehenden Site das Generic Setup-Profil aktualisiert werden. Gehen Sie hierzu in das ZMI der Site und dort ins Types-Tool, Reiter „Import“, und wählen sie das Profil „vs.registration“, und setzen das Häkchen bei „Types Tool - Import types tool's type information objects“. Vor dem tatsächlich Import sollte noch „include dependencies“ abgewählt werden. Nach dem Import sollten sich in Registration-Ordnern nur noch Teilnehmer hinzufügen lassen.

Rollen und Berechtigungen

Werden keine Änderungen an den voreingestellten Berechtigungen vorgenommen dürfen Manager und Redakteure Registrierungs-Container und Teilnehmer anlegen. Wir ändern die `profiles/default/rolemap.xml`, so dass nur Manager neue Container anlegen können:

```
<?xml version="1.0"?>
<rolemap>
   <permissions>
       <permission name="vs.registration: Add Registration" acquire="False">
           <role name="Manager" />
       </permission>
       <permission name="vs.registration: Add Registrant" acquire="False">
           <role name="Manager" />
           <role name="Owner" />
           <role name="Contributor" />
       </permission>
   </permissions>
</rolemap>
```

Um das Hinzufügen von Content über den ZMI-Reiter `security` festlegen zu können wird `content/configure.zcml` mit folgenden Angaben versehen:

```
<configure
      ...
      i18n_domain="vs.registration">

      <class class=".registration.Registration">
      <require
          permission="zope2.View"
          interface="..interfaces.IRegistration"
          />
  </class>

  <class class=".registrant.Registrant">
      <require
          permission="zope2.View"
          interface="..interfaces.IRegistrant"
          />
  </class>
...
</configure>
```

Nähere Erläuterungen der Berechtigungen finden Sie im Abschnitt Sicherheit und Arbeitsabläufe.

In der Datei `profiles/default/Registration.xml` z.B. werden Eigenschaften des Objekts definiert, die sich auf die Sichtbarkeit beziehen. Die Aufrufe für Ansehen und Bearbeiten erfordern unterschiedliche Rechte:

```
<action title="View"
          action_id="view"
          category="object"
          condition_expr=""
          url_expr="string:${folder_url}/"
          visible="True">
      <permission value="View"/>
  </action>
  <action title="Edit"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
          action_id="edit"
          category="object"
          condition_expr=""
          url_expr="string:${object_url}/edit"
          visible="True">
      <permission value="Modify portal content"/>
  </action>
```

Dateisystemansicht des Produkts

Das fertige Produkt sollte im Dateisystem in etwa so aussehen:

```
|-- README.txt
|-- __init__.py
|-- config.py
|-- configure.zcml
|-- browser
|   |-- __init__.py
|   |-- configure.zcml
|   |-- registrant_icon.gif
|   |-- registrantview.pt
|   |-- registrantview.py
|   |-- registration_icon.gif
|   |-- registrationview.pt
|   |-- registrationview.py
|-- content
|   |-- __init__.py
|   |-- configure.zcml
|   |-- registrant.py
|   |-- registration.py
|-- interfaces
|   |-- __init__.py
|   |-- registrant.py
|   |-- registration.py
|-- locales
|   `-- de
|       `-- LC_MESSAGES
|-- portlets
|   |-- __init__.py
|   `-- configure.zcml
|-- profiles
|     `-- default
|         |-- factorytool.xml
|         |-- metadata.xml
|         |-- portlets.xml
|         |-- rolemap.xml
|         |-- types
|         |-- types.xml
|         |-- workflows
|         `-- workflows.xml
`-- tests
    |-- __init__.py
    |-- base.py
    `-- test_doctest.py
```

Die Dateien und Ordner werden im folgenden kurz erläutert, die Angaben sind relativ zu `/src/vs`.

registration/vs/registration, außer wo abweichend angegeben:

```
- ``/__init__.py``: Das Initialisierungsmodul.
- ``/configure.zcml``: beschreibt Konfigurationsangaben, meldet das Produkt am␣
↪Generic Setup-Tool an und enthält weitere Verweise auf Sub-packages (=Unterordner),␣
↪die ihrerseits Konfigurationsangaben enthalten. Hier werden auch die Übersetzungen␣
↪des i18n-Layers registriert.
- ``/config.py``: Fügt den Content-Typen die Berechtigungen für das Hinzufügen von␣
↪Registrierungen und Teilnehmern zu hinzu. Die Berechtigungen werden durch ``__init__
↪.py`` aufgerufen.
- ``/browser/``: kann zusätzliche Templates für die Gestaltung enthalten.
- ``/browser/configure.zcml`` registriert die Komponenten und legt fest, welche␣
↪Pagetemplates für welchen Content-Typ verwendet werden, und definiert die␣
↪angepassten Icons.
- ``/browser/registrantview.py``, ``browser/registrationview.py`` weisen dem View␣
↪spezifische Templates zu. Die Angaben werden durch FTI (factory type information)␣
↪in ``profiles/default/types/*.xml`` aufgerufen.
- ``browser/registrantview.pt``, ``browser/registrationview.pt`` sind Page-Templates␣
↪in TAL/METAL.
- ``content``: enthält die Implementierungen der Content-types.
- ``content/configure.zcml`` legt die Berechtigungen für die Content-types fest, die␣
↪mindestens erfüllt sein müssen.
- ``content/registrant.py``, ``content/registration.py`` enthalten die eigentliche␣
↪Definition der Datenstrukturen, und melden die Content-typen bei Archetypes an
- ``interfaces/``: Ordner enthält Beschreibungen der Interfaces für unsere␣
↪definierten Klassen, ``IRegistrant``und ``IRegistration``.
- ``interfaces/__init__.py`` importiert die Interface-Definitionen aus den Content-
↪Types. Datei wird vom Template ``contenttype`` angelegt.
- ``interfaces/registrant.py``, ``interfaces/registration.py`` legt Pflichtfelder und␣
↪die Benennungen der Felder fest. In ``registration.py`` contains('vs.registration.
↪interfaces.IRegistrant',)
- ``locales/de/LC_MESSAGES/`` kann einmal die sprachspezifischen Übersetzungsdateien␣
↪enthalten, in der Form ``vsregistration-de.po``. Genauere Informationen erhält man␣
↪in `i18n-locales und Plone 3.0`_
- ``profiles/default``: die Dateien definieren das Extension Profile gegenüber␣
↪Generic Setup; dieser Pfad wird in ``/configure.zcml`` festgelegt.
- ``profiles/default/factorytool.xml`` macht die Content-typen dem Factory-Tool␣
↪bekannt. Datei wird vom Template ``contenttype`` angelegt.
- ``profiles/default/metadata.xml`` enthält eine Versionsnummer. Datei wird vom␣
↪Template ``contenttype`` angelegt/modifiziert.
- ``profiles/default/portlets.xml`` sofern Portlets mit ``paster addcontent portlet``␣
↪erzeugt wurden.
- ``profiles/default/rolemap.xml`` Datei wird vom Template ``contenttype`` angelegt.
- ``profiles/default/types.xml`` Datei wird vom Template ``contenttype`` angelegt.
- ``profiles/default/workflows.xml`` Datei wird vom Template ``contenttype`` angelegt.
- ``profiles/default/types``
- ``profiles/default/types``
- ``tests/``: Unit-Tests für das Produkt.
```

Konventionen

The types are configured with the corresponding files in `types/*.xml`. Note that spaces are allowed in type names, but the corresponding XML file uses an underscore instead.

Die `types/*.xml`-Dateien werden mit Unterstrichen benannt werden

The „Factory-based Type Information with dynamic views“ refers to an FTI from Products.CMFDynamicViewFTI, which supports Plone's „display“ menu.

Metadaten zu vs.registration

Die Kontaktdaten des Autors (Name, Emailadresse, Homepage) sowie die Adresse des SVN-Repository werden mit einigen kommentierenden Angaben in `vs.registration.egg-info/PKG-INFO` geschrieben. Vor der Weitergabe des Produkts sollten einige Angaben in dieser Datei sowie in `CHANGES.txt` präzisiert werden.

Zum Weiterlesen

- ZopeSkel Archetypes HOWTO

Kollektionen

Kollektionen sollen auch Felder unserer Artikeltypen suchen können. Hierzu sind doe folgenden beiden Profile hinzuzufügen:

Hinzufügen von Index und Metadaten zum *Catalog Tool*

Diese werden im `catalog.xml`-Profil hinzugefügt:

```
<?xml version="1.0"?>
<object name="portal_catalog" meta_type="Plone Catalog Tool">
     <index name="myfield" meta_type="FieldIndex">
         <indexed_attr value="myfield"/>
    </index>
    <column value="myfield"/>
</object>
```

Folgende Typen von Indizees sind möglich:

- `DateIndex`
- `DateRangeIndex`
- `ExtendePathIndex`
- `FieldIndex`
- `KeywordIndex`
- `PathIndex`
- `TextIndex`
- `TopicIndex`
- `ZCTextIndex`

Hinzufügen von Index und Metadaten zum *ATCT Tool*

Diese werden im `portal_atct.xml`-Profil hinzugefügt:

```
<?xml version="1.0"?>
<atcttool>
    <topic_indexes>
        <index name="myfield"
               description="myfield's description"
               enabled="True"
               friendlyName="mytype's myfield">
            <criteria>ATSimpleStringCriterion</criteria>
        </index>
    </topic_indexes>
    <topic_metadata>
        <metadata  name="myfield"
                   description="myfield's description"
                   enabled="True"
                   friendlyName="mytype's myfield"/>
    </topic_metadata>
</atcttool>
```

Image Field hinzufügen

Als Änderungsanforderung kam die Darstellung eines Fotos für Registration.

Definieren des Interfaces

Das Interface wird in `registration/interfaces.py` erweitert:

```
from zope.interface import Interface
from Products.ATContentTypes.interface.image import IImageContent
...

class IRegistration(Interface, IImageContent):
    ...
```

Schemadefinition

Das Schema wird in `content/registration.py` definiert:

```
RegistrationSchema = folder.ATFolderSchema.copy() + atapi.Schema((
    ...
    atapi.ImageField('image',
               primary=True,
               languageIndependent=True,
               storage = AnnotationStorage(migrate=True),
               swallowResizeExceptions = zconf.swallowImageResizeExceptions.enable,
               pil_quality = zconf.pil_config.quality,
               pil_resize_algo = zconf.pil_config.resize_algo,
               max_size = zconf.ATImage.max_image_dimension,
               sizes= {'large'   : (768, 768),
                       'preview' : (400, 400),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                        'mini'    : (200, 200),
                        'thumb'   : (128, 128),
                        'tile'    :  (64, 64),
                        'icon'    :  (32, 32),
                        'listing' :  (16, 16),
                       },
                validators = (('checkImageMaxSize', V_REQUIRED)),
                widget = ImageWidget(
                         description = 'Image for this registration',
                         label= _(u'label_image', default=u'Image'),
                         show_content_type = False,)),

    ), marshall=PrimaryFieldMarshaller()
    )
...
RegistrationSchema['image'].storage = atapi.AnnotationStorage()
```

Anschließend wird das neue Feld implementiert:

```
class Registration(folder.ATFolder):
    """Contains multiple registrants
    """
    implements(IRegistration)
    ...
```

Anmerkung: Wenn das Feld nicht `image` heisst, sollten zusätzlich neue *getter*- und *setter*-Methoden zur Verfügung gestellt werden. Hierzu können Sie sich z.B. in *ATContentTypes* `getImage` und `setImage` anschauen.

Adapter für `IImageContent` schreiben

Der Adapter wird implementiert in `content/registration.py`:

```
from zope.interface import implements
...
from zope.publisher.interfaces import IPublishTraverse
from ZPublisher.BaseRequest import DefaultPublishTraverse

...

@adapter(IRegistration)
class ImageTraverser(DefaultPublishTraverse):

    implements(IPublishTraverse)

    def __init__(self, context, request):
        self.context = context
        self.request = request

    def publishTraverse(self, request, name):
        if name.startswith('image'):
            field = self.context.getField('image')
            image = None
            if name == 'image':
                image = field.getScale(self.context)
            else:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                scalename = name[len('image_'):]
                if scalename in field.getAvailableSizes(self.context):
                    image = field.getScale(self.context, scale=scalename)
            if image is not None and not isinstance(image, basestring):
                # image might be None or '' for empty images
                return image
        else:
            return super(ImageTraverser, self).publishTraverse(request, name)
```

Anschließend wird der Adapter konfiguriert in der Datei `content/configure.zcml`:

```
<adapter
 for="Products.ATContentTypes.interface.image.IImageContent zope.publisher.interfaces.
↪http.IHTTPRequest"
 factory=".registration.ImageTraverser"
 provides="zope.publisher.interfaces.IPublishTraverse" />
```

Erstellen eines Views

1. Zunächst wird der *View* registriert in `browser/configure.zcml`:

```
<browser:page
    for="Products.ATContentTypes.interface.image.IImageContent"
    class=".imagesupport.ImageView"
    permission="zope2.View"
    name="img_view"
    allowed_interface="..interfaces.IRegistration"
/>

<browser:page
    for="Products.ATContentTypes.interface.image.IImageContent"
    name="fullscreen"
    class=".views.FullscreenView"
    permission="zope2.View"
/>
<browser:page
    for="..interfaces.IRegistration"
    name="view"
    class=".registration.RegistrationView"
    permission="zope2.View"
/>
```

2. Anschließend wird der *View* implementiert wobei zunächst die Datei `browser/imagesupport.py` angelegt wird mit folgendem Inhalt:

```
from zope.interface import implements
from Products.CMFCore.utils import getToolByName
from Products.Five.browser import BrowserView
import urllib

class ImageView(BrowserView):

    def __init__(self, context, request):
        self.context = context
        self.request = request
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    def tag(self, **kwargs):
        """ tag """
        return self.context.getField('image').tag(self.context , **kwargs)

    def getImageSize(self, scale=None):
        """ image size """
        field = self.context.getField('image')
        return field.getSize(self.context,scale=scale)

    def hasImage(self):
        """ image size
        """
        field = self.context.getField('image')
        return field.get_size(self.context)
```

3. Anschließend wird noch der `FullscreenView` in `browser/registration.py` angegeben:

```
...

class FullscreenView(BrowserView):
    """
    """
    __call__ = ViewPageTemplateFile('fullscreen_view.pt')
```

4. Nun kopieren wir uns das PageTemplate `parts/plone/CMFPlone/skins/plone_content/image_view_fullscreen.pt` in `src/vs.registration/vs/registration/browser/` und ändern es folgendermaßen ab:

```
...
<div class="visualWrapper"
   tal:define="img_view context/@@img_view">
    <a href=""
       tal:attributes="href request/HTTP_REFERER"
       tal:condition="request/HTTP_REFERER">
        <span i18n:translate="label_back_to_site">
            Back to site
        </span>
        <br />
        <tal:block replace="structure img_view/tag" />
    </a>
    <a href=""
       tal:attributes="href here/portal_url"
       tal:condition="not: request/HTTP_REFERER">
        <span i18n:translate="label_home">
            Home
        </span>
        <br />
        <tal:block replace="structure img_view/tag"  />
    </a>
</div>
```

5. Schließlich ergänzen wir auch noch `browser/registration.pt`:

```
...
<div metal:fill-slot="main">
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
       <tal:main-macro
           metal:define-macro="main"
           tal:define="img_view context/@@img_view;
...
<span tal:condition="img_view/hasImage" tal:omit-tag="">
    <a href=""
        class="discreet"
        tal:attributes="href string:$here_url/fullscreen">
        <tal:block replace="structure python: img_view.tag(scale='preview')" />
        <br />
        <span class="visualNoPrint">
            <img src="" alt="" tal:replace="structure here/search_icon.gif" />
            <span i18n:translate="label_click_to_view_full_image" i18n:domain=
↪"plone">
                Click to view full-size image&hellip;
            </span>
        </span>
    </a>
</span>
```

3.6 Dexterity Artikeltypen

Dexterity ist ein modernes Content-Type-Framework, das die schnelle Erstellung von Artikeltypen erlaubt.

Im folgenden werden wir zwei Artikeltypen erstellen:

Registration Eine Veranstaltung, die Anmeldungen enthalten kann

Attendee Anmeldung zu einer Veranstaltung

Dabei wird in der Anleitung gezeigt:

1. die Erstellung von Artikeltypen durch ihre Schemadefinition
2. die Registrierung dieser Artikeltypen
3. die Erstellung von Ansichten für diese Artikeltypen
4. Das Ändern von Widgets für einzelne Felder
5. Die Eingabe von Standardwerten oder die Auswahl aus Vokabularien für bestimmte Felder
6. Validatoren zum Überprüfen der Feldinhalte

3.6.1 Download

Das vollständige Produkt können Sie sich in unserm SVN-Repository anschauen und herunterladen: https://dev.veit-schiele.de/svn/pen/vs.registration/trunk/.

3.6.2 Zum Weiterlesen

- Dexterity Developer Manual
- Local behavior support for Dexterity
- collective.miscbehaviors

Paket erstellen

Unsere Artikeltypen werden in einem eigenständigen Paket erstellt: `vs.registration`. Folgende Schritte sind hierzu notwendig:

1. Im `/src`-Verzeichnis erstellen wir aus der ZopeSkel-Vorlage `dexterity` das Grundgerüst für unser neues Produkt:

```
$ ../bin/zopeskel dexterity vs.registration
```

   Dabei verwenden wir den *èasy`*-Modus und antworten mit `True` bei der Frage, ob wir eine GenericSetup_Profil erstellen wollen.

   Dies erzeugt folgende Paketstruktur:

```
vs.registration
├── CHANGES.txt
├── CONTRIBUTORS.txt
├── DEXTERITY_README.rst
├── README.txt
├── bootstrap.py
├── buildout.cfg
├── docs
│   ├── LICENSE.GPL
│   └── LICENSE.txt
├── setup.cfg
├── setup.py
└── src
    └── vs
        ├── __init__.py
        └── registration
                ├── INTEGRATION.txt
            ├── __init__.py
            ├── configure.zcml
            ├── locales
            │   └── README.txt
            ├── profiles
            │   └── default
            │       ├── metadata.xml
            │       └── types.xml
            ├── static
            │   └── README.txt
            ├── testing.py
            └── tests
                ├── __init__.py
                ├── robot_test.txt
                ├── test_example.py
                └── test_robot.py
```

2. In der generierten `setup.py`-Datei sind unter `install_requires` folgende Pakete eingetragen:

```
install_requires=[
    'setuptools',
    'plone.app.dexterity',
    'plone.namedfile [blobs]',
```

3. Darüberhinaus werden meistens noch die folgenden Erweiterungen benötigt:

```
install_requires=[
    ...
    'collective.autopermission',
    'plone.app.referenceablebehavior',
    'plone.app.relationfield',
],
```

collective.autopermission erstellt Berechtigungen in Zope2 sobald eine `<permission />`-Anweisung verwendet wird.

plone.app.referenceablebehavior ermöglicht, dass unsere Dexterity-Artikeltypen von Archetypes-Referenzfeldern aus verfügbar werden.

plone.app.relationfield erlaubt uns, Referenzfelder zu verwenden

plone.namedfile erlaubt uns zusammen mit dem `[blobs]`-Extra, ZODB-Blobs zum Speichern unserer Dateien und Bilder zu verwenden.

Weitere bekannte Erweiterungen sind

plone.app.stagingbehavior basierend auf plone.app.iterate bietet es Unterstützung von Staging für Dexterity-Artikeltypen. Erfordert Plone 4.1.

plone.app.versioningbehavior basierend auf Products.CMFEditions bietet Unterstützung beim Speichern von Versionen für Dexterity-Artikeltypen. Erfordert Plone 4.0

collective.z3cform.datagridfield ein `z3c.form`-Widget zum Editieren von Listen von Subobjects mittels eines tabellenförmigen UI.

plone.app.lockingbehavior Locking-Integration für Dexterity-Artikeltypen.

4. Als nächstes fügen wir in der `configure.zcml`-Datei die folgenden Zeile ein:

```
<configure
    ...
    xmlns:grok="http://namespaces.zope.org/grok">
    <includeDependencies package="." />
    <grok:grok package="." />
    ...
</configure>
```

`<includeDependencies package="." />` schließt die Konfiguration für die in der `setup.py`-Datei aufgelisteten Abhängigkeiten ein.

`<grok:grok package="." />` ermöglicht Grok, automatisch Schema-Interfaces und `content`-Klassen zu initialisieren.

1. Als nächstes ändern wir die `buildout.cfg`-Datei um Dexteritys *Known Good Set of Versions* hinzuzufügen:

```
[buildout]
extensions =
    mr.developer
    ...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
extends =
    ...
    http://good-py.appspot.com/release/dexterity/1.0rc1?plone=4.1
    versions.cfg
...
[test]
recipe = zc.recipe.testrunner
eggs =
    vs.registration
defaults = ['--exit-with-status', '--auto-color', '--auto-progress']

[sources]
...
vs.registration = svn https://dev.plone.org/svn/pen/vs.registrtion/trunk
```

2. Nun können wir das `vs.registration`-Paket auch als Abhängigkeit in unser `vs.policy`-Produkt eintragen. Zunächst wird `vs.registration` in die Liste *`install_requires`* in `src/vs.policy/vs/policy/setup.py` eingetragen:

```
install_requires=[
    'setuptools',
    'Plone',
    'vs.event',
    'vs.theme',
    'vs.registration',
],
```

3. Schließlich editieren wir auch noch `profiles/default/metadata.xml` im selben Paket:

```
<dependencies>
    <dependency>profile-vs.event:default</dependency>
    <dependency>profile-vs.theme:default</dependency>
    <dependency>profile-vs.registration:default</dependency>
</dependencies>
```

Schema Interfaces

Für unsere Registrierungssoftware erstellen wir zunächst den Artikeltyp *Attendee* in `attendee.py`:

```
from five import grok
from zope import schema

from plone.directives import form, dexterity

from plone.app.textfield import RichText
from plone.namedfile.field import NamedImage

from vs.registration import _

class IAttendee(form.Schema):
    """An attendee for the event.
    """

    title = schema.TextLine(
            title=_(u"Name"),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        )

    description = schema.Text(
            title=_(u"A short summary"),
        )
```

from vs.registration import _ importiert die *Message Factory* aus der `__init__.py`-Datei:

```
from zope.i18nmessageid import MessageFactory
_ = MessageFactory("vs.registration")
```

Die Ereignisse, für die die Anmeldungen erfolgen können, werden in `registration.py` definiert:

```
from five import grok
from zope import schema

from plone.directives import form, dexterity
from plone.app.textfield import RichText

from vs.registration import _

class IRegistration(form.Schema):
    """A registration container for attendees.
    """

    title = schema.TextLine(
            title=_(u"Event name"),
        )

    description = schema.Text(
            title=_(u"Event summary"),
        )

    start = schema.Datetime(
            title=_(u"Start date"),
            required=False,
        )

    end = schema.Datetime(
            title=_(u"End date"),
            required=False,
        )

    details = RichText(
            title=_(u"Details"),
            description=_(u"Details about the event"),
            required=False,
        )
```

Factory Type Informations

Um die *Factory Type Informations* hinzuzufügen, erstellen wir in `profiles/default` das Profil `types.xml`:

```
<?xml version="1.0"?>
<object name="portal_types">
 <object name="vs.registration.registration" meta_type="Dexterity FTI" />
 <object name="vs.registration.attendee" meta_type="Dexterity FTI" />
</object>
```

- Um Konflikte zu vermeiden, wird der Paketname vorangestellt.
- Als `meta_type` muss `Dexterity FTI` verwendet werden.

Anschließend wird der Ordner `profiles/default/types` erstellt, in dem wir die Profile unserer beiden Artikeltypen erstellen, zunächst vs.reigistration.attendee.xml:

```
<?xml version="1.0"?>
<object name="vs.registrtion.attendee" meta_type="Dexterity FTI"
   i18n:domain="vs.registration" xmlns:i18n="http://xml.zope.org/namespaces/i18n">

 <!-- Basic metadata -->
 <property name="title" i18n:translate="">Attendee</property>
 <property name="description" i18n:translate="">An attendee</property>
 <property name="content_icon">user.gif</property>
 <property name="allow_discussion">True</property>
 <property name="global_allow">False</property>
 <property name="filter_content_types">False</property>
 <property name="allowed_content_types" />

 <!-- schema interface -->
 <property name="schema">vs.registration.attendee.IAttendee</property>

 <!-- class used for content items -->
 <property name="klass">plone.dexterity.content.Item</property>

 <!-- add permission -->
 <property name="add_permission">cmf.AddPortalContent</property>

 <!-- enabled behaviors -->
 <property name="behaviors">
     <element value="plone.app.content.interfaces.INameFromTitle" />
</property>

 <!-- View information -->
 <property name="default_view">view</property>
 <property name="default_view_fallback">False</property>
 <property name="view_methods">
  <element value="view"/>
 </property>

 <!-- Method aliases -->
 <alias from="(Default)" to="(dynamic view)"/>
 <alias from="edit" to="@@edit"/>
 <alias from="sharing" to="@@sharing"/>
 <alias from="view" to="(selected layout)"/>

 <!-- Actions -->
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 <action title="View" action_id="view" category="object" condition_expr=""
    url_expr="string:${object_url}" visible="True">
  <permission value="View"/>
 </action>
 <action title="Edit" action_id="edit" category="object" condition_expr=""
    url_expr="string:${object_url}/edit" visible="True">
  <permission value="Modify portal content"/>
 </action>
</object>
```

Anschließend erstellen wir noch `registration.xml`:

```
<?xml version="1.0"?>
<object name="vs.registration.registration"  meta_type="Dexterity FTI"
   i18n:domain="vs.registration" xmlns:i18n="http://xml.zope.org/namespaces/i18n">

 <!-- Basic metadata -->
 <property name="title" i18n:translate="">Registration</property>
 <property name="description" i18n:translate="">A folderish content type for attendees
↪</property>
 <property name="content_icon">folder_icon.gif</property>
 <property name="allow_discussion">False</property>
 <property name="global_allow">True</property>
 <property name="filter_content_types">True</property>
 <property name="allowed_content_types">
     <element value="vs.registration.attendee" />
 </property>

 <!-- schema interface -->
 <property name="schema">vs.registration.registration.IRegistration</property>

 <!-- class used for content items -->
 <property name="klass">plone.dexterity.content.Container</property>

 <!-- add permission -->
 <property name="add_permission">cmf.AddPortalContent</property>

 <!-- enabled behaviors -->
 <property name="behaviors">
     <element value="plone.app.content.interfaces.INameFromTitle" />
 </property>

 <!-- View information -->
 <property name="default_view">view</property>
 <property name="default_view_fallback">False</property>
 <property name="view_methods">
  <element value="view"/>
 </property>

 <!-- Method aliases -->
 <alias from="(Default)" to="(dynamic view)"/>
 <alias from="edit" to="@@edit"/>
 <alias from="sharing" to="@@sharing"/>
 <alias from="view" to="(selected layout)"/>

 <!-- Actions -->
 <action title="View" action_id="view" category="object" condition_expr=""
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    url_expr="string:${object_url}" visible="True">
  <permission value="View"/>
 </action>
 <action title="Edit" action_id="edit" category="object" condition_expr=""
    url_expr="string:${object_url}/edit" visible="True">
  <permission value="Modify portal content"/>
 </action>
</object>
```

Schließlich können Sie Ihre Instanz starten mit:

```
$ ./bin/instance fg
```

Beim Erstellen einer neuen Plone-Site wählen Sie das Profil `vs.registration`. Und nachdem die Site erstellt wurde, sollten Sie *Registration* im hinzufügen-Menü finden. Schließlich sollten Sie in dem Artikel vom Typ *Registration* einen *Attendee* hinzufügen können.

Views

Üblicherweise werden die benötigten Views zum Editieren und Ansehen der Artikel automatisch generiert. Falls diese Ansichten geändert werden sollen kann dies mittels `five.grok` einfach geschehen, indem z.B. in `registration.py` ein View erstellt werden mit:

```
from Acquisition import aq_inner
from Products.CMFCore.utils import getToolByName

from vs.registration.attendee import IAttendee

class View(grok.View):
    grok.context(IRegistration)
    grok.require('zope2.View')

    def attendees(self):
        """Return a catalog search result of attendees to show
        """

        context = aq_inner(self.context)
        catalog = getToolByName(context, 'portal_catalog')
        return catalog(object_provides=IAttendee.__identifier__,
                       path='/'.join(context.getPhysicalPath()),
                       sort_on='sortable_title')
```

`class View` Der Klassenname wird umgewandelt in Kleinbuchstaben und biltet dann den Namen des Views, in unserem Fall `@@view`.

Ggf. kann mit `grok.name('other-name')` ein anderer Klassenname vergeben werden.

`grok.context()` Diese Anweisung spezifiziert, dass der View in Artikeln mit dem `IRegistration`-Interface verwendet wird.

Soll zusätzlich ein Browser-Layer spezifiziert werden, so kann dies mit der `grok.layer()`-Anweisung erfolgen.

`grok.require()` Diese Anweisung spezifiziert die erforderliche Berechtigung zur Anzeige dieses Views.

Es wird der Zope3-Deklaration verwendet.

`zope2.View` und `zope.Public` sind die am häufigsten verwendeten Berechtigungen. Eine Liste weiterer Berechtigungen finden Sie z.B. in `parts/omelette/Products/Five/permissions.zcml`.

Anschließend kann auf derselben Ebene wie `registration.py` ein Ordner `registration_templates` und darin die Datei `view.pt` erstellt werden:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      xmlns:i18n="http://xml.zope.org/namespaces/i18n"
      lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="vs.registration">
<body>

<metal:main fill-slot="main">
    <tal:main-macro metal:define-macro="main"
        tal:define="toLocalizedTime nocall:context/@@plone/toLocalizedTime">

        <div tal:replace="structure provider:plone.abovecontenttitle" />

        <h1 class="documentFirstHeading" tal:content="context/title" />

        <div class="discreet">
            <tal:block condition="context/start">
                <span i18n:translate="label_from">From:</span>
                <span tal:content="python:context.start.strftime('%x %X')" />
            </tal:block>
            <tal:block condition="context/end">
                <span i18n:translate="label_to">To:</span>
                <span tal:content="python:context.end.strftime('%x %X')" />
            </tal:block>
        </div>

        <div tal:replace="structure provider:plone.belowcontenttitle" />

        <p class="documentDescription" tal:content="context/description" />

        <div tal:replace="structure provider:plone.abovecontentbody" />

        <h2 i18n:translate="heading_attendees">Attendees</h2>
        <dl>
            <tal:block repeat="attendee view/attendees">
                <dt>
                    <a tal:attributes="href attendee/getURL"
                       tal:content="attendee/Title" />
                </dt>
                <dd tal:content="attendee/Description" />
            </tal:block>
        </dl>

        <div tal:replace="structure provider:plone.belowcontentbody" />

    </tal:main-macro>
</metal:main>

</body>
</html>
```

Bemerkung: Damit im Feld mit dem visuellen Editor die HTML-Tags nicht ausgefiltert weren, muss hier die Angabe `/output` angegeben werden.

Siehe auch:

- Asko Soukka: Create custom views for Dexterity-types TTW

Widgets

Falls die übliche Verwendung von Widgets nicht ausreicht, können auch spezifische Widgets verwendet werden, die eine größere Kontrolle über die Ausgabe erlauben.

Als Beispiel verwenden wir die `View`-Klasse aus Views, die wir jedoch diesmal von `plone.directives.dexterity.DisplayForm` ableiten:

```
class View(dexterity.DisplayForm):
    grok.context(ISession)
    grok.require('zope2.View')
```

Hierdurch erhalten wir einige zusätzliche Eigenschaften, die wir in unserem Template verwenden können:

`view.w` ist ein Dictionary aller Display-Widgets.

Als Schlüssel für diese Widgets wird der Feldname verwendet oder, sofern das Feld aus einem *Behavior* kommt, wird dem Feldnamen das Interface dieses *Behavior* vorangestellt

`view.widgets` enthalten eine Liste von Widgets in der Reihenfolge des Standard-Fieldset

`view.groups` enthält eine Liste von Fieldsets

`view.fieldsets` enthält ein Distionary, das Fieldset-Namen Fieldsets zuweist.

Auf einen Fieldset (group) können alle dort verfügbaren Widgets aufgelistet werden.

In `project_templates/view.pt` kann dann z.B.:

```
<div tal:content="structure context/details/output" />
```

ersetzt werden durch:

```
<div tal:content="structure view/w/details/render" />
```

Standardwerte, Vokabularien und Autovervollständigung

Standardwerte

Oft vereinfacht es die Bedienung deutlich, wenn in Feldern Standardwerte eingetragen werden. Diese Werte werden im Hinzufügen-Formular gesetzt.

In unserem Beispiel sollen die Standardwerte für den Beginn und das Ende einer Veranstaltung eine Woche in der Zukunft liegen. Hierzu fügen wir in `registration.py` folgendes hinzu:

```
import datetime
...
@form.default_value(field=IRegistration['start'])
def startDefaultValue(data):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # To get hold of the folder, do: context = data.context
    return datetime.datetime.today() + datetime.timedelta(7)

@form.default_value(field=IRegistration['end'])
def endDefaultValue(data):
    # To get hold of the folder, do: context = data.context
    return datetime.datetime.today() + datetime.timedelta(10)
```

Der *Decorator* kann ein oder mehrere Unterscheidungskriterien haben. Folgende Unterscheidungskriterien sind möglich:

context Der Kontexttyp, z.B. ein Interface

request Der Request-Tp, z.B. ein Layer Marker Interface

view Der Formulartyp, z.B. eine Formularinstanz oder ein Interface.

field Die Feld-Instanz oder das Interface eines Feldes

Neben dem `default_value`-Decorator gibt es noch zwei weitere Decorators:

widget Der Widget-Typ, z.B. ein Interface

widget_label bietet ein dynamische Label für Widgets wobei es dieselben Unterscheidungskriterien zulässt wie `default_value`.

button_label bietet dynamische Label für Tasten mit den Unterscheidungskriterien `content, ``request`, `form`, `manager` und `button`.

In der Dokumentation zu plone.directives.form finden Sie weitere Informationen hierzu.

Vokabularien

Vokabularien werden üblicherweise zusammen mit Auswahlfeldern verwendet. Um nur eine Auswahl zuzulassen, kann das `Choise`-Feld direkt verwendet werden:

```
class IMySchema(form.Schema):
    myChoice = schema.Choice(...)
```

Für Multiple-Choice-Felder können `List`, `Tuple`, `Set` oder `Frozenset` mit `value_type=schema.Choice` verwendet werden, also z.B.:

```
class IMySchema(form.Schema):
    myList = schema.List(
        title=u"My list",
        value_type=schema.Choice(values=['red', 'green', 'blue', 'yellow']))
```

Ein `Choice`-Feld kann eines der folgenden Argumente erhalten:

- Werte aus einer Liste statischer Werte
- Werte aus einer Quelle, die mit `IContextSourceBinder` oder einer `ISource`-Intanz angegeben werden
- Werte können aus einem Vokabular stammen, das als `ÌVocabulary`-Instanz oder als Name eines `IVocabularyFactory`-Utility angegeben wird

term Eintrag in ein Vokabular

token ASCII-Zeichenkette, die beim Abschicken eines Formulars übermittelt wird um den Term eindeutig zu identifizieren.

value Der aktuelle Wert, der in einem Objekt gespeichert wird

title Übersetzbare Unicode-Zeichenkette

Verfügbare Vokabularien

In Plone werden Ihnen bereits eine ganze Reihe von Vokabularien in `plone.app.vocabularies` zur Verfügung gestellt. Die gebräuchlichsten sind:

plone.app.vocabularies.AvailableContentLanguages Eine Liste aller verfügbaren Sprachen

plone.app.vocabularies.SupportedContentLanguages Eine Liste aller aktuell unterstützten Sprachen

plone.app.vocabularies.Roles Die in der Site verfügbaren Rollen

plone.app.vocabularies.PortalTypes Eine Liste der im Portal Types Tool registrierten Artikeltypen

plone.app.vocabularies.ReallyUserFriendlyTypes Eine Liste derjenigen Artikeltypen, die für Nutzer von Bedeutung sind

plone.app.vocabularies.Workflows Eine Liste aller Arbeitsabläufe

plone.app.vocabularies.WorkflowStates Eine Liste aller Arbeitsablaufstadien

plone.app.vocabularies.WorkflowTransitions Eine Liste aller Übergänge zwischen Arbeitsablaufstadien

Mit `plone.principalsource` steht uns ein weiteres Paket mit verschiedenen Vokabularien bereit, das zur Auswahl von Nutzern und Gruppen hilfreich ist:

plone.principalsource.Users Eine Liste aller Nutzer

plone.principalsource.Groups Eine Liste aller Gruppen

plone.principalsource.Principals Eine Liste aller Berechtigungen für Nutzer und Gruppen

Statische Vokabularien

Hier ein Beispiel für ein statisches Vokabular:

```
from zope.schema.vocabulary import SimpleVocabulary, SimpleTerm

organisers = SimpleVocabulary(
    [SimpleTerm(value=u'vsc', title=_(u'Veit Schiele Communications')),
     SimpleTerm(value=u'zopyx', title=_(u'Zopyx Limited'))]
    )

organiser = schema.Choice(
            title=_(u"Organiser"),
            vocabulary=organisers,
            required=False,
        )
```

Dynamische Vokabularien

Statische Vokabularien sind in zweierlei Hinsicht beschränkt: zum einen sind sie hartkodiert in Python, zum anderen werden die gespeicherten Werte nicht getrennt von den Labels gespeichert.

Ein dynamische Vokabular kann nun erzeugt werden indem ein sog. *Context Source Binder* verwendet wird. Ein solcher kann einfach aufgerufen werden durch eine Funktion oder ein Objekt mit einer `__call__`-Methode, die das `IContextSourceBinder`-Interface zusammen mit einem Kontext-Argument bereitstellt. Der Aufruf soll ein Vokabular ausgeben, das am einfachsten zu bekommen ist, wenn die `SimpleVocabulary`-Klasse aus `zope.schema` verwendet wird.

Im folgenden nun ein Beispiel für eine Funktion, die alle Nutzer einer bestimmten Gruppe zurückgibt:

```
from zope.schema.interfaces import IContextSourceBinder
from zope.schema.vocabulary import SimpleVocabulary
from Products.CMFCore.utils import getToolByName

@grok.provider(IContextSourceBinder)
def possibleOrganisers(context):
    acl_users = getToolByName(context, 'acl_users')
    group = acl_users.getGroupById('organisers')
    terms = []

    if group is not None:
        for member_id in group.getMemberIds():
            user = acl_users.getUserById(member_id)
            if user is not None:
                member_name = user.getProperty('fullname') or member_id
                terms.append(SimpleVocabulary.createTerm(member_id, str(member_id),␣
↪member_name))

   return SimpleVocabulary(terms)
```

Parametriesierte Vokabularien

Das obige Beispiel kann erweitert werden indem der Gruppenname aus der Funktion herausgenommen wird und sich dann für jedes Feld unabhängig setzen lässt. Hierfür wird dann `IContextSourceBinder` in eine eigene Klasse ausgelagert, die mit dem Gruppennamen initialisiert wird:

```
class GroupMembers(object):
    """Context source binder to provide a vocabulary of users in a given
    group.
    """

    grok.implements(IContextSourceBinder)

    def __init__(self, group_name):
        self.group_name = group_name

    def __call__(self, context):
        acl_users = getToolByName(context, 'acl_users')
        group = acl_users.getGroupById(self.group_name)
        terms = []

        if group is not None:
            for member_id in group.getMemberIds():
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                user = acl_users.getUserById(member_id)
                if user is not None:
                    member_name = user.getProperty('fullname') or member_id
                    terms.append(SimpleVocabulary.createTerm(member_id, str(member_
↪id), member_name))

        return SimpleVocabulary(terms)
```

Benannte Vokabularien

Sollen Vokabularien nicht nur im Kontext verfügbar sein sondern als Komponenten, werden sog. *Named Vocabularies* erstellt. Diese werden als *named utilities* registriert werden und sind anschließend in einem Schema mit ihrem Namen referenziert werden. Damit lassen sich Vokabularien in eigenständigen Paketen erstellen.

VDEX-Vokabularien

collective.vdexvocabulary erlaubt die Verwendung von IMS VDEX-Vokabularien und bietet darüberhinaus noch weitere Vorteile wie:

- `i18n`-Unterstützung, wie sie im IMS VDEX-Standard definiert ist.
- Unterstützung für Sortierung auch von Unicode-Zeichen. sofern zope.ucol installiert ist
- Einfache Registrierung mit `zcml`
- Relationen wie sie im IMS VDEX-Standard spezifiziert sind

`collective.elephantvocabulary`

collective.elephantvocabulary ist ein Wrapper für `zope.schema`-Vokabularien wodurch diese keinen ihre Einträge mehr vergessen.

Autovervollständigung

`plone.formwidget.autocomplete` erweitert `z3c.formwidget.query` um ein nutzerfreundlicheres Interface für Felder bereitzustellen, bei dem nach der Eingabe von wenigen Zeichen bereits die möglichen Werte angezeigt werden.

Das Widget wird bereits mit `plone.app.dexterity` mitgeliefert, sodass wir es einfach z.B. in `registration.py` verwenden können, mit:

```
form.widget(organiser=AutocompleteFieldWidget)
organiser = schema.Choice(
        title=_(u"Organiser"),
        vocabulary=u"plone.principalsource.Users",
        required=False,
    )
```

Computed Fields

In unserem speziellen Fall wollen wir aus den zwei Feldern *Vorname* und *Nachname* den Titel unseres *Employee*-Artikeltyps erstellen lassen.

Hierzu erhält die Klasse `Employee` die Eigenschaft `title`, die sich aus den Feldern `first_name` und `surname` zusammensetzt:

```
class Employee(Item):
"""Customised Employee content class"""
    @property
    def title(self):
        if hasattr(self, 'first_names') and hasattr(self, 'surname'):
            return self.first_name + ' ' + self.surname
        else:
            return ''
```

Generieren der ID

Etwas komplexer ist das Erstellen der ID aus einem berechneten Titel:

```
from plone.app.content.interfaces import INameFromTitle
class INameFromEmployeeTitle(INameFromTitle):
    def title():
        """Return a processed title"""

class NameFromEmployeeTitle(object):
    implements(INameFromEmployeeTitle)

    def __init__(self, context):
        self.context = context

    @property
    def title(self):
        return self.context.first_name + ' ' + self.context.surname

    def setTitle(self, value):
        return
```

Nun registrieren wir noch einen Adapter für die Dexterity-Interface-Klasse:

```
<adapter
    for="vs.registration.employee.ISampleEmployee"
    provides="vs.registration.employee.INameFromEmployeeTitle"
    factory="vs.registration.employee.NameFromEmployeeTitle"
/>
```

Validatoren

Die einfachste Form der Validierung liefert bereits die `z3c.form`-Bibliothek indem sie überprüft, ob die Eingaben dem Feldtyp entsprechen. Darüberhinaus können für jedes Feld die folgenden Eigenschaften angegeben werden, mit denen der Inhalt des Feldes überprüft wird:

`required` Als Werte sind hier `True` und `False` zulässig um den Wert eines Feldes als erforderlich oder optional zu kennzeichnen.

`min` und `max` wird verwendet für Felder vom Typ

- `Int`
- `Float`
- `Datetime`
- `Date`
- `Timedelta`

Hiermit lassen sich die zulässigen Mindest- und Höchstwerte definieren.

`min_length` und `max_length` kann für folgende Felder angegeben werden:

- Kollektionen
  - `Tuple`
  - `List`
  - `Set`
  - `Frozenset`
  - `Dict`
- Textfelder
  - `Bytes`
  - `BytesLine`
  - `ASCII`
  - `ASCIILine`
  - `Text`
  - `TextLine`

Bedingungen (Constraints)

Eine Constraint-Funktion sollte als Argument den Wert des Feldes erhalten und als Ergebnis `True` oder `False` liefern. Dabei sind Constraints zwar einfach zu schreiben, sie geben jedoch selten eine nutzerfreundliche Fehlermeldung aus. Mit `z3c.form`-Error-View-Snippets lassen sich diese Fehlermeldungen jedoch anpassen. Weitere Informationen erhalten Sie in Customizing Error Messages.

Invarianten

Während Constraints immer nur ein einzelnes Feld überprüfen können, lassen sich mit Invarianten mehrere Felder abgleichen.

Ein Beispiel hierfür ist der Vergleich von Start- und Enddatum:

```
from zope.interface import invariant, Invalid

class StartBeforeEnd(Invalid):
    __doc__ = _(u"The start or end date is invalid")

class IRegistration(form.Schema):
    ...
    start = schema.Datetime(
            title=_(u"Start date"),
            required=False,
        )

    end = schema.Datetime(
            title=_(u"End date"),
            required=False,
        )
    ...
    @invariant
    def validateStartEnd(data):
        if data.start is not None and data.end is not None:
            if data.start > data.end:
                raise StartBeforeEnd(_(u"The start date must be before the end date.
↪"))

class StartBeforeEnd(Invalid):
    __doc__ = _(u"The start or end date is invalid")
```

Formular-Validatoren

Mächtigere Validatoren können mit den `z3c.form`-Widget-Validatoren geschrieben werden. Weitere Informationen hierzu erhalten Sie in der z3c.form-Dokumentation.

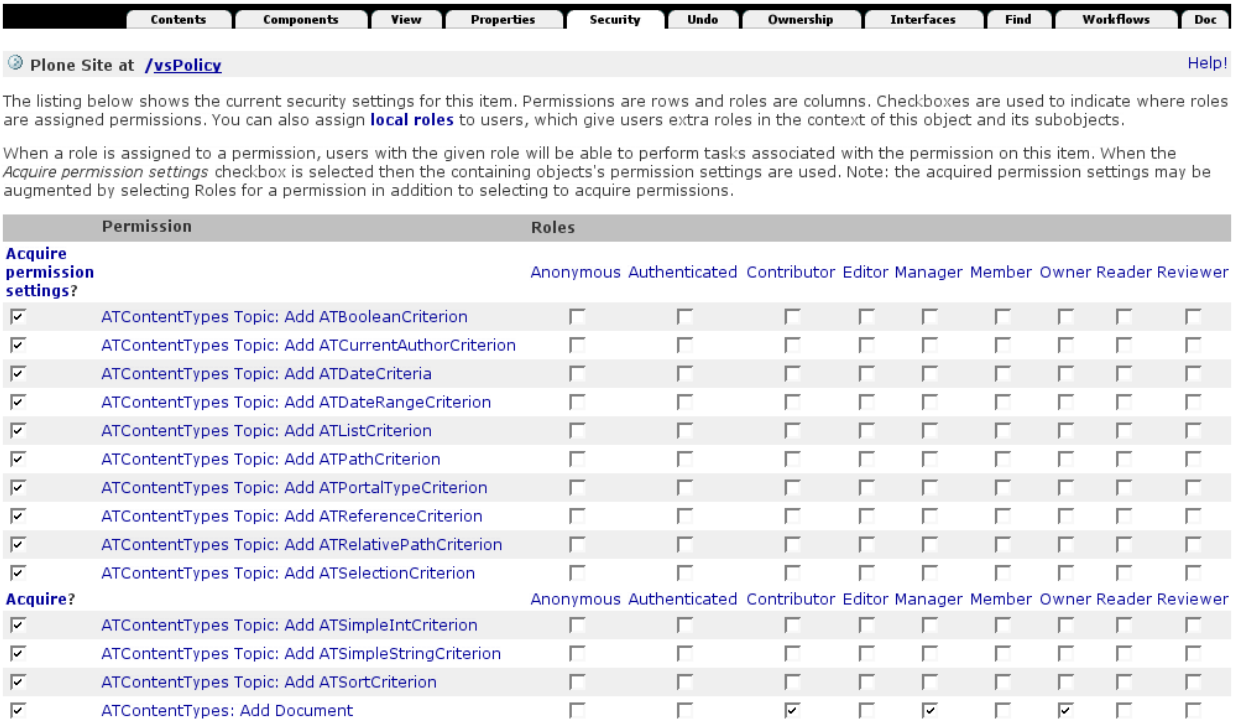
3.7 Sicherheit und Arbeitsabläufe

3.7.1 Berechtigungen

In Plone kann für viele verschiedene Nutzungsfälle eine Zuordnung zu Rollen vorgenommen werden, die zur Ausführung berechtigt sind.

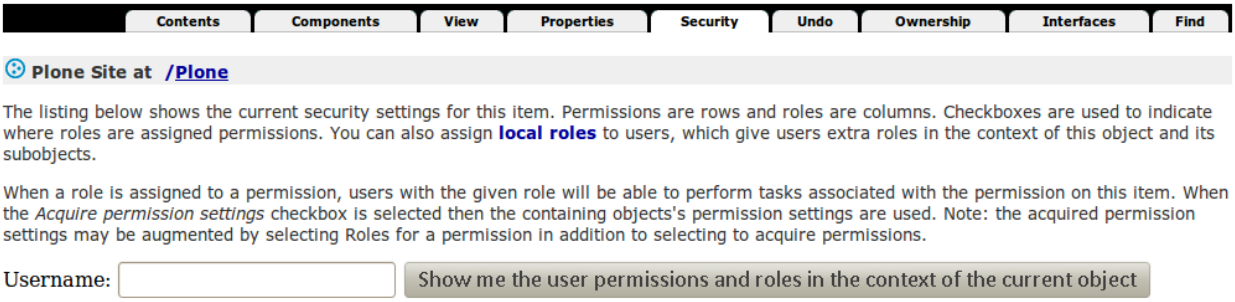
Sie können sich diese Zuordnungstabelle anschauen, indem Sie im ZMI auf den *Security*-Reiter klicken:

In der Tabelle können für ein bestimmtes Objekt die verschiedenen Berechtigungen den einzelnen Rollen zugewiesen werden. Beachten Sie bitte, dass für die meisten Berechtigungen *Acquire permission settings?* angeklickt ist und damit die Berechtigungen vom übergeordneten Objekt übernommen werden. Weitere Rollen können dann einfach durch Anklicken hinzugefügt werden.



Plone 4

Plone 4 kommt mit Zope 2.12 und der damit gegebenen Möglichkeit, die Berechtigungen eines bestimmten Nutzers im Kontext anzuzeigen:



Anmerkung 1: Sie sollten niemals die Berechtigungen außerhalb des Wurzelverzeichnisses der Site ändern, da ansonsten kaum noch zu kontrollieren ist, welche Berechtigungen wo gesetzt sind.

Anmerkung 2: Im Quellcode sollte immer die Berechtigung (*permission*) und nie die Rolle (*role*) überprüft werden.

Die am häufigsten verwendeten Berechtigungen sind:

CMFCore Permission	Five Permission	Beschreibung
AccessContentsInformation	zope2.AccessContentsInformation	Niedrigschwellige Zope-Berechtigung, die den Zugang zu Objekten kontrolliert
View	zope2.View	Zugang zu der Hauptansicht eines Inhaltsobjekts
ListFolderContents	cmf.ListFolderContents	Berechtigung, die Verzeichnisübersicht zu sehen
ModifyPortalContent	cmf.ModifyPortalContent	Editiermöglichkeit für die Inhalte
ManagePortal	cmf.ManagePortal	Operationen, die typischerweise der Manager-Rolle zugewiesen werden
AddPortalContent	cmf.AddPortalContent	Möglichkeit, neue Inhalte hinzuzufügen. Darüberhinaus haben viele Artikeltypen ihre eigenen Berechtigungen zum Hinzufügen von Inhalten, so dass beide Berechtigungen benötigt werden

Eine vollständige Übersicht über alle `CMFCore`-Berechtigungen erhalten Sie in `Products.CMFCore.permissions` und über alle `Five`-Berechtigungen in `Products.Five.permissions.zcml`.

Diese Berechtigungen können in ZCML-Anweisungen angegeben werden, z.B. in `src/vs.registration/vs/registration/browser/configure.zcml`:

```
<browser:page
    for="..interfaces.IRegistration"
    name="view"
    class=".registration.RegistrationView"
    permission="zope2.View"
    />
```

Erstellen von neuen Berechtigungen

Obwohl die Standard-Berechtigungen verwendet werden sollten, um grundlegende Aktionen (anzeigen, ändern, löschen, schreiben) zu steuern, ist es manchmal sinnvoll, neue Berechtigungen erstellen. Kombiniert mit benutzerdefinierten Workflows können benutzerdefinierte Berechtigungen verwendet werden, um angepasste Anwendungen zu erstellen.

Die Berechtigungen können dann einfach in der `configure.zcml`-Datei erstellt werden:

```
<permission
     id="vs.registration.AddRegistrant"
     title="vs.registration: Add registrant"
     />

 <permission
     id="vs.registration.ModifyRegistrant"
     title="vs.registration: Modify registrant"
     />
```

In Plone < 4.0 oder genauer in Zope < 2.12 muss zusätzlich collective.autopermission installiert werden. Hierfür wird in der `setup.py`-Datei folgendes eingetragen:

```
install_requires=[
    ...
    'collective.autopermission',
],
```

Neue Berechtigungen werden üblicherweise nur der Manager-Rolle übertragen. Um anderen Rollen diese Berechtigungen ebenfalls zu übertragen, kann das Profil `rolemap.xml` erstellt werden:

```
<?xml version="1.0"?>
<rolemap>
  <permissions>
    <permission name="vs.registration: Add registrant" acquire="True">
      <role name="Owner"/>
      <role name="Manager"/>
      <role name="Member"/>
      <role name="Contributor"/>
    </permission>
    <permission name="vs.registration: Modify registrant" acquire="True">
      <role name="Manager"/>
      <role name="Reviewer"/>
    </permission>
  </permissions>
</rolemap>
```

Nutzer

Plone 4 ermöglicht mit plone.app.users neue Felder zum Registrieren an der Website und den persönlichen Einstellungen hinzuzufügen.

Überschreiben des bestehenden Schemas

Das Standardschema von `plone.app.users` ist in einem Hilfsprogramm (*Utility*) definiert, das überschrieben werden muss um ein neues Schema anzulegen. Dies geschieht in der Datei `profiles/default/componentregistry.xml`:

```
<?xml version="1.0"?>
<componentregistry>
  <utilities>
    <utility
      interface="plone.app.users.userdataschema.IUserDataSchemaProvider"
      factory="vs.policy.userdataschema.UserDataSchemaProvider"
    />
  </utilities>
</componentregistry>
```

Nun wird die *factory* in `userdataschema.py` erstellt:

```
from plone.app.users.userdataschema import IUserDataSchemaProvider

class UserDataSchemaProvider(object):
    implements(IUserDataSchemaProvider)

    def getSchema(self):
        """
        """
        return IEnhancedUserDataSchema
```

Schließlich wird eine Unterklasse des Standardschemas `IUserDataSchema` erstellt:

```
from plone.app.users.userdataschema import IUserDataSchema

class IEnhancedUserDataSchema(IUserDataSchema):
    """ Use all the fields from the default user data schema, and add various
    extra fields.
   """
```

Hinzufügen neuer Felder

Nun können neue Felder definiert werden, z.B.:

```
newsletter = schema.Bool(
    title=_(u'label_newsletter', default=u'Subscribe to newsletter'),
    description=_(u'help_newsletter',
                  default=u"If you tick this box, we'll subscribe you to "
                    "our newsletter."),
    required=False,
    )
```

Bedingungen

Felder mit Bedingungen für eine erfolgreiche Anmeldung können als `constraint` angegeben werden, z.B.:

```
def validateAccept(value):
    if not value == True:
        return False
    return True

class IEnhancedUserDataSchema(IUserDataSchema):
    # ...
    accept = schema.Bool(
        title=_(u'label_accept', default=u'Accept terms of use'),
        description=_(u'help_accept',
                      default=u"Tick this box to indicate that you have found,"
                      " read and accepted the terms of use for this site. "),
        required=True,
        constraint=validateAccept,
        )
```

Eigenschaften

Neue Eigenschaften lassen sich in den *memberdata properties* speichern indem eine Datei `memberdata_properties.xml` in `profiles/default/` erstellt wird. Dabei werden alle Felder hinzugefügt bis auf das `accept`-Feld, das zwingend für die Registrierung erforderlich ist:

```
<?xml version="1.0"?>
<object name="portal_memberdata" meta_type="Plone Memberdata Tool">
<property name="subscribe_newsletter" type="boolean"></property>
</property>
</object>
```

Registrierung

Die Felder für die Registrierung werden in `profiles/default/propertiestool.xml` angegeben:

```
<?xml version="1.0"?>
<object name="portal_properties" meta_type="Plone Properties Tool">
 <object name="site_properties" meta_type="Plone Property Sheet">
  <property name="user_registration_fields" type="lines">
   <element value="newsletter" />
   <element value="accept" />
  </property>
 </object>
</object>
```

Persönliche Informationen

Um die Felder auch im Formular mit den persönlichen Informationen `@@personal-information` zu sehen, wird zunächst der Adapter des Nutzerobjekts in der `overrides.zcml` überschrieben:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    i18n_domain="vs.policy.userdata">
  <adapter
    provides=".userdataschema.IEnhancedUserDataSchema"
    for="Products.CMFCore.interfaces.ISiteRoot"
    factory=".adapter.EnhancedUserDataPanelAdapter"
    />
</configure>
```

Anschließend müssen leider die Felder nochmals angegeben werden. Hierzu fügen wir `adapter.py` hinzu mit:

```
from plone.app.users.browser.personalpreferences import UserDataPanelAdapter

class EnhancedUserDataPanelAdapter(UserDataPanelAdapter):
    """
    """
    def get_newsletter(self):
        return self.context.getProperty('newsletter', '')
    def set_newsletter(self, value):
        return self.context.setMemberProperties({'newsletter': value})
    newsletter = property(get_newsletter, set_newsletter)

    def get_accept(self):
        return self.context.getProperty('accept', '')
    def set_accept(self, value):
        return self.context.setMemberProperties({'accept': value})
    accept = property(get_accept, set_accept)
```

Zum Weiterlesen

Member manipulation Getting logged-in member, any member and member information

collective.examples.userdata Python-Egg mit Beispielen, wie das Schema der Nutzerdaten erweitert werden kann.

Gruppen

Um neue Gruppen programmatisch hinzuzufügen, erstellen wir in der `setuphandlers.py`-Datei die Methode `setupGroups`, die sowohl den Pluggable Authentication Service (PAS) als auch das `portal_groups`-Tool verwendet:

```
import plone.api
def setupGroups(portal):
    acl_users = plone.api.portal.get_tool('acl_users')
    if not acl_users.searchGroups(name='Staff'):
        gtool = getToolByName(portal, 'portal_groups')
        gtool.addGroup('Staff', roles=['StaffMember'])
def importVarious(context):
    """Miscellanous steps import handle
    """
    if context.readDataFile('vs.policy-various.txt') is None:
       return
    portal = context.getSite()
    setupGroups(portal)
```

Für Plone 4.2 steht plone.api noch nicht zur Verfügung. Stattdessen muss noch `getToolByName` verwendet werden:

```
from Products.CMFCore.utils import getToolByName
def setupGroups(portal):
    acl_users = getToolByName(portal, 'acl_users')
    if not acl_users.searchGroups(name='Staff'):
        gtool = getToolByName(portal, 'portal_groups')
        gtool.addGroup('Staff', roles=['StaffMember'])
...
```

Tests

```
def test_staffmember_group_added(self):
    portal = self.layer['portal']
    acl_users = portal['acl_users']
    self.assertEqual(1,
        len(acl_users.searchGroups(name='Staff')))
```

Rollen

Rollen können sowohl Nutzern als auch Gruppen zugewiesen werden. Dabei empfiehlt sich im Allgemeinen, Rollen zu erstellen, denen bestimmte Rechte zugewiesen werden, anstatt jedem Nutzer die jeweiligen Rechte zuzuweisen.

So haben z.B. die Gruppen *Reviewers* und *Administrators* die entsprechenden *Manager*- oder *Reviewer*-Rollen.

- Die **globale** Zuordnung von Nutzern und Gruppen zu einer Rolle können Sie in *Plone Konfiguration* → *Benutzer und Gruppen* vornehmen.
- Die **lokale** Zuweisung von Rollen erfolgt im Allgemeinen im *Zugriff*-Reiter eines Inhaltsobjekts. Dort suchen Sie zunächst nach einem Nutzer oder einer Gruppe, um in den Suchergebnissen anschließend Zuweisungen zu bestimmten Rechten vornehmen zu können. Die lokal verfügbaren Rollen sind beschränkt auf die explizit angegebenen (s.a. `plone/app/workflow/localroles.py` und `plone/app/workflow/configure.zcml`).

Plone selbst kommt mit sieben verschiedenen Rollen:

`Member` ist die Standardrolle für einen angemeldeten Nutzer, der nur wenige Rechte zugewiesen sind.

`Manager` ist die *super-user*-Rolle; sie ist der *Administrators*-Gruppe zugeordnet.

`Reviewer` erlaubt Inhabern der Rolle, Artikel zu sehen und zu bestätigen, deren Inhalte zur Veröffentlichung vorgeschlagen wurden.

`Reader` ist lediglich als lokale Rolle vorgesehen, wenn `Member` die Inhalte nicht sehen können.

`Editor` ist das Pendant zu `Reader`, um Eigentümern die Möglichkeit zu geben, lokal Schreibrechte zu erteilen.

`Contributor` ermöglicht einem Eigentümer, das *Hinzufügen*-Recht in einem Ordner an andere zu übertragen. Es erscheint im *Zugriff*-Reiter unter *Kann hinzufügen*.

`Site Administrator` hat Management-Rechte bis auf

- die Verwendung des *Wartung*-Kontrollfeldes
- den Zugang zum Zope Management Interface (ZMI)
- das Hinzufügen und Entfernen von Produkten
- die Änderung des Aussehens
- die Änderung der Cache-Konfiguration

Darüberhinaus definiert Zope noch drei automatisch zugewiesene Rollen:

`Owner` wird normalerweise dem Nutzer zugeordnet, der das Inhaltsobjekt erstellte.

`Authenticated` wird angemeldeten Nutzern zugewiesen. Diese Rolle ist niedrigschwelliger als `Member` und kann nicht explizit zugewiesen werden.

`Anonymous` ist die Rolle für nicht-angemeldete Nutzer.

Rechte und Rollen programmatisch ändern

Die meisten niedrigschwelligen Sicherheitsmethoden werden in der `AccessControl.Role.RoleManager`-Klasse definiert. Diese ist in allen Inhaltsobjekten verfügbar einschließlich des *Plone Site*-Objekts selbst. Sie können sich diese Methoden im *Doc*-Reiter im Wurzelverzeichnis Ihrer Plone-Site anschauen:

|RoleManager|

Darüberhinaus empfiehlt sich auch ein Blick in den *Doc*-Reiter des *PlonePAS Membership Tool*.

Und hier noch einige der am häufigsten verwendeten programmatischen Änderungen von Rechten und Rollen:

Rechte überprüfen:

```
from AccessControl import getSecurityManager
from Products.CMFCore.permissions import ModifyPortalContent

sm = getSecurityManager()
if sm.checkPermission(ModifyPortalContent, context):
    # do something
```

Rechte ändern:

```
context.manage_permission("Portlets: Manage portlets",
                          roles=['Manager', 'Owner'], acquire=1)
```

Überprüfen, ob ein Nutzer angemeldet ist oder nicht (d.i. *anonymous*):

```
from Products.CMFCore.utils import getToolByName
mtool = getToolByName(context, 'portal_membership')
if mtool.isAnonymousUser():
    # do something
```

Herausfinden des aktuellen Nutzers:

```
member = mtool.getAuthenticatedMember()
user_id = member.getId()
```

Finden von Mitgliedern anhand der ID:

```
admin_user = mtool.getMemberById('admin')
```

Lokale Rollen

Seit Plone 4.0 lassen sich Rollen mit dem GenericSetup-Profil `sharing.xml` für die *Freigabe*-Ansicht konfigurieren.

Lokale Rollen erstellen

Im folgenden Beispiel wird die Rolle *Event-Manager* erstellt:

```
<?xml version="1.0"?>
<sharing xmlns:i18n="http://xml.zope.org/namespaces/i18n"
         i18n:domain="vs.policy">
  <role
      id="Event-Manager"
```

(Fortsetzung auf der nächsten Seite)

RoleManager An object that has configurable permissions			
ac_inherited_permissions	Change permissions (Manager)	self, all=0	method
aclAChecked	(Anonymous, Contributor, Editor, Manager, Reader)		str
aclEChecked	(Anonymous, Contributor, Editor, Manager, Reader)		str
aclPChecked	(Anonymous, Contributor, Editor, Manager, Reader)		str
acquiredRolesAreUsedBy	Change permissions (Manager)	self, permission	Used by management screen.
get_local_roles	(Anonymous, Contributor, Editor, Manager, Reader)	self	method
get_local_roles_for_userid	(Anonymous, Contributor, Editor, Manager, Reader)	self, userid	method
get_valid_userids	(Anonymous, Contributor, Editor, Manager,	self	method

(Fortsetzung der vorherigen Seite)

```
      title="Manages the registrations for events"
      permission="Add Registration"
      i18n:attributes="title"
      />
</sharing>
```

Lokale Rollen überschreiben

Soll z.B. die Rolle *Reviewer* an die Berechtigung *Modify portal content* geknüpft werden, so kann dies mit folgendem Eintrag geschehen:

```
<role
    id="Reviewer"
    title="Review submitted articles"
    permission="Modify portal content"
    i18n:attributes="title"
    />
```

Lokale Rollen löschen

```
<role
  remove="True"
  id="Reviewer"
  />
```

Hinzufügen einer Rolle zum *Freigabe*-Reiter

Um eine Rolle der Tabelle im *Freigabe*-Reiter hinzuzufügen kann einfach ein entsprechendes GenericSetup-Profil `sharing.xml` erstellt werden, z.B.:

```
<sharing xmlns:i18n="http://xml.zope.org/namespaces/i18n"
         i18n:domain="plone">
  <role
      id="Site Manager"
      title="Is a site coordinator"
      permission="Manage portal"
      i18n:attributes="title"
      />
</sharing>
```

The title is the name to be shown on the sharing page. The required_permission is optional. If given, the user must have this permission to be allowed to manage the particular role.

Siehe auch

- Local roles

Plone 3

1. Zunächst wird in der `permissions.py`-Datei die Rolle für die Ansicht im *Freigabe*-Reiter registriert und die Rolle angegeben, auf der unsere neue Rolle basieren soll:

```
from Products.CMFCore.permissions import setDefaultRoles
from AccessControl import ModuleSecurityInfo

security = ModuleSecurityInfo('vs.policy')

security.declarePublic('MyRole')
MyRole = 'Sharing page: My Role'
setDefaultRoles(MyRole, ('Reviewer',))
```

2. Dann wird die Rolle für die *Freigabe*-Seiten registriert in `localroles.py`:

```
from zope.interface import implements
from plone.app.workflow.interfaces import ISharingPageRole

import permissions

class ManagerRole(object):
    implements(ISharingPageRole)

    title = u'My Role'
    required_permission = permissions.MyRole
```

3. Schließlich wird in der `configure.zcml`-Datei die entsprechende Berechtigung angelegt:

```
<permission
    id="plone.MyRole"
    title="Sharing page: My Role"
    />
```

Arbeitsabläufe

Arbeitsabläufe (Workflows) erlauben, lokal die Rechte für Inhaltsobjekte zu verändern, ohne die Rechte für jedes dieser Objekte einzeln ändern zu müssen. Damit bleibt die Rechteverwaltung übersichtlich.

Die Arbeitsabläufe selbst werden für eine Site zentral im *Workflow Tool* (`portal_workflow`) verwaltet. Zunächst werden Sie den *Workflows*-Reiter sehen, in dem den verschiedenen Artikeltypen Arbeitsabläufe zugeordnet werden. Die Definitionen der Arbeitsabläufe lassen sich im *Contents*-Reiter anschauen. Jeder der Arbeitsabläufe besteht aus verschiedenen Stadien (*states*), wie z.B. *private* oder *published*, und Übergängen (*transitions*) zwischen ihnen.

Visualisierung von Workflows

Mit collective.workflowed gibt es einen Javascript-basierten graphischen Editor für Arbeitsabläufe, der in Plones Website-Konfiguration aufgerufen werden kann. Die dort editierten Workflows können anschließend im Generic Setup Tool exportiert und in das eigene Produkt integriert werden.

Workflows und Berechtigungen

Solche Übergänge können durch bestimmte Rechte, Rollen und Gruppen geschützt werden.

Anmerkung 1: Wie schon an früherer Stelle bemerkt, sollten die Sicherheitseinstellungen durch die Zuweisung entsprechender Rechte (*Permissions*) erfolgen, **nicht** durch die Zuweisung von Rollen oder Gruppen.

Für jedes Stadium können eine Reihe von Rechten angegeben werden, die für ein entsprechendes Inhaltsobjekt gelten. Die in einem Arbeitsablauf zu vergebenden Rechte werden im *Permissions*-Reiter für jeden Workflow angegeben:

Beachten Sie bitte, dass Änderungen an den Rechten keinen unmittelbaren Einfluss auf die Rechte bestehender Objekte haben. Hierzu müssen in *Plone-Konfiguration* → *Artikeltypen* zunächst die Artikeltypen angegeben werden, für die der Workflow geändert werden soll. Anschließend lassen sich Zuordnungen von alten auf neue Stadien treffen.

Neuen Arbeitsablauf erstellen

Wir wollen nun für unsere beiden Artikeltypen spezifische Arbeitsabläufe erstellen. Hierzu fügen wir in `src/vs.registration/vs/registration/profiles/default/` die Datei `workflows.xml` mit folgendem Inhalt hinzu:

```
<?xml version="1.0"?>
<object name="portal_workflow"
        meta_type="Plone Workflow Tool">
    <object name="registrant_workflow" meta_type="Workflow"/>
    <object name="registration_workflow" meta_type="Workflow"/>
    <bindings>
        <type type_id="Registrant">
            <bound-workflow workflow_id="registrant_workflow"/>
        </type>
        <type type_id="Registration">
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            <bound-workflow workflow_id="registration_workflow"/>
        </type>
    </bindings>
</object>
```

Anschließend werden die verschiedenen Stadien und Übergänge der neuen Arbeitsabläufe definiert. Dazu werden die Ordner `src/vs.registration/vs/registration/profiles/default/workflows/registrant_workflow/` und `src/vs.registration/vs/registration/profiles/default/workflows/registration_workflow/` erstellt. Der Name der Ordner muss dabei exakt der in `workflows.xml` angegebenen ID entsprechen. In jedem dieser Ordner wird dann die Datei `definition.xml` angelegt. Für den Artikeltyp `registrant` sieht sie z.B. so aus:

```
<?xml version="1.0"?>
<dc-workflow workflow_id="registrant_workflow"
             title="registrant_workflow"
             state_variable="review_state"
             initial_state="unconfirmed">
```

Zunächst werden allgemeine Angaben zum Arbeitsablauf wie ID, Titel, Variablenname und initialer Status gemacht. Der Variablenname `state_variable` sollte dabei immer `review_state` sein.

Anschließend werden die Rechte (*Permissions*) angegeben, die durch den Arbeitsablauf geändert werden sollen:

```
<permission>Delete objects</permission>
<permission>Modify portal content</permission>
<permission>View</permission>
```

Nun werden die verschiedenen Stadien definiert. Dabei wird für jedes Stadium in `exit-transition` angegeben, welche Übergänge möglich sind und eine Zuweisung der Rollen und Rechte vorgenommen:

```
<state state_id="confirmed"
       title="Confirmed">
    <exit-transition transition_id="reject-open"/>
    <permission-map name="Delete objects"
                    acquired="False">
        <permission-role>Manager</permission-role>
    </permission-map>
    <permission-map name="Modify portal content"
                    acquired="False">
        <permission-role>Owner</permission-role>
        <permission-role>Manager</permission-role>
    </permission-map>
    <permission-map name="View"
                    acquired="False">
        <permission-role>Manager</permission-role>
    </permission-map>
</state>
...
```

Den Übergängen werden ID, Titel und Auslöser (`trigger`) zugewiesen. `trigger` kann dabei die Werte `USER` oder `AUTOMATIC` annehmen. Der `<action />`-Tag enthält den Namen, der in Plone's *Status*-Menü angezeigt wird und die URL, auf die diese Aktion verlinkt. Üblicherweise wird hier das `content_status_modify`-Skript verwendet. Schließlich wird der Übergang noch geschützt durch die Angabe im `<guard />`-Tag:

```
    <transition transition_id="confirm"
                title="Confirm"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                new_state="confirmed"
                trigger="USER"
                before_script=""
                after_script="">
        <action url="%(content_url)s/content_status_modify?workflow_action=confirm"
                category="workflow">
            Confirm
        </action>
        <guard>
        </guard>
    </transition>
    ...
</dc-workflow>
```

Der Arbeitsablauf kann mittels `i18n:`-Attributen internationalisiert werden. Dabei besteht prinzipiell Zugriff auf alle verwendeten Zeichenketten. Siehe auch *Übersetzungen in der Plone-Domäne*.

Programmatische Änderung von Artikeln

Erhalten des aktuellen Stadiums Für Plone 4.3 kann das aktuelle Stadium mit plone.api ermittelt werden:

```
import plone.api
wftool = plone.api.portal.get_tool('portal_workflow')
review_state = wftool.getInfoFor(context, 'review_state')
```

Für Plone 4.2 erhalten wir das Workflow-Tool noch mit `getToolByName`:

```
from Products.CMFCore.utils import getToolByName
wftool = getToolByName(context, 'portal_workflow'
review_state = wftool.getInfoFor(context, 'review_state')
```

Bemerkung: Wird das Stadium im *Catalog Tool* (`portal_catalog`) abgefragt, so wird das Stadium als Metaangabe des Objekts ausgegeben:

```
from Products.CMFCore.utils import getToolByName
catalog = getToolByName(context, 'portal_catalog')
for result in catalog(portal_type = ('Document', 'News Item'),
                      review_state = ('published', 'public', 'visible',)):
    review_state = result.review_state
    # do something with the review state
```

Ändern des Stadiums

```
wftool.doActionFor(context, action='publish')
```

Rechte, Rollen und Arbeitsabläufe testen

```
def test_role_added(self):
    portal = self.layer['portal']
    self.assertTrue("StaffMember" in portal.validRoles())
def test_workflow_installed(self):
    portal = self.layer['portal']
    workflow = getToolByName(portal, 'portal_workflow')
    self.assertTrue('vs_sitecontent_workflow' in workflow)
def test_workflows_mapped(self):
    portal = self.layer['portal']
    workflow = getToolByName(portal, 'portal_workflow')
    self.assertEqual(('vs_sitecontent_workflow',),
        workflow.getDefaultChain())
def test_view_permisison_for_staffmember(self):
    portal = self.layer['portal']
    self.assertTrue('View' in [r['name']
        for r in portal.permissionsOfRole('Reader')
        if r['selected']])
    self.assertTrue('View' in [r['name']
        for r in portal.permissionsOfRole('StaffMember')
        if r['selected']])
def test_staffmember_group_added(self):
    portal = self.layer['portal']
    acl_users = portal['acl_users']
    self.assertEqual(1,
        len(acl_users.searchGroups(name='Staff')))
```

Protected und Trusted Code

Sicherheitsannahmen werden an den folgenden Stellen getroffen:

- `browser`-Komponenten wie `views` und `resource`, die in ZCML deklariert und mit einer Berechtigung versehen werden.
- *Page Templates* und andere Ressourcen, die durch das *Skins*-Tool verwaltet werden, werden explizit bestimmten Rollen zugewiesen. So wird z.B. `prefs_install_products_form.pt` in der assoziierten Datei `prefs_install_products_form.pt.metadata` explizit auf die `Manager`-Rolle eingeschränkt:

  ```
  [security]
  View = 0:Manager
  ```

- Attribute und Methoden von persistenten Objekten wie Artikeltypen und Tools können auf zweierlei Art geschützt werden:
  - durch ZCML-Berechtigungen mit `class`- oder `require`-Anweisungen
  - durch Python-Code, der ein `AccessControl.ClassSecurityInfo`–Objekt verwendet.

Zusätzlich kann die Variable `__allow_access_to_unprotected_subjects__` einer Klasse hinzugefügt werden um zu bestimmen, wie sich Attribute, die ihrerseits nicht durch Security-Annahmen geschützt sind, verhalten sollen.

Restricted Python

Zope erlaubt privilegierten Nutzern *Page Templates, DTML-Methoden und Python-Skripte *through-the-web* zu erstellen. *Restricted Python* gewährleistet nun, dass diese Nutzer nicht Skripte oder Templates erstellen können, die Zugang zu Ressourcen oder Methoden erlauben würden, die ihnen nicht zugestanden wurden. Die Berechtigungen werden automatisch überprüft und führen ggf. zu einer *Unauthorized*-Exception.

Es lassen sich sog. *Proxy Roles* für Templates oder Python-Skripte entweder im ZMI oder in einer `*.metadata`-Datei angeben. So gibt es z.B. für das *Controller Python Script* `send_feedback.cpy` eine korrespondierende Datei `send_feedback.cpy.metadata` mit folgendem Inhalt:

```
[default]
proxy=Manager,Anonymous
[security]
View=0:Authenticated
```

Dies ist notwendig, da das Skript normalerweise von Nutzern aufgerufen wird, die nicht auf die E-Mail-Konfiguration der Site zugreifen dürften, das Skript jedoch bestimmte angaben aus dieser Konfiguration benötigt.

Restricted Python gewährleistet ebenfalls, dass *through-the-web* erstellte Skripte nicht auf das Dateisystem zugreifen oder unautorisierte Module importieren können, die die Sicherheit des Servers kompromittieren könnten. Lediglich die von``AccessControl`` in `allow_module()` und `allow_class()` angegebenen Module und Klassen können importiert werden. Zusätzlich werden alle Methoden und Variablen, deren Namen mit einem Unterstrich _ beginnen, als *privat* betrachtet und können nicht aufgerufen werden.

Durch diese Sicherheitsmechanismen sollten Administratoren bei sachgemäßer Handhabung daran gehindert werden, versehentlich Sicherheitslücken in eine Anwendung zu reißen. Bei unsachgemäßer Handhabung kann jedoch weiterhin erheblicher Schaden angerichtet werden.

Nutzer programmatisch anlegen

Hierzu wird in der Datei `vs/policy/setuphandlers.py` folgendes angegeben:

```
def installUsers(site):

    site.portal_membership.addMember('me', 'secret', ['Member'], ())
    site.portal_membership.addMember('myself', 'secret', ['Editor', 'Member'], ())
    site.portal_membership.addMember('i', 'secret', ['Manager', 'Member'], ())

def setupVarious(context):

    if context.readDataFile('vs.policy_various.txt') is None:
       return


    ...
    installUsers(site)
```

Für das Anlegen mehrerer Nutzer stehen zwei Erweiterungen zur Verfüugung:

- atreal.usersinout
- collective.mass_subscriptions

Debugging

Um die Rollen eines Nutzers in einem bestimmten Kontext angezeigt zu bekommen, erstellen wir folgenden View.

Zunächst wird der View konfiguriert in `browser/configure.zcml`:

```
<browser:page
  name="debug-user"
  for="*"
  permission="zope2.View"
  class=".debug.DebugUser"
  />
```

Anschließend erstellen wir noch das Python-Skript `browser/debug.py`:

```
from Products.Five.browser import BrowserView
from AccessControl import getSecurityManager

class DebugUser(BrowserView):
    """ Current user debugging """

    def __call__(self):

        result = list()
        user = getSecurityManager().getUser()
        result.append('User: %s' % user.getUserName())
        result.append('Roles: %s' % user.getRoles())
        result.append('Roles in context: %s' % user.getRolesInContext(self.context))
        result.append('')
        result.append(self.request.text())
        return '\n'.join(result)
```

Nach einem Neustart der Instanz können Sie nun in jedem Kontext den View im Browser aufrufen mit `@@debug-user` und erhält neben der ID des Nutzers auch dessen globale und kontextabhängigen Rollen. Darüberhinaus erhalten Sie auch Informationen zur Session etc.

3.8 Formulare

Bisher haben wir nur Formulare erstellt, deren Inhalte von Archetypes in der ZODB gespeichert wurden. In diesem Abschnitt werden wir nun eigenständige Formulare programmatisch erstellen.

Bemerkung: Mit PloneFormGen gibt es ein Produkt, das die einfache Erstellung von Formularen über ein Web-Interface erlaubt. PloneFormGen und die erforderlichen Zusatzprodukte lassen sich einfach installieren indem unter `eggs` im `[buildout]`-Abschnitt `Products.PloneFormGen` eingetragen wird.

3.8.1 Generierte Formulare

Im Abschnitt Portlet erstellen haben wir bereits gesehen, wie mit `zope.formlib` aus einem Interface *add*- und *edit*-Formulare generiert wurden. Dies war jedoch nur ein Spezialfall, nun wollen wir die Formulargenerierung mit `zope.formlib` allgemeiner betrachten. Hierzu definieren wir zunächst einmal einen View in `browser/configure.zcml`:

```
<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
    name="enquiry"
    class=".enquiry.EnquiryForm"
    permission="zope2.View"
    />
```

Anschließend definieren wir das Formular vollständig in `browser/enquiry.py` – es ist kein zusätzliches *Page Template* erforderlich:

```
import re
from zope.interface import Interface
from zope import schema
from zope.formlib import form
from Products.Five.formlib import formbase
from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile
from Products.statusmessages.interfaces import IStatusMessage
from Acquisition import aq_inner
from Products.CMFCore.utils import getToolByName
from vs.registration import RegistrationMessageFactory as _

# Define a valiation method for email addresses
class NotAnEmailAddress(schema.ValidationError):
    __doc__ = _(u"Invalid email address")

check_email = re.compile(r"[a-zA-Z0-9._%-]+@([a-zA-Z0-9-]+.)*[a-zA-Z]{2,4}").match
def validate_email(value):
    if not check_email(value):
        raise NotAnEmailAddress(value)
    return True

MESSAGE_TEMPLATE = """\
Enquiry from: %(name)s <%(email_address)s>

%(message)s
"""

class IEnquiryForm(Interface):
    """Define the fields of our form
    """
    subject = schema.TextLine(title=_(u"Subject"),
                              required=True)
    name = schema.TextLine(title=_(u"Your name"),
                           required=True)
    email_address = schema.ASCIILine(title=_(u"Your email address"),
                                     description=_(u"We will use this to contact you␣
↪if you request it"),
                                     required=True,
                                     constraint=validate_email)
    message = schema.Text(title=_(u"Message"),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                            description=_(u"Please keep to 1,000 characters"),
                            required=True,
                            max_length=1000)
```

Die `constraint`-Eigenschaft des `email_address`-Feldes sollte bei einem Aufruf `True` zurückgeben, wenn der Wert gültig ist. Andernfalls wird die in `__doc__` angegebene Fehlermeldung ausgegeben.

Eine Übersicht über die verfügbaren Felder und deren Optionen erhalten Sie in `zope.schema.interfaces`.

Der View für das Formular wird dann abgeleitet aus der Basisklasse von `zope.formlib`, die in einem Zope2-Produkt mittels `Products.Five.formlib.formbase` zur Verfügung steht:

```
class EnquiryForm(formbase.PageForm):
    form_fields = form.FormFields(IEnquiryForm)
    label = _(u"Enquiry")
    description = _(u"Got a question? Please make an enquiry using the form below!")
```

- Die `form_fields`-Variable enthält die darzustellenden Felder. Hier können auch eigene Widgets hinzugefügt oder Felder aus der Liste ausgenommen werden. Weitere Informationen hierzu erhalten Sie in den Interfaces `IFormFields` und `IFormField` in `zope.formlib.interfaces`.
- `label` und `description` werden als Titel und Beschreibung oben auf der Seite dargestellt.

Mit `__call__` wird das Formular gerendert. Dabei ändern wir die übliche Darstellung indem der Rahmen und die Reiter beim Editieren nicht angezeigt werden:

```
def __call__(self):
    self.request.set('disable_border', True)
    return super(EnquiryForm, self).__call__()
```

Anschließend wird die einzige Taste dieses Formulars angegeben:

```
@form.action(_(u"Send"))
def action_send(self, action, data):
```

Beim Abschicken des Formulars wird die *Decorator*-Funktion aufgerufen und die Werte des Formulars in das Wörterbuch `data` geschrieben.

Anschließend wird die E-Mail konstruiert:

```
context = aq_inner(self.context)
mailhost = getToolByName(context, 'MailHost')
urltool = getToolByName(context, 'portal_url')
portal = urltool.getPortalObject()
email_charset = portal.getProperty('email_charset')

to_address = portal.getProperty('email_from_address')
source = "%s <%s>" % (data['name'], data['email_address'])
subject = data['subject']
message = MESSAGE_TEMPLATE % data

mailhost.secureSend(message, to_address, str(source),
                    subject=subject, subtype='plain',
                    charset=email_charset, debug=False,
                    From=source)
```

Und schließlich wird auf die Startseite weitergeleitet und dort eine Statusmeldung angezeigt:

```
confirm = _(u"Thank you! Your enquiry has been received and we will respond as soon␣
↪as possible")
IStatusMessage(self.request).addStatusMessage(confirm, type='info')

self.request.response.redirect(portal.absolute_url())
return ''
```

Eigene Widgets schreiben

Wir wollen nun am Beispiel eines Boolschen Feld zum Abonnieren eines Newsletters zeigen, wie eigene Widgets definiert werden können. Hierzu erweitern wir zunächst unsere Schemadefinition:

```
class IEnquiryForm(Interface):
    ...
    newsletter = schema.Bool(title=u'Subscribe to Newsletter?',
                             default=False,
                             required=True)
```

Nun können wir ein Widget definieren, wobei statt *True* und *False* die Werte *Yes* und *No* angezeigt werden sollen:

```
from zope.schema import vocabulary as schemavocabulary
from zope.app.form import browser as formbrowser

def YesNoWidget(field, request, true=_('yes'), false=_('no')):
    vocabulary = schemavocabulary.SimpleVocabulary.fromItems(((true, True),
                                                              (false, False)))
    return formbrowser.RadioWidget(field, vocabulary, request)
```

Dieses Widget wird schließlich in der EnquiryForm-Klasse dem `newsletter`-Feld zugewiesen:

```
class EnquiryForm(formbase.PageForm):
    form_fields = form.FormFields(IEnquiryForm)
    form_fields['newsletter'].custom_widget = YesNoWidget
    ...
```

Das Formular sollte nun so aussehen:

Bestehende Widgets verwenden

In `plone.app.form` stehen bereits diverse Widgets bereit:

`CheckBoxWidget` Plone-spezifisches Widget, das eine Checkbox links neben `label` anzeigt, mit dem die Felder `title`, `label` und `required` ausgeblendet werden können.

`DateComponents` Ein *View*, der einige Hilfsmehtoden für Datumswidgets bereitstellt.

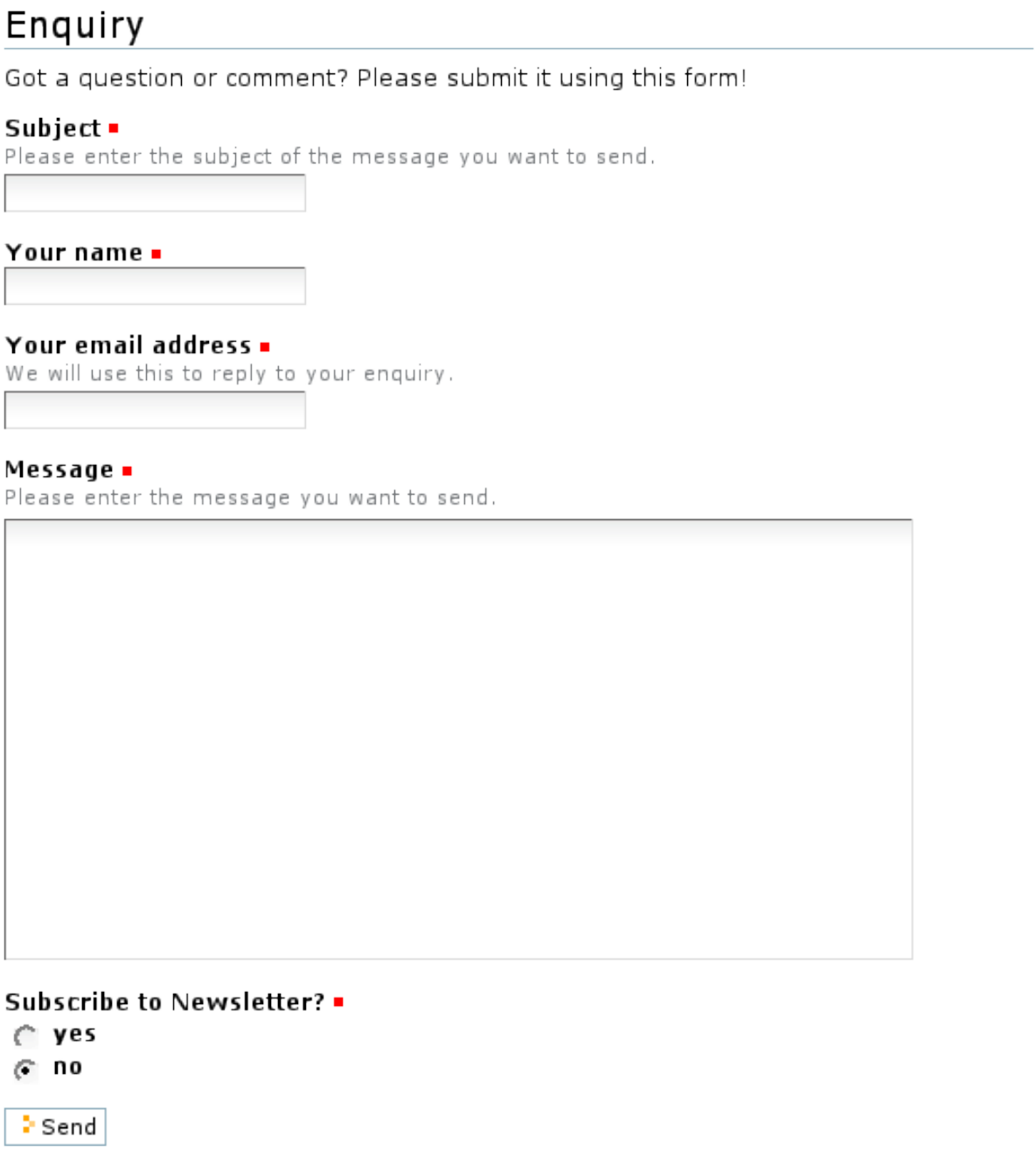
`LanguageDropdownChoiceWidget` Ein Dropdown-Widget, das eine lokalisierte Sprachauswahl darstellt.

`UberSelectionWidget` Widget zum Manipulieren von Auswahlfeldern.

Ein Proof of Concept findet sich in https://svn.plone.org/svn/plone/CMFPlone/branches/plip124-ueberselection-widget/

`WYSIWYGWidget` Widget für die Verwendung des WYSIWYG-Editors Kupu zum Editieren von Formularfeldern.

Um ein WYSIWYG-Feld in einem Formular anzuzeigen, kann z.B. in `enquiry.py` folgendes angegeben werden:


Enquiry
Got a question or comment? Please submit it using this form!
Subject ■
Please enter the subject of the message you want to send.
Your name ■
Your email address ■
We will use this to reply to your enquiry.
Message ■
Please enter the message you want to send.
Subscribe to Newsletter? ■
yes
no
Send

```
from plone.app.form.widgets.wysiwygwidget import WYSIWYGWidget
...
class IEnquiryForm(Interface):
...
    text = schema.Text(title=_(u"Text"),
                       description=_(u"A field which can contain rich text."),
                       required=True)
...
class EnquiryForm(base.EditForm):
    form_fields = form.FormFields(IEnquiryForm)
    ...
    form_fields['text'].custom_widget = WYSIWYGWidget
```

Weitere Widgets finden Sie auch in z3c.widget, u.a.:

Country selection Widgets Dropdown-Widget zur Auswahl eines Landes.

Date Selection Widget Das `DateSelectWidget`-Widget bietet drei Auswahlboxen für Tag, Monat und Jahr.

Flash Upload Widget Konfigurierbares Flash-Frontend

Image Widget Dieses Widget kann als `custom_widget` verwendet werden um Bilder hochzuladen.

Site action für unser Formular

Schließlich fügen wir in `vs.policy` noch eine *site action* hinzu. Hierzu erstellen wir die Datei `src/vs.policy/vs/policy/profiles/default/actions.xml` mit folgendem Inhalt:

```
<?xml version="1.0"?>
<object name="portal_actions" meta_type="Plone Actions Tool"
        xmlns:i18n="http://xml.zope.org/namespaces/i18n">
    <object name="site_actions"
            meta_type="CMF Action Category">
        <object name="contact"
                meta_type="CMF Action"
                i18n:domain="vs.registration">
            <property name="title" i18n:translate="">Enquiry</property>
            <property name="description" i18n:translate=""></property>
            <property name="url_expr">string:$portal_url/@@enquiry</property>
            <property name="icon_expr"></property>
            <property name="available_expr"></property>
            <property name="permissions">
                <element value="View"/>
            </property>
            <property name="visible">True</property>
        </object>
    </object>
</object>
```

Dabei wird mit `name="contact"` die Aktion, die auf das Plone-Kontaktformular verweist, ersetzt.

Bemerkung: In Forms erhalten Sie weitere Informationen zur `zope.formlib`.

3.8.2 Erstellen eigener Formulare

Um ein eigenes Formular zu erstellen, erzeugen wir zunächst ein Page Template `registrationreport` in `src/vs.registration/vs/registration/browser/` mit folgendem Inhalt:

```
<form method="get"
      tal:attributes="action string:${context/absolute_url}/${view/__name__}">

    <div i18n:translate="registration_report_days_searched">
        Show changes in the the last
        <input type="text" size="2" name="days"
            tal:attributes="value view/days"
            i18n:name="num_days" />
        days.
        <input type="submit" class="context" name="form.button.UpdateDays"
            value="Refresh"
            i18n:name="submit_button"
            i18n:attributes="value" />
    </div>

</form>
```

Dabei wird das `action`-Attribut dynamisch generiert um zu gewährleisten, dass immer dasselbe Formular im selben Kontext aufgerufen wird.

Anmerkung 1: Würde ein Skin-Template verwendet, müsste statt der Variablen `${view/__name__}` die Variable `${template/getId}` verwendet werden.

Das Formular wird verarbeitet, wenn der View in `registrationreport.py` mit der `__call__()`-Methode aufgerufen wird:

```
template = ViewPageTemplateFile('registrationreport.pt')

def __call__(self):
    # Hide the editable-border
    self.request.set('disable_border', True)

    # The number for days with a silent fallback on the default
    # if the input is invalid.
    try:
        self.days = int(self.request.get('days', 7))
    except ValueError:
        self.days = 7

    return self.template()
```

Anmerkung 2: Falls eine Anfrage nicht nur auf den View selbst zugreifen muss, ist `self.request` nicht mehr ausreichend; stattdessen ist dann die Anfrage zu akquirieren mit `request = context.REQUEST`.

Anmerkung 3: Für `HTTP POST`-Anfragen sollte statt `self.request` `self.request.form` angegeben werden, da dies versehentlich akzeptierte Variablen verhindert.

Die anderen Methoden des Views untersuchen `self.days` zum Erstellen der Such-Parameter:

```
def recently_modified_registrants(self):
    context = aq_inner(self.context)
    catalog = getToolByName(context, 'portal_catalog')
    results = []
    for r in catalog(object_provides=IRegistrant.__identifier__,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                    modified=dict(query=self.modified_after(), range='min'),
                    sort_on='modified',
                    sort_order='reverse',):
        results.append(dict(url=r.getURL(),
                            title=r.Title,
                            description=r.Description,
                            modified=self.localize(r.modified)))
    return results

def localize(self, time):
    return self._time_localizer()(time, None, aq_inner(self.context), domain=
↪'plonelocales')

def modified_after(self):
    return DateTime() - self.days

@memoize
def _time_localizer(self):
    context = aq_inner(self.context)
    translation_service = getToolByName(context, 'translation_service')
```

Mehrere `submit`-Tasten

Sind für ein Formular mehrere Tasten verfügbar, muss überprüft werden, welche der Tasten gedrückt wurde und welche Aktion hierfür auszuführen ist. Wie dies geschieht, können Sie sich z.B. in `plone.app.workflow.browser.sharing` anschauen:

```
<input class="context" type="submit" name="form.button.Save" value="Save"␣
↪i18n:attributes="value label_save" />
<input class="standalone" type="submit" name="form.button.Cancel" value="Cancel"␣
↪i18n:attributes="value label_cancel"/>
```

In `sharing.py` wird dann zunächst überprüft, welche Taste gedrückt wurde. Dabei ist zu beachten, dass Web-Browser immer nur den Wert für `name` derjenigen Taste senden, auf die geklickt wurde. Anschließend wird die entsprechende Aktion zuzuweisen:

```
class SharingView(BrowserView):

    # Actions

    template = ViewPageTemplateFile('sharing.pt')

    def __call__(self):
        """Perform the update and redirect if necessary, or render the page
        """

        postback = True

        form = self.request.form
        submitted = form.get('form.submitted', False)

        save_button = form.get('form.button.Save', None) is not None
        cancel_button = form.get('form.button.Cancel', None) is not None

        if submitted and not cancel_button:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            if not self.request.get('REQUEST_METHOD','GET') == 'POST':
                raise Forbidden

            # Update the acquire-roles setting
            inherit = bool(form.get('inherit', False))
            self.update_inherit(inherit)

            # Update settings for users and groups
            entries = form.get('entries', [])
            roles = [r['id'] for r in self.roles()]
            settings = []
            for entry in entries:
                settings.append(
                    dict(id = entry['id'],
                         type = entry['type'],
                         roles = [r for r in roles if entry.get('role_%s' % r,␣
↪False)]))
            if settings:
                self.update_role_settings(settings)

        # Other buttons return to the sharing page
        if cancel_button:
            postback = False

        if postback:
            return self.template()
        else:
            context_state = self.context.restrictedTraverse("@@plone_context_state")
            url = context_state.view_url()
            self.request.response.redirect(url)
```

Eingabekonverter

Aus dem `sharing.pt`-*Page Template* lässt sich auch ablesen, wie Formularfelder berechnet werden können:

```
<input
    type="hidden"
    name="entries.id:records"
    tal:attributes="value entry/id"
    />
<input
    type="hidden"
    name="entries.type:records"
    tal:attributes="value entry/type"
    />
```

Dies wird dargestellt in einer `tal:repeat`-Schleife. Wird das Formular abgeschickt, wird die Variable `entries` mit einer Liste von Wörterbüchern aus den Schlüsselwörtern `"id"` und `"type"` übertragen. Ein Überblick über alle verfügbaren Konverter erhalten Sie hier:

Konverter	Beispiel	Anmerkung
`boolean`, `int`, `long`, `float`, `string`, `date`, `boolean`	`<input type="hidden" name="limit:int" value="8 />"`	Wandelt die Variable in den zugehörigen Python-Typ um, `date` führt so zu `DateTime`. Solche Umwandlungen sind normalerweise nur in `hidden`-Feldern sinnvoll; werden unzulässige Werte eingegeben, ist die resultierende Fehlermeldung für die meisten Nutzer wenig aussagekräftig.
`text`	`<textarea name="description:text" />`	Konvertiert eine Zeichenkette mit normalisierten Zeilenumbrüchen entsprechend der Server-Plattform
`list`, `tuple`	`<input type="checkbox" name="status:list" value="1"`	Erstellt eine Liste oder ein Tuple aus mehreren Feldern mit demselben Namen oder aus einer mehrwertigen Listenauswahl. Dieser Konverter kann mit anderen kombiniert werden, z.B. `int:list` um eine Liste ganzer Zahlen zu erhalten
`tokens`, `lines`	`<input type="text" name="keywords:tokens" />`	Wandelt eine durch Leerzeichen (tokens) oder neue Zeilen (lines) getrennte Zeichenkette in eine Liste um
`record`, `records`	`<input type="text" name="data.id:record" />`	Erstellt ein Wörterbuch (`record`) oder eine Liste von Wörterbüchern (`records`). Der Name vor `.` ist der Variablenname, der Name danach der Schlüssel.
`required`	`<input type="text" name="title:required" />`	Gibt eine Fehlermeldung aus wenn das Feld nicht ausgefüllt wurde.
`ignore_empty`	`<input type="text" name="id:ignore_empty" />`	Die Variable wird bei einem Request nicht angegeben wenn sie leer ist. Dieser Konverter kann mit anderen kombiniert werden.
`default`	`<input type="hidden" name="accept:boolean:default" value="True" /><input type="checkbox" name="accept:boolean:default" value="False" />`	Standardwert, falls in keinem Feld mit demselben Namen ein Wert übermittelt wurde. Dies ist vor allem für Checkboxen sinnvoll, die nicht übertragen werden wenn für sie keine Angabe gemacht wurde. Dieser Konverter kann mit anderen kombiniert werden.

3.8.3 Form Controller Tool

Plone kommt mit dem CMF Form Controller-Produkt, mit dem die Abläufe zwischen Formularen und Skripten geregelt werden können. Gerade für komplexe Abläufe ist es sehr hilfreich, unterstützt jedoch keine Zope3-Views und kann daher nur in Skin-Layern definiert werden.

Schauen wir uns nun als Beispiel Plone's *Send this page to someone*-Formular an, das in `CMFPlone/skins/plone_forms/sendto_form.cpt` definiert ist. Dabei steht der `cpt`-Suffix für *Controller Page Template*:

```
<div metal:fill-slot="main"
     tal:define="errors options/state/getErrors;">
  ...
  <form name="sendto_form"
        class="enableAutoFocus"
        action="sendto_form"
        method="post"
        enctype="multipart/form-data"
        tal:attributes="action string:$here_url/$template_id">
    ...
    <div class="field"
         tal:define="error errors/send_to_address|nothing;"
         tal:attributes="class python:test(error, ``field error``, ``field``)">
      ...
      <div class="formControls">
        <input class="context"
               type="submit"
               name="form.button.Send"
               value="Send"
               i18n:attributes="value label_send;"
               />
    </div>
    ...
  </form>
```

- Zunächst fällt auf, dass eine Variable `errors` definiert wird, die das Auffinden von Validierungsfehlern erlaubt.
- Dann sehen wir, dass das Formular – wie bei CMFFormController üblich – wieder auf sich selbst verweist.
- Schließlich erkennen wir die versteckte Variable `form.submitted`, wobei das *Controller Page Template* überprüft, ob das Formular einfach aufgerufen oder bereits abgeschickt wurde.

CMFFormController benötigt zur Auswertung des Formulars eine korrespondierende Datei `sendto.cpy.metadata` im selben Verzeichnis:

```
[default]
title=Send this page to somebody

[validators]
validators=validate_sendto

[actions]
action.success=traverse_to:string:sendto
action.failure=traverse_to:string:sendto_form
```

Schauen wir uns nun die Validatoren und Aktionen genauer an.

Validatoren angeben

Allgemein lassen sich für Für *Controller Page Templates* folgendermaßen Validatoren angeben:

```
[validators]
validators = validate_script1, validate_script2
```

Diese Angabe startet zwei Prüfskripte: zunächst *validate_script1* , dann `validate_script2` . Ein Prüfskript untersucht die Formulardaten wobei Fehler dem Form Controller Status hinzugefügt werden.

Objekttyp-spezifische Validierung

Soll die Validierung vom Objekttyp *Document* verschieden sein von der des Objekttyps *Image* sieht die Metaangabe so aus:

```
validators.Document = validate_script1
validators.Image = validate_script2
```

Button-spezifische Validierung

Kommen im Formular mehrere Buttons (Tasten) vor, z.B.:

```
<input type="submit" name="form.button.button1" value="Value1" />
<input type="submit" name="form.button.button2" value="Value2" />
```

können für diese auch unterschiedliche Validierungen angegeben werden:

```
validators.button1 = validate_script1
validators.button2 = validate_script2
```

Aktionen angeben

Für *Controller Page Templates* lassen sich auch Aktionen angeben, z.B.:

```
[actions]
action.success = traverse_to:string:script1
```

Haben die Prüfskripte den Status `success` ausgegeben, wird die Aktion `traverse_to` mit dem Argument `string:script1` aufgerufen.

Schlägt ein Prüfskript fehl, wird gemäß den Standardeinstellungen das Formular erneut geladen.

Wie bei Validatoren kann auch bei Aktionen zwischen Dokumenttypen und Buttons unterschieden werden:

```
action.success.Document = traverse_to:string:document_script
action.success.Image = traverse_to:string:image_script
```

```
action.success.button1 = traverse_to:string:script1
action.success.button2 = traverse_to:string:script2
```

Folgende Aktionen sind möglich:

- `redirect_to`

- redirect_to_action
- traverse_to
- traverse_to_action.

Dabei rufen die traverse_to-Aktionen direkt ein Template oder Skript auf dem Server auf, wohingegen die redirect_to-Aktionen eine Weiterleitung des Browsers bewirken. Normalerweise werden die Zwischenschritte mit traverse_to-Aktionen und nur der letzte Schritt mit einer redirect_to-Aktion angegeben, sodass die Angabe der URL im Browser die aktuelle Seite wiedergibt. So ist z.B. in unserem Beispiel in sendto.cpy.metadata folgendes angegeben:

```
[validators]
validators=validate_sendto

[actions]
action.success = redirect_to_action:string:view
action.failure = redirect_to_action:string:view
```

Validator-Skripte schreiben

Schauen wir uns nun das Validator-Skript validate_sendto.vpy genauer an, auf das in sendto.cpy.metadata verwiesen wurde:

```
## Controller Script Python "validate_sendto"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind state=state
##bind subpath=traverse_subpath
##parameters=send_to_address='',send_from_address=''
##title=validates the email adresses

from Products.CMFPlone import PloneMessageFactory as _
plone_utils=context.plone_utils

if not send_to_address:
    state.setError('send_to_address', _(u'Please submit an email address.'), 'email_
↪required')
...
if state.getErrors():
    context.plone_utils.addPortalMessage(_(u'Please correct the indicated errors.'),
↪'error')
    return state.set(status='failure')
else:
    return state
```

Aktionen schreiben

Ist die Validierung erfolgreich, fährt der CMFFormController, wie in `sendto_form.cpt.metadata` angegeben mit dem Skript `sendto.cpy` fort. Dieses Skript gibt schließlich den Wert für `state` aus:

```
## Controller Python Script "sendto"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind state=state
##bind subpath=traverse_subpath
##parameters=
##title=Send an URL to a friend
##
REQUEST=context.REQUEST

...

if not mtool.checkPermission(AllowSendto, context):
    context.plone_utils.addPortalMessage(_(u'You are not allowed to send this link.'),
↪ 'error')
    return state.set(status='failure')

...

context.plone_utils.addPortalMessage(_(u'Mail sent.'))
return state
```

3.9 Internationalisierung

3.9.1 Spracheinstellungen

Das Plone User Interface ist bereits in viele Sprachen übersetzt worden. Die Übersetzungen werden jedoch nicht mit Plone selbst mitgeliefert, sondern in einem eigenen Modul:

- Bis zu Version Plone 3.1 in Plone Translations,
- ab Version Plone 3.3 in plone.app.locales.

Üblicherweise erhalten Sie je nach Spracheinstellung ihres Browsers die entsprechende Sprache für das Plone-Interface angezeigt.

Da vermutlich jedoch die wenigsten Betrachter ihrer Website bewusst die Spracheinstellung in ihrem Browser vorgenommen haben, gibt es mit dem *Plone Language Tool* ein Produkt, das vielfältige Spracheinstellungen für eine Plone-Sie ermöglicht.

Plone Language Tool

Das Plone Language Tool kann, nachdem es im *Products*-Ordner ihrer Zope-Instanz installiert wurde, in der Plone-Konfiguration unter *Produkte hinzufügen/entfernen* für jede Plone-Site installiert werden.

Anschließend können sie folgende Spracheinstellungen vornehmen:

Standardsprache [engl.: Default language) Die hier ausgewählte Sprache wird verwendet sofern die vom Nutzer bevorzugte Sprache nicht verfügbar ist.

Erlaubte Sprachen (engl.: Allowed languages) Hier können sie diejenigen Sprachen angeben, die in ihrer Website ausgewählt werden können.

Verwenden Sie die *ctrl*-Taste zum Aus- und Abwählen der einzelnen Sprachen.

Erlauben spezifischer Sprachvarianten wie `de_DE`, `de-AT` und `de-CH`.

Anzeigen von Flaggen für die Sprachauswahl.

Verhandlungsschema (engl.: *Negotiation Scheme*) Die Reihenfolge, in der die Sprache ausgehandelt wird.

1. Sollen Sprachcodes in der URL für Sprachangaben des Betrachters verwendet werden?
2. Sollen Cookies für die Sprachangaben des Betrachters verwendet werden?
3. Soll die Spracheinstellung des Browsers berücksichtigt werden?
4. Die voreingestellte Standardsprache (s.o.) wird verwendet.

Kontrolleinstellungen für die Inhalte Einstellungen, die auf mehrsprachigen **Inhalt** zutreffen.

1. Erzwinge unterschiedliche URL's für jede Sprache (Umleitung).
2. Erlaube, auf eine andere Sprache auszuweichen, sollte dieses von der Implementierung unterstützt werden.
3. Erstelle Artikel anfänglich als sprachneutral.

Dies Einstellungen können auch im Profil `profiles/default/portal_languages.xml` vorgenommen werden:

```
<?xml version="1.0"?>
<object name="portal_languages" meta_type="Plone Language Tool">
 <default_language value="en"/>
 <use_content_negotiation value="False"/>
 <use_path_negotiation value="False"/>
 <use_cookie_negotiation value="True"/>
 <authenticated_users_only value="False"/>
 <use_request_negotiation value="False"/>
 <use_cctld_negotiation value="False"/>
 <use_subdomain_negotiation value="False"/>
 <use_combined_language_codes value="False"/>
 <display_flags value="False"/>
 <start_neutral value="False"/>
 <supported_langs>
  <element value="en"/>
  <element value="de"/>
 </supported_langs>
</object>
```

Testen

```
def testLanguageSettings(self):
    default_language = self.portal.portal_languages.getDefaultLanguage()
    self.assertEqual(default_language == 'de', True)
    # return [(country code, countryname), ...]
    supported_languages = [r[0] for r in self.portal.portal_languages.
↪listSupportedLanguages()]
    self.assertEqual('en' in supported_languages, True)
    self.assertEqual('de' in supported_languages, True)
```

Internationalisieren des User-Interfaces

Plone nutzt den Placeless Translation Service um das User-Interface zu übersetzen. Ist das Produkt installiert, werden alle verfügbaren Übersetzungsdateien in `/Control_Panel/TranslationService/manage_main` angezeigt. Dabei schaut der Placeless Translation Service nach Übersetzungsdateien in `i18n`- und `locales`-Ordnern innerhalb von `INSTANCE_HOME` und `INSTANCE_HOME/Products`.

Internationalisieren von Page Templates

Domäne

Für jedes Page Template kann im HTML-Header die Standard-Übersetzungsdomäne angegeben werden, z.B.:

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      i18n:domain="vs.registration">
```

Existiert jedoch für eine Zeichenkette bereits eine Übersetzung in einer anderen Domain, verweisen Sie auf diese, z.B.:

```
<a href=""
   tal:attributes="href python:test(item_type in use_view_action, item_url+'/view',␣
↪item_url);"
   i18n:translate="read_more">
    Read More&hellip;
</a>
```

Inhalte

Text wird mit einer bestimmten *message id*, hier `read_more`, verknüpft um übersetzt werden zu können. Würde die *message id* leer sein, also `i18n:translate=""`, dann wird stattdessen der zu übersetzende Text selbst, hier `Read More`, verwendet. Dabei sind für die Präfixe der *message ids* in Plone bestimmte, unten genannte Regeln festgelegt worden.

Attribute

Um Attribute zu übersetzen wird nicht `i18n:translate` sondern `i18n:attributes` verwendet. Auch können mehrere Attribute gleichzeitig adressiert werden, wie z.B. in:

```
<img src="plone.gif"
     alt="Plone Icon"
     title="Plone Icon Title"
     i18n:attributes="alt vs_registration_plone_icon;
                      title vs_registration_plone_icon_title">
```

Dynamische Inhalte

Mit `i18n:name` lassen sich auch dynamische Inhalte übersetzen, z.B.:

```
<p i18n:translate="text_download">There have been
    <span tal:content="here/download_count"
          i18n:name="count">100.000</span>
    downloads of Plone. </p>
```

Der Eintrag in die Übersetzungsdatei sieht dann folgendermaßen aus:

```
msgid "text_download"
msgstr "There have been ${count} downloads of Plone."
```

Für Datum und Zeit wird die `localized_time`-Methode mit den zwei message ids `date_format_long` und `date_format_short` verwendet. Gibt es keine Übersetzungen für eine Sprache wird das Standardformat `strftime`, wie in den `portal_properties` angegeben, verwendet. Siehe auch Übersetzen des User-Interfaces: Datum und Urzeit.

Verschachtelungen

Message ids lassen sich auch verschachteln, z.B.:

```
<p i18n:translate="contentrules_controlpanel_link">
    Use the
    <a i18n:name="controlpanel_link"
       i18n:translate="contentrules_control_panel"
       tal:attributes="href string:${portal/absolute_url}/@@rules-controlpanel">
        content rules control panel
    </a>
    to create new rules or delete or modify existing ones.
</p>
```

In der Übersetzungsdatei findet die Übersetzung in verschiedenen Abschnitten statt:

```
msgid "contentrules_controlpanel_link"
msgstr "Benutzen Sie ${controlpanel_link} um neue Regeln zu erstellen, zu löschen␣
↪oder zu modifizieren."

msgid "contentrules_control_panel"
msgstr "die Regeleinstellungen"
```

Internationalisieren von Pythonskripten

In Pythonskripten sollten eigentlich keine Angaben zum User Interface stehen. Meist sind jedoch zumindest `label` und `description` von Widgets in den Pythonskripten selbst enthalten. Um diese lokalisieren zu können, werden gegebenenfalls `label_msgid` und `description_msgid` eingefügt, z.B.:

```
StringField('event_type', vocabulary='EventTypes',
    widget=SelectionWidget(
        label='Event Type',
        label_msgid='label_event_type',
        description='The type of the event',
        description_msgid='help_event_type',
        i18n_domain='vs.registration'),),
```

Seit Plone 2.5 kann die *Zope Message Factory* in Pythonskripten verwendet werden. Hierzu wird zunächst in der `__init__.py`-Datei des Pakets die *Message Factory* registriert:

```
from zope.i18nmessageid import MessageFactory
RegistrationMessageFactory = MessageFactory('vs.registration')
```

Anschließend lässt sich diese *Message Factory* in ein Pythonskript importiert mit:

```
from vs.registration import RegistrationMessageFactory as _
```

Und nun lassen sich folgende Angaben einfach übersetzen:

```
label=_(u"Body Text"),
description=_(u"Text for front page of registration")
```

Mit dem `u`-Präfix wird Unicode als Kodierung für die Zeichenketten festgelegt. Dieses Beispiel wird in einem Page Template weiterverwendet werden (z.B. in `tal:replace` oder `tal:content`).

Falls das Pythonskript nicht in einem Page Template verarbeitet wird, muss `translation_service` direkt aufgerufen werden:

```
from Products.CMFCore.utils import getToolByName
...
translation_service = getToolByName(self, 'translation_service')
value = u'John Doo'
return translation_service.utranslate('plone',
                                      u'My name is ${fullname}',
                                      mapping={u'fullname' : value})
```

Schließlich können auf diese Weise die Titel und Aktionen von Inhaltstypen in einer eigenen Produktdomäne verwaltet werden – es wird keine zusätzliche `.pot`-Datei für die Plone-Domäne benötigt.

Siehe auch Translating text in code.

Internationalisieren von GenericSetup-Profilen

Verschiedene Angaben in den `.xml`-Dateien des `profiles`-Ordner können ebenfalls lokalisert werden, z.B. `src/vs.registration/vs/registration/profiles/default/types/Registration.xml`:

```
<object name="Registration"
        meta_type="Factory-based Type Information with dynamic views"
        i18n:domain="vs.registration"
        xmlns:i18n="http://xml.zope.org/namespaces/i18n">
    <property name="title"
              i18n:translate="">Registration</property>
```

Standardisierte Präfixe

In Plone werden bestimmte Präfixe für *message ids* verwendet. Da die IDs später alphabetisch sortiert werden, lassen sich semantische Differenzen leichter erkennen.

`heading_` für `<h>`-Elemente.

`description_` erläuternder Text direkt unterhalb von `<h1>`.

`legend_` verwendet für `<legend>`-Elemente.

`label_` Für `label` von `input`- und `textarea`-Felder sowie `<a>`-Elemente.

`help_` für Hilfetexte von `input`-Fledern.

`box_` für die Inhalte von Portlets.

`listingheader_` für `header`-Angaben in Tabellen (üblicherweisse in der Klasse``listing``).

`date_` für datumsspezifische Angaben, wie z.B. *Gestern*, *letzte Woche*.

`text_` Nachrichten, die keiner anderen Kategorie zugeordnet werden konnten, üblicherweise innerhalb von `<p>`.

`batch_` für Stapeldarstellungen wie *X bis Y von Z.*

`summary_` für Zusammenfassungen in Tabellen.

`title_` für Titel aller Elemente.

`message_` für Text in `portal_status_message`

Den Präfixen folgt nach dem Unterstrich eine kurze Beschreibung der Nachricht, wie z.B. *label_address*.

Siehe auch: Guide to Prefixes.

Tipps & Tricks

Vermeiden sie aufwändige Differenzierungen

Es ist möglich, z.B. zwischen Singular und Plural zu unterscheiden:

```
<p i18n:translate="">
  Cart has <tal:block replace="number">#</tal:block>
  book<tal:block condition=
      "python: number <> 1">s</tal:block>.</p>
```

Dies macht jedoch die Arbeit der Übersetzer sehr schwierig, da in manchen Sprachen der Plural nicht einfach durch ein bis zwei angehängte Buchstaben gebildet wird.

Schließen sie die gesamte Phrase ein

Schließen sie die gesamte Phrase einschließlich der Satzzeichen in die zu übersetzende *message id* ein.

Teilen sie Sätze nicht in zwei verschiedene *message id* auf

Verwenden sie für den gesamten Satz den `i18n:translate`-Tag (s.o.) und verwenden `i18n:name` für den eingeschlossenen Teil.

jarn.jsi18n

jarn.jsi18n bietet eine i18n-Infrastruktur für Javascripts in Plone.

Im Einzelnen bietet es

- das einfache Laden von `gettext`-Katalogen aus Plone
- Message Factories, die denen in Python sehr ähnlich sind
- die Verweundung von *local storage* moderner Browser um das wiederholte Laden der *Message Cataologs* zu vermeiden.

Installation und Aktivierung

Ihr Paket sollte abhängen von `jarn.jsi18n`. Hierzu wird in der `setup.py`-Datei unter `install_requires` folgendes eingetragen:

```
install_requires=[
    'setuptools',
    'jarn.jsi18n'
],
```

Für die Aktivierung auf der Plone-Site wird es noch in die `profiles/default/metadata.xml` eingetragen:

```
<?xml version="1.0"?>
<metadata>
  ...
  <dependencies>
    ...
    <dependency>profile-jarn.jsi18n:default</dependency>
  </dependencies>
</metadata>
```

`MessageFactory`

Zum Instanziieren und Verwenden Um einer `MessageFactory` muss der `i18n`-Katalog geladen werden. Dies kann z.B. so geschehen:

```
$(document).ready(function () {
    jarn.i18n.loadCatalog('plone', 'de');
    _ = jarn.i18n.MessageFactory('plone')
});
```

Der zweite Parameter in `loadCatalog` spezifiziert die Sprache. Diese Angabe ist optional wenn das `lang`-Attribut im HTML-Tag verwendet wird.

Übersetzungen

Nachdem wir nun eine *Message Factory* haben, können wir auch Zeichenketten übersetzen:

```
> _('contribute');
hinzufügen
```

oder mit Keyword-Parametern:

```
> _('Groups are: ${names}', {names: 'staff'})
"Gruppen sind: Mitarbeiter"
```

Ggf. können auch mehrere Kataloge oder mehrere Sprachen für denselben Katalog geladen und instanziiert werden, z.B.:

```
> jarn.i18n.loadCatalog('plone', 'en');
> _en = jarn.i18n.MessageFactory('plone', 'en');
> _en('Contributor');
"Contributor"
```

Caching

Sofern der Browser *local storage* unterstützt, werden die Kataloge lokal gespeichert. Somit lässt sich vermeiden, dass nicht jedesmal ein Ajax-Request ausgelöst wird um den Katalog erneut zu laden. Der gespeicherte Katalog ist üblicherweise für 24 Stunden gültig. Die Lebensdauer kann ggf. geändert werden mit `jarn.i18n.setTTL(86400000)` wobei der Wert in Millesekunden angegeben wird.

Erstellen der Übersetzungsdateien

Zum Erstellen des GNU gettext-*message catalogs* verwenden wir i18ndude.

`i18ndude`-Installation

`i18ndude` lässt sich am einfachsten als eigenständiges Buildout-Projekt installieren um Versionskonflikte in den Abhängigkeiten zu vermeiden. Hierzu erhält die `buildout.cfg`-Datei ausschließlich den Abschnitt `scripts`:

```
[buildout]
parts =
    scripts

eggs =
    i18ndude

[scripts]
recipe = zc.recipe.egg
eggs = i18ndude
```

Erstellen der .pot-Datei

Die .pot-Datei ist ein Template, aus dem die sprachspezifischen Übersetzungsdateien abgeleitet werden. Im Template tragen üblicherweise die msgid-Zeilen Strings aus Page Templates (*.pt) und Python-Dateien (*.py), die msgstr sind leer. Strings aus .xml-Dateien werden über die passende Angabe i18n:domain im Header erfasst. Um Fehlermeldungen zu vermeiden erstellen wir zuerst die benötigten Verzeichnisse.

```
$ mkdir -p src/vs.registration/vs/registration/locales/de/LC_MESSAGES
$ cd /home/veit/i18ndude_buildout
$ ./bin/i18ndude rebuild-pot --pot /home/veit/myproject/src/vs.registration/vs/
↪registration/locales/vs.registration.pot --create vs.registration /home/veit/
↪myproject/src/vs.registration/vs/registration
```

Mit der Option rebuild-pot --pot gibt man die Datei an, in die das neue Template geschrieben wird. Mit dem zweiten Parameter --create gibt man zuerst die Domäne und danach das Verzeichnis an, das rekursiv nach relevanten Dateien durchsucht wird.

Weitere Optionen liefert der Aufruf von ./bin/i18ndude --help.

Erstellen der .po-Datei

Die .po-Dateien enthalten die Übersetzungen in den msgstr-Zeilen. Diese werden für die Anzeige der Übersetzung verwendet. Mit dem folgenden Befehl werden sie erstellt und in src/vs.registration/vs/registration/locales/de/LC_MESSAGES/ abgelegt. Um Fehlermeldungen zu vermeiden erstellen wir zuerst eine leere .po-Datei.

```
$  touch src/vs.registration/vs/registration/locales/de/LC_MESSAGES/vs.registration.po
$ ./bin/i18ndude sync --pot src/vs.registration/vs/registration/locales/vs.
↪registration.pot src/vs.registration/vs/registration/locales/de/LC_MESSAGES/vs.
↪registration.po
```

Der Befehl liefert beim ersten Durchlauf eine Rückmeldung wie z.B. diese:

```
src/vs.registration/vs/registration/locales/de/LC_MESSAGES/vs.registration.po: 31␣
↪added, 0 removed
```

Sollte eine Fehlermeldung besagen, dass die Zieldatei nicht existiert, dann wurde womöglich der XML-Namespace i18n nicht verwendet oder die Eigenschaft i18n:translate nicht korrekt verwendet.

Anmerkung 1: Werden die Dateien erneut synchronisiert, werden auch die Kommentare verglichen. Dabei wird ein fuzzy-Kommentar hinzugefügt sofern sich die msgid geändert hat. Diese Angaben sollten dann überprüft werden.

Anmerkung 2: Kommen in einer msgid URLs vor, die in Anführungszeichen (") gesetzt sind, so müssen diese *escaped* werden mit \, also z.B. www.veit-schiele.de.

Aktualisieren bestehender Übersetzungen

Um die Übersetzungen zu aktualisieren, etwa wenn sich die Zahl der Strings verändert hat, wird zunächst die `.pot`-Datei wie oben beschrieben aktualisiert. Anschließend wird die deutsche Übersetzungsdatei mit diesem Befehl aktualisiert:

```
$ ./bin/i18ndude sync --pot src/vs.registration/vs/registration/locales/vs.
↪registration.pot src/vs.registration/vs/registration/locales/de/LC_MESSAGES/vs.
↪registration.po
```

Skripte zum Aktualisieren der Übersetzungen

Um nicht bei jeder Aktualisierung erneut die obigen Shell-Kommandos eingegeben werden müssen, kann auch ein Shell-Skript angelegt werden, z.B. `rebuild.sh` mit folgendem Inhalt:

```
#!/usr/bin/env bash
./bin/i18ndude rebuild-pot --pot src/vs.policy/vs/policy/locales/vs.policy.pot --
↪create "vs.policy" --merge src/vs.policy/vs/policy/locales/vs.policy-manual.pot src/
↪vs.policy*
./bin/i18ndude rebuild-pot --pot src/vs.policy/vs/policy/locales/plone.pot --create
↪"plone" --merge src/vs.policy/vs/policy/locales/plone-manual.pot src/vs.policy/vs/
↪policy/profiles/
./bin/i18ndude sync --pot src/vs.policy/vs/policy/locales/vs.policy.pot src/vs.policy/
↪vs/policy/locales/de/LC_MESSAGES/vs.policy.po
msgfmt --no-hash -o src/vs.policy/vs/policy/locales/de/LC_MESSAGES/vs.policy.mo src/
↪vs.policy/vs/policy/locales/de/LC_MESSAGES/vs.policy.po
./bin/i18ndude sync --pot src/vs.policy/vs/policy/locales/plone.pot src/vs.policy/vs/
↪policy/locales/de/LC_MESSAGES/plone.po
msgfmt --no-hash -o src/vs.policy/vs/policy/locales/de/LC_MESSAGES/plone.mo src/vs.
↪policy/vs/policy/locales/de/LC_MESSAGES/plone.po
```

Sortierung der Einträge in `.po`-Dateien

`i18ndude` sortiert die Übersetzungen alphabetisch nach der `msgid`. Bei umfangreichen Übersetzungen, die sich bezüglich der Anzahl der Strings nicht mehr ändern, kann es sinnvoll sein diese nach der Herkunftsdatei zu sortieren, wie von gettext empfohlen. Fertige `.po`-Dateien lassen sich mit `gettext` entsprechend bearbeiten, das Paket stellt hierfür den Befehl `msgcat` und den Parameter `--sort-by-file` zur Verfügung. Siehe die Dokumentation von GNU gettext.

Übersetzen des User-Interfaces

Übersetzungen können nicht nur zur Übersetzung in völlig andere Sprachen verwendet werden, sondern auch z.B. zur Unterscheidung von österreichischen und deutschen Redewendungen. Desweiteren ermöglichen sie auch eine standardisierte Terminologie bei der Verwendung von Zusatzprodukten.

Zope und Plone nutzen GNU gettext und dessen *message catalogs*, eine Liste übersetzter Texte.

1. Wie eine sprachspezifische `.po`-Datei erstellt wird ist bereits in Erstellen der *.po-Datei beschrieben.
2. Als Werkzeug zum Übersetzen können Sie poEdit oder einen einfachen Texteditor verwenden. Bei einem Texteditor sollte `utf-8` als Zeichenkodierung verwendet werden.
3. Jeder Abschnitt gliedert sich üblicherweise in folgende Angaben:

#: Eine Zeile, die mit diesen Zeichen beginnt, gibt das Template an, in dem die Zeichenkette verwendet wird. Kommt eine Zeichenkette in mehreren Templates vor, wird erhält jedes Template eine eigene Zeile.

#. Zeilen, die mit diesen Zeichen beginnen, geben den Abschnitt an, dem die Zeichenkette entstammt.

msgid Diese Zeile gibt die exakte Zeichenkette des zu übersetzenden Textes an. Variablen wie `${foo}` sind in die Übersetzung ohne Veränderung zu übernehmen.

msgstr Diese Zeile enthält den übersetzten Text.

Bevor sie nun mit dem Übersetzen beginnen, sollten sie sich noch die grundlegenden Begriffe anschauen, die Plone in den Language Specific Terms festlegt. Produkte, die auf Plone aufsetzen, sollten diese Begriffe übernehmen um konsistente Bezeichnungen zu gewährleisten. Daher empfiehlt sich auch für Ihr Produkt eine solche Liste spezifischer Begriffe.

Datum und Urzeit

Datum und Zeit werden im `strftime`-Format lokalisiert, z.B.:

```
msgid "date_format_long"
msgstr "${Y}-${m}-${d} ${H}:${M}"

msgid "date_format_short"
msgstr "${Y}-${m}-${d}"
```

Anweisung	Beschreibung
`%a`	Lokalisierter abgeküzter Name des Wochentags.
`%A`	Lokalisierter Wochentag.
`%b`	Lokalisierter abgekürzter Name des Monats.
`%B`	Lokalisierter Monatsname
`%c`	Lokalisierte entsprechende Datums- und Zeitdarstellung.
`%d`	Tag des Monats als Dezimalzahl `01` – `31`.
`%H`	Stunde als Dezimalzahl `00` – `23`.
`%I`	Stunde als Dezimalzahl `01` – `12`.
`%j`	Tag des Jahres als Dezimalzahl. `001` – `366`.
`%m`	Monat als Dezimalzahl `01` – `12`.
`%M`	Minute als Dezimalzahl `00` – `59`.
`%p`	Lokalisiertes Äquivalent von AM oder PM.[1]
`%S`	Sekunde als Dezimalzahl `00` – `61`.[2]
`%U`	Wochenzahl eines Jahres als Dezimalzahl `00` – `53` (Sonntag als erster Tag der Woche). Alle Tage eines Jahres vor dem ersten Sonntag werden der Woche 0 zugerechnet.[3]
`%w`	Wochentag als Dezimalzahl `0` (Sonntag) – `6`.
`%W`	Wochenzahl eines Jahres als Dezimalzahl `00` – `53` (Montag als erster Tag der Woche) Alle Tage eines Jahres vor dem ersten Montag werden der Woche 0 zugerechnet.[3]
`%x`	Lokalisierte angemessene Datumsdarstellung.
`%X`	Lokalisierte angemessene Zeitdarstellung.
`%y`	Jahr ohne Jahrhundert als Dezimalzahl `00` – `99`.
`%Y`	Jahr mit Jahrhundert als Dezimalzahl.
`%Z`	Name der Zeitzone (keine Zeichen, wenn keine Zeitzone existiert.
`%%`	Das Zeichen `%`.

Übersetzungen für Pakete

1. Fügen sie in ihr Paket folgende Unterverzeichnisse für jede gewünschte Sprache ein: `locales/<locale>/LC_MESSAGES/`, z.B.:

```
locales/de/LC_MESSAGES/
locales/en/LC_MESSAGES/
```

2. Erstellen Sie die entsprechenden `vs.theme.po`-Dateien in den `LC_MESSAGES`-Ordnern und editieren diese z.B. mit poEdit.

3. Registrieren sie die Übersetzungen in der `configure.zcml`-Datei mit folgenden Anweisungen:

```
<configure
    ...
    xmlns:i18n="http://namespaces.zope.org/i18n"
    i18n_domain="vs.theme">
    ...
    <i18n:registerTranslations directory="locales" />
</configure>
```

4. Normalerweise werden die `.po`-Dateien beim Starten des Zope-Servers vom PlacelessTranslationService kompiliert, d.h. `.mo`-Dateien erzeugt.

5. Gegebenenfalls können auch mit folgendem Skript die `.po`-Dateien kompiliert werden:

```
# Compile po files
for lang in $(find locales -mindepth 1 -maxdepth 1 -type d); do
    if test -d $lang/LC_MESSAGES; then
        msgfmt -o $lang/LC_MESSAGES/${PRODUCT_NAME}.mo $lang/LC_MESSAGES/$
↪{PRODUCT_NAME}.po
    fi
done
```

Plone 4

In Plone 4 nutzt der `PlacelessTranslationService` `zope.i18n`. Die Ordner `i18n` und `locales` verwenden dieselbe Katalog-Engine.

In Plone 3 kompilierte der `PlacelessTranslationService` die `*.po`-Dateien erneut. In Plone 4 jedoch nutzt er das `zope.i18n`-Modul, für das explizit angegeben werden muss, dass es `*.po`-Dateien kompilieren soll. Hierzu wird in der Buildout-Konfiguration dann folgendes angegeben:

```
[instance]
...
environment-vars =
    zope_i18n_compile_mo_files = true
```

[1] In der `strftime` -Funktion ändert die `%p` -Anweisung nur die Ausgabe, wenn `%I` verwendet wird.
[2] Der Umfang ist tatsächlich `00` – `61` um Schaltsekunden («leap seconds» und «double leap seconds») berücksichtigen zu können.
[3] In `strftime` -Funktionen wird `%U` und `%W` nur berechnet, wenn Tag, Woche und Jahr angegeben sind.

Einschränken der verwendeten Sprachen

Um den Startprozess von der Zope-Instanzen zu beschleunigen und weniger Speicher zu verbrauchen, kann eine Umgebungsvariable gesetzt werden, die das Kompilieren und Laden von `*.po`-Dateien einschränkt. Um die Sprachen z.B. auf Englisch und Deutsch zu beschränken, kann in der `buildout.cfg`-Datei folgendes angegeben werden:

```
[instance]
...
environment-vars =
    ...
    PTS_LANGUAGES en de
    zope_i18n_allowed_languages en de
```

`PTS_LANGUAGES` In Plone 3 beeinflusst diese Angabe sowohl das Kompilieren der Übersetzungsdateien in `i18n`- und `locales`-Ordnern.

In Plone 4 wird hierdurch nur noch das Kompilieren der Übersetzungsdateien in `i18n`-Ordnern.

`zope_i18n_allowed_languages` In Plone 4 wird hierdurch das Kompilieren der Übersetzungsdateien in `locales`-Ordnern gesteuert.

Siehe auch:

- Internationalizing a Package.

Übersetzungen in der Plone-Domäne

Die folgenden Konfigurationsdateien des *GenericSetup Tools* müssen in der Plone-Domäne übersetzt werden:

- `portal_atct.xml`
- `portlets.xml`
- `workflows/MYWORKFLOW/definition.xml`

Erweitern der Übersetzungen

In **Plone 3** können Übersetzungen der Übersetzungsdomäne `plone` nicht im `locales`-Ordner angepasst werden. Sollten Sie dies dennoch gemacht haben, werden Sie feststellen, dass nur noch Ihre eigenen Übersetzungen angezeigt werden. Damit Ihre Übersetzungen zusätzlich verwendet werden, müssen die Übersetzungsdateien im `i18n`-Ordner erstellt werden.

Schemata

Zur Lokalisierung von Schemata wird die `message_id` generiert. Dabei wird der ID des Schemata `label_schema_` vorangestellt. Im Folgenden ein Beispiel aus `plone/app/locales/locales/de/LC_MESSAGES/plone.po`:

```
#. Default: "Categorization"
#: ./ATContentTypes/content/schemata.py
msgid "label_schema_categorization"
msgstr "Kategorisierung"
```

Überschreiben von Plone-Übersetzungen

Hierzu erstellen wir zunächst in unserem Buildout-Projekt einen Ordner namens `i18n` und darin `plone-vs-de.po`. Anschließend fügen wir in der `buildout.cfg`-Datei einen neuen Abschnitt hinzu:

```
[buildout]
...
instance
i18n-overwrites
...

[i18n-overwrites]
recipe = plone.recipe.command
command =
    ln -sf ${buildout:directory}/i18n ${instance:location}/
update-command =
    ${i18n-overwrites:command}
```

Plone 4

In **Plone 4** hingegen wird das `i18n`-Verzeichnis in der Instanz ignoriert. Um Übersetzungen der Domäne Plone zu überschreiben, müssen Sie ein eigenes `locales`-Verzeichnis anlegen und die Plone-Übersetzungsdatei `plone.app.locales-4.0.2-py2.6.egg/plone/app/locales/locales/de/LC_MESSAGES/plone.po` dahin kopieren, also z.B. nach `vs.theme/vs/theme/locales/de/LC_MESSAGES/plone.po`. Anschließend können Sie aus dieser Datei die Übersetzungen löschen, die Sie nicht ändern wollen und Ihre Übersetzungen hinzufügen.

Nun müssen wir in unserer Buildout-Konfiguration nur noch beachten, dass die `zcml`-Datei von `vs.theme` vor allen anderen Paketen geladen wird. Dies kann gewährleistet werden indem `vs.theme` als erstes in der Liste der `zcml`-Optionen gelistet wird, z.B.:

```
[instance]
...
eggs =
    Zope2
    Plone
    ${buildout:eggs}

zcml =
    vs.theme
    ...
```

Inhalte übersetzen

LinguaPlone

LinguaPlone ist ein Produkt, das mehrsprachige **Inhalte** in einer Plone-Site ermöglicht. Um LinguaPlone verwenden zu können, müssen zunächst mehrere Sprachen im Plone Language Tool ausgewählt worden sein, für das ein Profil mit der Datei `portal_languages.xml` angelegt werden kann. Diese sieht z.B. so aus:

```
<?xml version="1.0"?>
<object name="portal_languages">
 <default_language value="de"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 <use_cookie_negotiation value="True"/>
 <use_content_negotiation value="True"/>
 <use_request_negotiation value="True"/>
 <display_flags value="False"/>
 <start_neutral value="False"/>
 <supported_langs>
   <element value="en"/>
   <element value="de"/>
 </supported_langs>
</object>
```

Anschließend sollte der View `/@@language-setup-folders` aufgerufen werden, damit auch die Startseite für beide Sprachen zur Verfügung gestellt werden kann. Dieser View liefert dann folgende Ausgabe:

```
Setup of language root folders on Plone site 'mysite'
Added 'en' folder: en
INavigationRoot setup on folder 'en'
Added 'de' folder: de
INavigationRoot setup on folder 'de'
Translations linked.
Portal default page removed.
Moved default page 'front-page' to folder 'de'.
Root language switcher set up.
```

Der View strukturiert die Site in folgenden Schritten um:

1. Zunchst wird für jede der angegebenen Sprachen ein Ordner erstellt.
2. Jeder dieser Ordner wird an das `INavigationRoot`-Interface gebunden.
3. Nun werden die entsprechenden Verlinkungen zwischen diesen Ordnern für die Übersetzung erstellt.
4. Anschließend wird die Standardseite in den Ordner verschoben, dessen Sprache sie zugeordnet ist.
5. Schließlich wird der *root language switcher* aufgesetzt um vom `ISiteRoot`-Objekt auf den Ordner der Standardsprache umzuleiten.

Programmatisch kann dies in der `setuphandlers.py`-Datei geschehen:

```
import transaction
...
class Generator:

    def configureLinguaPlone(self, portal):

        pl = getToolByName(portal, 'portal_languages')
        pl.supported_langs = ('de', 'en')

        transaction.savepoint(1)
        portal.restrictedTraverse('@@language-setup-folders')()

def setupVarious(context):
    ...
    gen.configureLinguaPlone(site)
```

TinyMCE für sprachneutrale Inhalte

Werden sprachneutrale Inhalte in mehreren Sprachen benötigt, z.B. für Bilder, so kann es sich empfehlen, einen Ordner mit diesen Inhalten auf derselben Ebene wie die Sprachenordner anzulegen.

Damit nun die Bilder in diesem Ordner auch aus dem Popup-Fenster von TinyMCE eingebunden werden können, erstellen wir einen MonkeyPatch mit collective.monkeypatcher. Zunächst wird folgendes in die `configure.zcml`-Datei eingetragen:

```
<configure
    ...
    xmlns:monkey="http://namespaces.plone.org/monkey">
...
    <monkey:patch
        description="TinyMCE JSON Folder listing should ignore INavigationRoot"
        class="Products.TinyMCE.adapters.JSONFolderListing.JSONFolderListing"
        original="getListing"
        replacement=".patches.getListing"
        />
```

Anschließend erstellen wir die `patches.py`-Datei mit der Methode `getListing`. Diese ist übernommen aus `Products.TinyMCE.adapters.JSONFolderListing.py` mit folgender Änderung:

```
68c68,69
< if INavigationRoot.providedBy(object) or (rooted == "True" and document_base_url[:-
↪1] == object.absolute_url()):
---
> #if INavigationRoot.providedBy(object) or (rooted == "True" and document_base_url[:-
↪1] == object.absolute_url()):
> if (rooted == "True" and document_base_url[:-1] == object.absolute_url()):
```

ReferenceBrowserWidget für sprachneutrale Inhalte

Damit auch sprachneutrale Referenzen eingebunden werden können, muss auch für das ReferenceBrowserWidget ein MonkeyPatch bereitgestellt werden. Zunächst wird dieser registriert in der `configure.zcml`:

```
<monkey:patch
    description="Navigation support for the ReferenceBrowserWidget across␣
↪INavigationRoot"
    class="archetypes.referencebrowserwidget.browser.view.ReferenceBrowserPopup"
    original="breadcrumbs"
    replacement=".patches.breadcrumbs"
    />
```

Nun wird in `patches.py` die Methode ``breadcrumbs`` kopiert aus `archetypes.referencebrowserwidget.browser.view.ReferenceBrowserPopup` und folgendermaßen abgeändert:

```
12c14,21
<                       portal_state.navigation_root_url())}]
---
>                       portal_state.portal_url())}]
>
> if portal_state.portal_url() != portal_state.navigation_root_url():
>     nav_root_path = portal_state.navigation_root_path()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
>     nav_root = self.context.restrictedTraverse(nav_root_path)
>     newcrumbs.append({'Title': nav_root.Title(),
>                       'absolute_url': self.genRefBrowserUrl(
>                             portal_state.navigation_root_url())})
```

Mehrsprachige Inhaltstypen erstellen

Wollen Sie LinguaPlone mit ihren eigenen Inhaltstypen verwenden, werden die Klassen und Methoden nicht direkt von Archetypes importiert, sondern es wird zunächst versucht, sie von LinguaPlone zu übernehmen:

```
try:
    from Products.LinguaPlone.public import *
except ImportError:
    # No multilingual support
    from Products.Archetypes.public import *
```

Sprachunabhängige Felder

Sprachunabhängige Felder, z.B. für Namen und Datum, werden vom Originalartikel (*canonical item*) übernommen. Die Werte werden jedoch in jedem übersetzten Artikel gespeichert, sodass jeder Artikel jedes Attribut enthält und damit aus dem Kontext verschoben oder direkt referenziert werden können.

Die Sprachunabhängigkeit wird für ein Feld in der AT-Schemadefinition angegeben mit `languageIndependent=1`.

Sprachauswahl

Beim ersten Aufruf einer LinguaPlone-Site wird der Header `HTTP_ACCEPT_LANGUAGE`, der vom Browser gesendet wird, verwendet, um zu entscheiden, welche Sprache verwendet wird. Anschließend wird ein Cookie mit dieser Entscheidung zurückgesendet. Diese Sprache wird dann solange verwendet, bis vom Nutzer explizit eine andere Sprache in der Plone-Site ausgewählt wird – dann wird auch der Cookie aktualisiert. Steht an einer anderen Stelle der Site ein Artikel nicht in der gewünschten Sprache zur Verfügung, wird eine Seite mit den verfügbaren Sprachen angezeigt.

Sprachspezifische Suche

LinguaPlone filtert in der Suche für alle Artikeltypen, die verschiedene Sprachen unterscheiden, diejenigen heraus, die nicht der Sprachauswahl entsprechen.

Soll in allen verfügbaren Sprachen gesucht werden, kann in der Suche `Language=all` angegeben werden.

Weitere Module

Mit slc.linguatools und raptus.multilanguagefields stehen noch zwei weitere Werkzeuge für mehrsprachige Inhalte in Plone zur Verfügung.

3.10 Relationale Datenbanken

In diesem Kapitel erfahren Sie, wie auch die Inhalte relationaler Datenbanken in Ihre Plone-Site integriert werden können.

3.10.1 Datenmodellierung

Für Registration als auch Registrant werden die Felder `registration_key`, respektive `occurrence_key` angelegt, die auch im Katalog indiziert werden sollen. Mit ihnen werden die hierarchischen ZODB-Inhalte auf ein relationales Datenbankmodell abgebildet.

Anschließend erstellen wir entsprechende Interfaces in `vs.registration.interfaces`:

```
class IRegistration(Interface):
    """A folder containing registrants
    """
    contains('vs.registration.interfaces.IRegistrant',)

    registration_key = schema.ASCIILine(title=_(u"Registrant key"),
                                        description=_(u"This should match the␣
↪registration key used by the booking system"),
                                        required=True)

    ...

class IRegistrant(Interface):
    """A registrant
    """

    occurrence_key = schema.ASCIILine(title=_(u"Occurrence key"),
                                      description=_(u"This should match the␣
↪occurrence key used by the booking system"),
                                      required=True)

    ...
```

Obwohl sie nicht in der ZODB gespeichert werden, werden *Occurrence*- und *Reservation*-Datensätze in Python-Code verarbeitet. Daher werden für beide zunächst ebenfalls Interfaces definiert:

```
class IOccurrence(Interface):
    """A single occurrence
    """

    occurrence_key = schema.Int(title=_(u"Occurrence identifier"),
                                description=_(u"A unique id for this occurrence"),
                                required=True,
                                readonly=True)

    registration = schema.Object(title=_(u"Registration"),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                                schema=IRegistration
                                required=True,
                                readonly=True)

    occurrence_time = schema.Date(title=_(u"Date/time"),
                                  required=True,
                                  readonly=True)

    vacancies = schema.Int(title=_(u"Vacancies"),
                           description=_(u"Vacancies for this Occurrence"))

class IReservation(Interface):
    """A reservation for a particular occurrence
    """

    customer_name = schema.TextLine(title=_(u"Customer name"),
                                    description=_(u"The name of the customer making␣
↪the reservation"),
                                    required=True)

    num_reservations = schema.Int(title=_(u"Number of reservations"),
                                  description=_(u""),
                                  required=True,
                                  min=1)


    occurrence = schema.Object(title=_(u"Occurrence"),
                               description=_(u"Occurrence to book for"),
                               schema=IOccurrence,
                               required=True)
```

Diese Interfaces sind implementiert als einfache Domainklassen, die wir später auf die Datenbanktabellen abbilden.

- `occurrence.py`:

```
class Occurrence(object):
    """A single occurrence.
    """

    implements(IOccurrence)

    occurrence_key = None
    registration = None
    occurrence_time = None
    vacancies = 0
```

- `reservation.py`:

```
class Reservation(object):
    """A reservation for a particular occurrence
    """

    implements(IReservation)

    customer_name = u""
    num_reservations = 0
    occurrence = None
```

In diesem Fall wird nicht wie in *Archetypes-Artikeltypen* letztlich abgeleitet von `persistence.Persistent`, und somit sind die Daten auch nicht in der ZODB verfügbar.

3.10.2 Hilfsmethoden

Im weiteren Verlauf sollen Listen der *Occurrences* in die Ansichten des *Registration*-Artikeltyps eingeblendet werden. Angemeldete Nutzer sollen so in der Lage sein, für eine bestimmte *Occurrence* Reservierungen vorzunehmen.

Um jedoch die Views nicht direkt mit der Datenbank kommunizieren zu lassen, werden Hilfsmethoden eingeführt, wobei die Datenbankoperationen abstrahiert werden. Hiermit werden dann auch Tests möglich, die nicht auf eine produktive Datenbank zurückgreifen müssen.

Zunächst werden die Interfaces für die Hilfsmethoden in `vs.registration.interfaces` definiert:

```
class IRegistrations(Interface):
    """Searches appropriate occurrences
    """

    def registrations_for_registrant(registrant):
        """Return a list of all registrations of a specific registrant as
        list of dictionaries with keys 'registration_key', 'url' and
        'title'.
        """

    def occurrence_by_key(occurence_key):
        """Get an IOccurrence from an occurrence key
        """

class IReservations(Interface):
    """A utility capable of making reservations
    """

    def __call__(reservation):
        """Make a reservation
        """
```

Im weiteren Verlauf werden diese Hilfsmethoden dann implementiert und Views erstellt, die diese nutzen.

3.10.3 Datenbank erstellen

In diesem Beispiel verwenden wir MySQL 5.0 mit InnoDB-Tabellen, die grundlegenden Konzepte lassen sich jedoch leicht auch auf andere relationale Datenbanken übertragen.

Hier das Skript zur Definition der Tabelle `registration` und einiger Beispieldaten:

```
create database if not exists registration;
use registration;

-- Occurrences
create table if not exists occurrence (
    occurrence_key integer unsigned not null auto_increment primary key,
    registration_key char(4) not null,
    registrant_key char(4) not null,
    occurrence_time datetime not null,
    vacancies integer unsigned not null,
    index showing_registration_key(registration_key),
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    index showing_registrant_key(registrant_key),
    index showing_occurrence_time(occurrence_time),
    index showing_vacancies(vacancies)
) engine=InnoDB;

-- Reservations
create table if not exists reservation (
    reservation_key integer unsigned not null auto_increment primary key,
    occurrence_key integer unsigned not null,
    num_reservations tinyint unsigned not null,
    customer_name varchar(64) not null,
    index reservation_num_reservations(num_reservations),
    foreign key(occurrence_key)
        references occurrence(occurrence_key)
            on update restrict
            on delete restrict
) engine=InnoDB;
```

Für unser Beispiel wird der Zugang zu MySQL durch `root` ohne Passwort ermöglicht. Für Produktivsysteme sollte jedoch selbstverständlich ein Passwort vergeben werden.

3.10.4 SQLAlchemy

SQLAlchemy ist eine Python-Bibliothek zur Integration relationaler Datenbanken. SQLAlchemy unterstützt eine Vielzahl relationaler Datenbanken, bietet eine niedrigschwellige Verwaltung der Verbindungen, eine Python-API zur Erstellung von SQL-Anfragen und *Object/Relational Mapping* (ORM)-Funktionalität.

In unserem Beispiel werden wir SQLAlchemy verwenden um unsere beiden Datenbanken zu implementieren, um Objekte auf Relationen abzubilden und um SQL-Anfragen zu erstellen. Darüberhinaus werden wir mit `collective.lead` Datenbankverbindungen verwalten und SQL-Transaktionen in Zope-Transaktionen zu überführen.

Wesentliche Komponenten von SQLAlchemy sind:

`Engine` verwaltet die Datenbankverbindungen.

In unserem Fall wird `Engine` verwaltet von `collective.lead.interfaces.IDatabase`.

`Table` repräsentiert eine Datenbanktabelle.

`Metadata` bindet Tabellen an eine spezifische `Engine`.

`Mapper` repräsentiert ein Eintrag in einer Datenbank als Python-Klasse.

`Session` verwaltet Instanzen von `Mapper`-Klassen. Eine Session kann neue Instanzen von einer Datenbank laden, Änderungen an Objekten sichern und neue Objekte als *Records* in der Datenbank speichern.

`Connection` erlaubt die Ausführung von SQL-Anfragen, entweder als Python-Anweisung oder als String.

3.10.5 Datenbankanbindungen

SQLAlchemy bietet standardisierte Interaktionsmuster zur Erstellung von Engines, Metadata, Tabellen und Mapper. Dabei ist zu beachten, dass

- verschiedene Produkte ihre eigenen Datenbankverbindungen aufbauen;
- jede geteilte Datenbankquelle *thread-safe* ist;
- Datenbanktransaktionen mit Zope-Transaktionen synchronisiert werden;
- *Data Source names* (DSN) zur Laufzeit nicht bekannt sind.

collective.lead bietet eine Basisklasse zur Erstellung von Hilfsmethoden, die Verbindungseinstellungen, Tabellen und Mapper kapseln. Damit wir dies in unseren Hilfsmethoden `vs.registrations` und `vs.reservations` verwenden können, wird `collective.lead` als Abhängigkeit in `vs.registration/setup.py` eingetragen:

```
install_requires=[
    'setuptools',
    # -*- Extra requirements: -*-
    'MySQL-python',
    'collective.lead>=1.0b3,<2.0dev',
],
```

Bei einem Aufruf von `./bin/buildout` sollte nun `collective.lead` mitinstalliert werden, welches dann die letzte unterstützte Version von SQLAlchemy installiert. Daneben benötigen wir noch das MySQL-python-Paket, welches MySQL-Treiber für Python bereitstellt.

Das Datenbank-Hilfsprogramm selbst wird dann in `vs.registration/vs/registration/db.py` erstellt. Die Datei enthält ebenfalls die Implementierung von `IDatabaseSettings`, eine persistente, lokale Hilfsmethode zum Speichern der Verbindungseinstellungen:

```
from persistent import Persistent

from zope.interface import implements
from zope.component import getUtility

from collective.lead import Database
from vs.registration.interfaces import IDatabaseSettings

from sqlalchemy.engine.url import URL
from sqlalchemy import Table, mapper, relation

from vs.registration.occurrence import Occurrence
from vs.registration.reservation import Reservation

class ReservationsDatabaseSettings(Persistent):
    implements(IDatabaseSettings)

    drivername = 'mysql'
    hostname = 'localhost'
    port = None
    username = ''
    password = None
    database = ''

class ReservationsDatabase(Database):
    @property
    def _url(self):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        settings = getUtility(IDatabaseSettings)
        return URL(drivername=settings.drivername, username=settings.username,
                   password=settings.password, host=settings.hostname,
                   port=settings.port, database=settings.database)

    def _setup_tables(self, metadata, tables):
        tables['occurrence'] = Table('occurrence', metadata, autoload=True)
        tables['reservation'] = Table('reservation', metadata, autoload=True)

    def _setup_mappers(self, tables, mappers):
        mappers['occurrence'] = mapper(Occurrence, tables['occurrence'])
        mappers['reservation'] = mapper(Reservation, tables['reservation'],
                                        properties = {
                                            'occurrence' : relation(Occurrence),
                                            })
```

Die `collective.lead.Database`-Klasse erlaubt uns, nur wenige Eigenschaften anzugeben, um eine Datenbankverbindung, Tabellen und Mapper zu erstellen.

Nun wird die `ReservationsDatabase`-Methode noch in `vs.registration/vs/registration/configure.zcml` registriert:

```
<utility
    provides="collective.lead.interfaces.IDatabase"
    factory=".db.ReservationsDatabase"
    name="vs.reservations"
    />
```

Da die lokale `ReservationsDatabaseSettings`-Hilfsmethode jedoch erst mit der Installation des Produkts registriert werden muss, kann das Generic Setup-Profil `vs.registration/vs/registration/profiles/default/componentregistry.xml` hierfür verwendet werden:

```
<?xml version="1.0"?>
<componentregistry>
    <utilities>
        <utility
            interface="vs.registration.interfaces.IDatabaseSettings"
            factory="vs.registration.db.ReservationsDatabaseSettings"
            />
    </utilities>
</componentregistry>
```

Nun sollte auf die Datenbank zugegriffen werden können mit:

```
>>> from zope.component import getUtility
>>> from collective.lead.interfaces import IDatabase
>>> db = getUtility(IDatabase, name='vs.reservations')
```

Dem `db`-Objekt stehen anschließend die Eigenschaften von `collective.lead.interfaces.IDatabase` zur Verfügung.

3.10.6 Konfigurationsformular

Um die Datenbankeinstellungen konfigurierbar zu machen, erstellen wir eine spezielle Website-Konfigurationsseite. Wir werden hierzu `zope.formlib` und folgendes Interface verwenden:

```
class IDatabaseSettings(Interface):
    """Database connection settings.
    """

    drivername = schema.ASCIILine(title=_(u"Driver name"),
                                  description=_(u"The database driver name"),
                                  default='mysql',
                                  required=True)

    hostname = schema.ASCIILine(title=_(u"Host name"),
                                description=_(u"The database host name"),
                                default='localhost',
                                required=True)

    port = schema.Int(title=_(u"Port number"),
                      description=_(u"The database port number. Leave blank to use␣
↪the default."),
                      required=False)

    username = schema.ASCIILine(title=_(u"User name"),
                                description=_(u"The database user name"),
                                required=True)

    password = schema.Password(title=_(u"Password"),
                               description=_(u"The database password"),
                               required=False)

    database = schema.ASCIILine(title=_(u"Database name"),
                                description=_(u"The name of the database on this␣
↪server"),
                                required=True)
```

Für das Formular selbst nutzen wir dann die Infrastruktur von `plone.app.controlpanel` in `vs.registration/vs/registration/browser/dbsettings.py` verwendet:

```
from zope.component import getUtility
from zope.formlib import form

from plone.app.controlpanel.form import ControlPanelForm

from collective.lead.interfaces import IDatabase

from vs.registration.interfaces import IDatabaseSettings
from vs.registration import RegistrationMessageFactory as _

def reservations_database_settings(context):
    return getUtility(IDatabaseSettings)

class ReservationsDatabaseControlPanel(ControlPanelForm):
    form_fields = form.FormFields(IDatabaseSettings)

    form_name = _(u"Reservations Database settings")
    label = _(u"Reservations Database settings")
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    description = _(u"Please enter the appropriate connection settings for the␣
↪database")

    def _on_save(self, data):
        db = getUtility(IDatabase, name='vs.reservations')
        db.invalidate()
```

reservations_database_settings Der Adapter bindet die Werte aus dem Konfigurationsformular an `context`, also an Plone Site Root.

ReservationsDatabaseControlPanel Die Klasse definiert den View für das Formular, das `zope.formlib` aus dem `IDatabaseSettings`-Interface generiert.

_on_save Funktion, die von der Basisklasse aufgerufen wird sobald auf *Speichern* geklickt wird. Dabei leert die `invalidate`-Methode alle internen Caches, sodass die neuen Einstellungen wirksam werden.

Anschließend werden in `vs.registration/vs/registration/browser/configure.zcml` Icon, View und Adapter registriert:

```
<browser:resource
    name="dbsettings_icon.gif"
    image="dbsettings_icon.gif"
    />

<browser:page
    name="reservations-database-controlpanel"
    for="Products.CMFPlone.interfaces.IPloneSiteRoot"
    class=".dbsettings.ReservationsDatabaseControlPanel"
    permission="cmf.ManagePortal"
    />

<adapter
    for="Products.CMFPlone.interfaces.IPloneSiteRoot"
    provides="vs.registration.interfaces.IDatabaseSettings"
    factory=".dbsettings.reservations_database_settings"
    />
```

Damit das Konfigurationsformular auf der Plone-Konfigurationsseite angezeigt wird, wird die Datei `vs.registration/vs/registration/profiles/default/controlpanel.xml` mit folgendem Inhalt erstellt:

```
<?xml version="1.0"?>
<object name="portal_controlpanel" meta_type="Plone Control Panel Tool">
    <configlet title="Reservations Database" action_id="ReservationsDatabase"
               appId="ReservationsDatabase" category="Products" condition_expr=""
               url_expr="string:${portal_url}/@@reservations-database-controlpanel"
               visible="True">
        <permission>Manage portal</permission>
    </configlet>
</object>
```

Nun wird noch das Icon in `vs.registration/vs/registration/profiles/default/actionicons.xml` registriert:

```
<?xml version="1.0"?>
<action-icons>
    <action-icon category="controlpanel"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                action_id="ReservationsDatabase"
                title="Reservations Database"
                priority="0" icon_expr="++resource++dbsettings_icon.gif"/>
</action-icons>
```

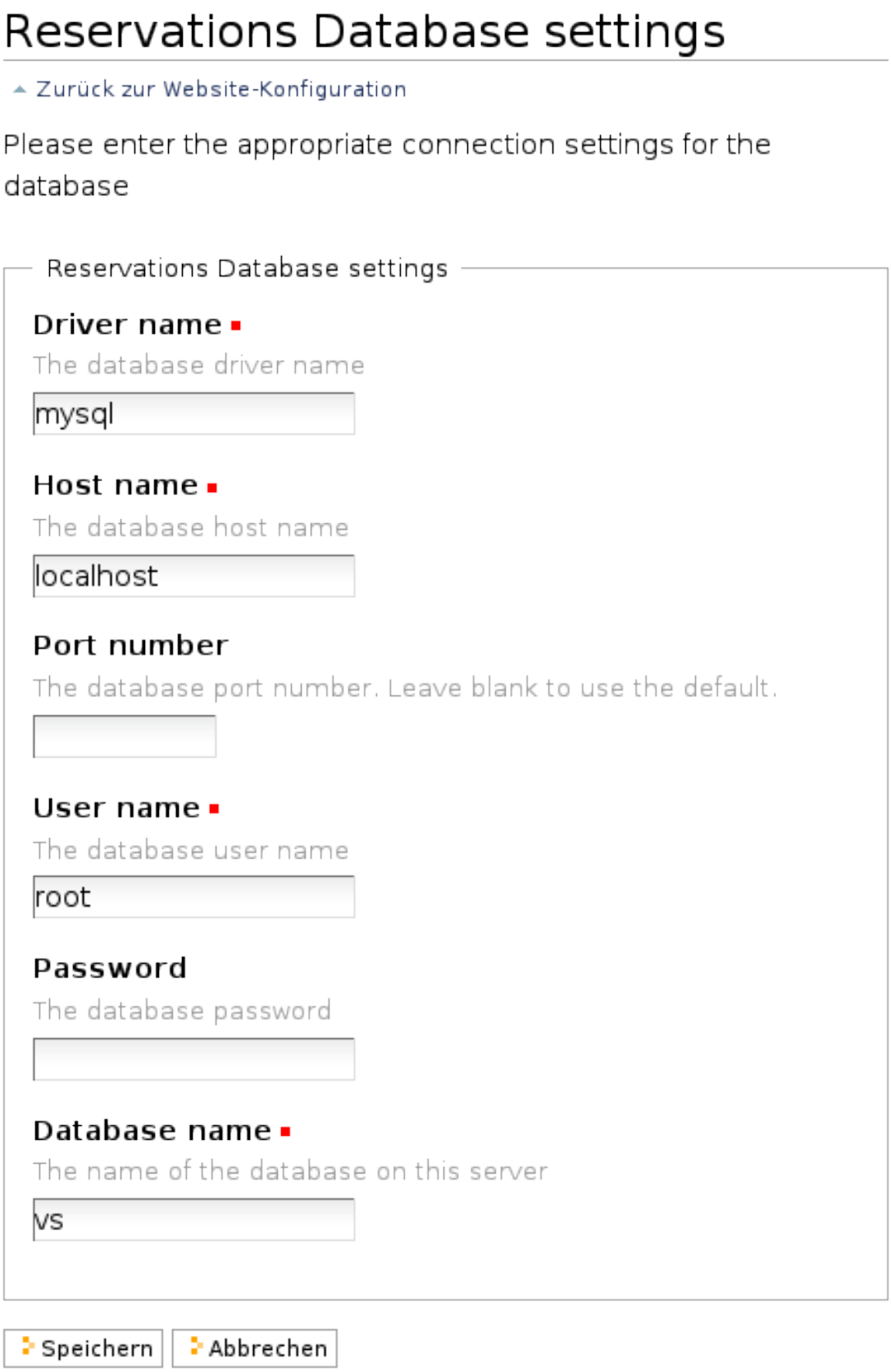
Beachten Sie bitte, dass das `action_id`-Attribut in beiden Profilen gleich ist. Und so sieht das Plone-Konfigurationsformular aus:

3.10.7 Datenbankabfragen

Die Implementierung des `IRegistrations`-Interfaces aus Hilfsmethoden erfolgt nun in `occurrence.py`:

```
from zope.interface import implements
from zope.component import getUtility
from zope.app.component.hooks import getSite

from Products.CMFCore.interfaces import ISiteRoot
from Products.CMFCore.utils import getToolByName

from vs.registration.interfaces import IRegistrant
from vs.registration.interfaces import IRegistration
from vs.registration.interfaces import IOccurrence
from vs.registration.interfaces import IRegistrations

import sqlalchemy as sql
from collective.lead.interfaces import IDatabase

...

class Registrations(object):
    implements(IRegistrations)

    def registrations_for_registrant(self, registrant):
        db = getUtility(IDatabase, name='vs.reservations')
        connection = db.connection

        statement = sql.select([Occurrence.c.registration_key],
                               sql.and_(
                                    Occurrence.c.registrant_key == registrant.
↪registrant_key,
                               ),
                               distinct=True)

        results = connection.execute(statement).fetchall()

        registration_keys = [row['registration_key'] for row in results]

        site = getSite()
        catalog = getToolByName(site, 'portal_catalog')

        return [ dict(registration_key=registration.registration_key,
                      url=registration.getURL(),
                      title=registration.Title,)
                 for registration in
                    catalog(object_provides=IRegistration.__keyentifier__,
```

(Fortsetzung auf der nächsten Seite)


Reservations Database settings
Zurück zur Website-Konfiguration
Please enter the appropriate connection settings for the database
Reservations Database settings
Driver name
The database driver name
mysql
Host name
The database host name
localhost
Port number
The database port number. Leave blank to use the default.
User name
The database user name
root
Password
The database password
Database name
The name of the database on this server
vs
Speichern
Abbrechen

(Fortsetzung der vorherigen Seite)

```
                                registration_key=registration_keys,
                                sort_on='sortable_title')
                ]
```

In der `Registrations`-Klasse werden *Occurrences* von *Registrants* in *Registrations* gefunden. Dabei wird zunächst die Datenbankverbindung hergestellt. Anschließend werden verschiedene SQLAlchemy-Konstrukte verwendet um eine Datenbankabfrage an der `occurrence`-Datenbank vorzunehmen. So bedeutet z.B. die Syntax `Occurrence.c.registrant_key`, dass ein Mapping zwischen der `registrant_key`-Spalte (`c` olumn) der `occurrence`-Tabelle und der `Occurrence`-Klasse stattfindet. Die Syntax ist umfangreich beschrieben in der SQLAlchemy-Dokumentation. Schließlich wird eine Plone-Katalogabfrage nach `registrant`-Objekten mit den in der `occurrence`-Tabelle gefundenen `registration_keys` erstellt. Diese werden dann in eine Liste von Wörterbuchern, wie sie in `IRegistrations` definiert sind, gepackt.

Die `occurrences`-Methode nutzt hingegen SQLAlchemys ORM-API um die *Occurrences* eines gegebenen *Registrant* als Liste von `Occurrence`-Objekten auszugeben:

```
def occurrences(self, registrant, registration):

    db = getUtility(IDatabase, name='vs.reservations')
    session = db.session

    occurrences = session.query(Occurrence).select(sql.and_(
                                                    Occurrence.c.registrant_
↪key==registrant.registrant_key,
                                                    Occurrence.c.registration_
↪key==registration.registration_key,
                                                    ),
                                                )

    for occurrence in occurrences:
        occurrence.registrant = registrant
        occurrence.registration = registration

    return occurrences
```

Da kein Mapping zwischen den `Registrant`- und `Registration`-Klassen und der Datenbank stattfindet, können sie von SQLAlchemy nicht geladen und zurückgegeben werden. Stattdessen werden die `registrant`- und `registration`-Attribute direkt beim Laden des Objekts gesetzt.

3.10.8 Daten schreiben

Im Gegensatz zur `Registrations`- werden mit der `Reservations`-Hilfsmethode nicht nur Abfragen an die Datenbank gestellt sondern auch Datensätze geschrieben. Die `reservations.py`-Datei sieht so aus:

```
from zope.interface import implements
from zope.component import getUtility

from vs.registration.interfaces import IReservations
from vs.registration.interfaces import ReservationError

from vs.registration.occurrence import Occurrence
from vs.registration import RegistrationMessageFactory as _

import sqlalchemy as sql
from collective.lead.interfaces import IDatabase
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
class Reservations(object):
    implements(IReservations)

    def __call__(self, reservation):
        db = getUtility(IDatabase, name='vs.reservations')
        session = db.session

        occurrence = reservation.occurrence
        session.refresh(occurrence)

        if occurrence.vacancies <= 0:
            raise ReservationError(_(u"There are not enough vacancies anymore!"))
        elif occurrence.vacancies < reservation.num_reservations:
            raise ReservationError(_(u"Not enough reservations remaining!"))

        occurrence.vacancies -= reservation.num_reservations
        session.update(occurrence)
        session.save(reservation)
        session.flush()
```

Die Klasse `Reservations` macht Reservierungen wobei zunächst überprüft wird, ob nach Plätze frei sind. Anschließend wird``occurrence`` aktualisiert (`refresh`) um zu vermeiden, dass eine andere Transaktion sich die freien Plätze genommen hat. Dann wird die Zahl der verbleibenden freien Plätze aktualisiert (`update`) und fügen die neue Reservierung hinzu (`save`). Sofort danach wird die Session beendet (`flush`) um zu gewährleisten, dass die Änderungen gespeichert werden.

Sind für die angefragte *Occurrence* keine Plätze mehr frei, wird die Fehlermeldung `ReservationError` ausgegeben. Diese ist definiert in `interfaces.py`:

```
class ReservationError(Exception):

    def __init__(self, message):
        Exception.__init__(self, message)
        self.error_message = message
```

3.10.9 Views

Folgende Änderungen können nun in `browser/registrant.py` hinzugefügt werden:

```
from zope.component import getUtility
...
@memoize
def registrations(self):
    context = aq_inner(self.context)
    registrations = getUtility(IRegistrations)
    return registrations.registrations_for_registrant(context)
```

und in `browser/registrant.pt`:

```
<h2 i18n:translate="title_registrated_at">Registrated at</h2>
<ul>
    <tal:block repeat="registration view/registrations">
        <li>
            <a tal:attributes="href registration/url"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                tal:content="registration/title" />
        </li>
    </tal:block>
</ul>
```

3.11 Produktivserver

3.11.1 Buildout für Produktivserver

Wir haben eine Entwicklungsumgebung aufgesetzt, die einige Entwicklungswerkzeuge enthält und für die an mehreren Stellen Debugging ermöglicht wurde. Mit Buildout ist es nun einfach möglich, dieses Projekt in eine Produktivumgebung zu überführen.

Zope Enterprise Objects

Zum Entwickeln haben wir eine einfache Zope-Instanz aufgesetzt. Für den Produktivserver wollen wir Zope Enterprise Objects (ZEO) verwenden, da so mehrere Zope-Instanzen auf eine ZODB, die vom ZEO-Server verwaltet wird, zugreifen können. Dies bietet mehrere Vorteile:

- Der ZEO-Server und die ZEO-Clients sollten auf verschiedenen Maschinen sitzen und so für eine höhere Ausfallsicherheit sorgen.
- Mehrere ZEO-Clients verteilen ggf. auftretende Last in der Anwendungslogik.
- ZEO-Clients können unterschiedliche Aufgaben übernehmen und speziell dafür konfiguriert werden, z.B. für anonyme Betrachter, Redakteure und Administratoren.

Konfiguration

Wir verwenden unser bisheriges Buildout-Projekt und ergänzen es um eine Konfiguration für den Produktivbetrieb. Hierzu erstellen wir eine weitere Konfigurationsdatei `deploy.cfg`:

```
[buildout]
extends =
    base.cfg

parts =
    zeoserver
    instance1

[zeoserver]
recipe = plone.recipe.zeoserver
zeo-address = 127.0.0.1:8000
blob-storage = ${buildout:directory}/var/blobstorage

[instance-base]
zeo-client = True
zeo-address = ${zeoserver:zeo-address}
blob-storage = ${zeoserver:blob-storage}
shared-blob = on
zserver-threads = 4
http-fast-listen = off
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[instance1]
<= instance-base
http-address = 8010
debug-mode = off
verbose-security = off
```

Ändern von IP und Ports

[zeoserver]

zeo-address gibt die Adresse des ZEO-Servers an.

Der Standardwert ist 8100.

[instance1], [instance-profile], [instance-debug]

zeo-client wird der Wert auf `true` gesetzt, wird aus der Instanz ein ZEO-Client, der auf einen ZEO-Server mit einer bestimmten `zeo-address` verweist.

http-address Port des HTTP-Servers. Der Standardwert ist `8080`.

ip-address ist die Standard-IP-Adresse, an der der ZEO-Client auf Anfragen horcht. Ist kein Wert angegeben, lauscht Zope auf allen IP-Adressen der Maschine. Die Anweisung kann überschrieben werden in den Server-Konfigurationen `<http-server>` etc. Üblicherweise ist keine IP-Adresse angegeben.

zeo-address gibt die Adresse des ZEO-Servers an, z.B. `212.42.230.152:8100`.

Der Standardwert ist `8100`.

Temporary Storage

In `temporary storages` werden z.B. Session-Daten gespeichert. Da diese nicht für jede einzelne Instanz gespeichert werden sollten sondern zentral, können Sie auf den ZEO-Server verlagert werden.

1. Zunächst sollte hierzu jede Instanz so konfiguriert werden, dass sie den `temporary storage` auf dem ZEO-Server anlegen soll:

```
eggs =
    tempstorage
zodb-temporary-storage =
    <zodb_db temporary>
        # Temporary storage database (for sessions)
        <zeoclient>
            server ${zeoserver:zeo-address}
            storage temp
            name zeostorage
            var ${buildout:directory}/var/filestorage
        </zeoclient>
        mount-point /temp_folder
        container-class Products.TemporaryFolder.TemporaryContainer
    </zodb_db>
```

Verändern eines bestehenden Abschnitts

Um einen bestehenden Abschnitt zu ergänzen, kann mit += z.B. der PDBDebugMode für den zweiten ZEO-Client hinzugefügt werden:

```
[instance-profile]
<= instance-base
...
environment-vars +=
    PROFILE_PUBLISHER 1
```

Umgekehrt können auch einzelne Werte entfernt werden:

```
eggs -=
    Products.PDBDebugMode
```

[zeoserver] verwendet `plone.recipe.zope2zeoserver`, um einen ZEO-Server in `parts/zeoserver` aufzusetzen.

zope2-location verweist auf die im zope2-Abschnitt angegebene Installation.

zeo-address gibt den Port des ZEO-Servers an, der Standardwert ist 8100.

Es können auch noch weitere Konfigurationsoptionen angegeben werden, z.B.

zeo-conf Ein relativer oder absoluter Pfad zur `zeo.conf`-Datei. Wird kein Pfad angegeben, wird eine `zeo.conf`-Datei mit den in `[zeoserver]` angegebenen Werten generiert.

zeo-conf-additional Zusätzliche Angaben zur `zeo.conf`-Datei. Dabei müssen die nachfolgenden Zeilen eingerückt sein.

Einen vollständigen Überblick über alle Optionen erhalten Sie in plone.recipe.zope2zeoserver.

[instance], [instance2] verwenden plone.recipe.zope2instance

zeo-client wird der Wert auf `true` gesetzt, wird aus der Instanz ein ZEO-Client, der auf einen ZEO-Server mit einer bestimmten `zeo-address` verweist.

zeo-address gibt die Adresse des ZEO-Servers an, der Standardwert ist 8100.

Meist empfiehlt es sich, den Wert aus dem `zeoserver`-Abschnitt zu übernehmen:

```
${zeoserver:zeo-address}
```

zodb-cache-size Anzahl der Objekte, die der ZEO-Client im Cache halten kann.

debug-mode, verbose-security Damit die instance-Instanz die Daten ausliefert und die `instance2`-Instanz zum Debuggen verwendet werden kann, werden nur für die `instance2`-Instanz die Werte auf On gesetzt.

zope-conf-additional erlaubt weitere Einstellungen der Zope-Konfiguration, in unserem Fall werden für den zweiten ZEO-Client die `zserver-threads` auf `1` heruntergesetzt. Debugging und Maintenance werden deutlich vereinfacht, da immer nur eine Anfrage gleichzeitig abgearbeitet wird.

Eine Übersicht über die für zope2instance verfügbaren Optionen erhalten Sie in http://pypi.python.org/pypi/plone.recipe.zope2instance.

Verschieben des Buildout-Projekts auf einen Produktivserver

Die Buildout-Umgebung unseres Projekts kann nun auf den Produktivserver verschoben werden. Hierzu sind mindestens folgende Dateien erforderlich:

`bootstrap.py` erstellt die Struktur des Buildout-Projekts einschließlich `bin/buildout`.

`base.cfg`, `devel.cfg`, `deploy.cfg`, `versions.cfg` die Konfigurationsdateien.

`src/` das Verzeichnis, das die gesamte Eigenentwicklung des Projekts enthält.

Anschließend kann das Projekt neu erstellt werden mit:

```
$ python2.7 bbootstrap.py -c deploy.cfg
$ ./bin/buildout -c deploy.cfg
```

Würde die Konfigurationsdatei nicht spezifiziert, würde Buildout die Standard-Konfigurationsdatei `buildout.cfg`-Datei erwarten.

Anschließend können ZEO-Server und ZEO-Client gestartet werden:

```
$ ./bin/zeoserver start
$ ./bin/instance1 start
```

Nun sollte Zope über den Port 8010 erreichbar sein. Falls dies nicht der Fall sein sollte, können Sie statt `start` auch `fg` verwenden, um die Prozesse im Vordergrund laufen zu lassen und eventuelle Fehlermeldungen auf der Konsole ausgegeben zu bekommen.

Anmerkung 1: Wird die Zope-Instanz unter Linux oder Mac OS X von root gestartet, muss in der `buildout.cfg`-Datei im `[instance]`-Abschnitt eine Direktive für `effective-user` angegeben werden, an dessen User ID der Prozess gebunden wird, nachdem die Ports zugewiesen wurden, z.B.:

```
[instance]
...
effective-user = plone
```

So können für die Zope-Instanz auch Ports mit Nummern kleiner 1024 verwendet werden.

Anmerkung 2: Unter Windows lässt sich eine Zope-Instanz als Service installieren, z.B. mit:

```
> bin\instance install
```

Konfigurieren des NFS für blobstorage

`zeoserver`: In `/etc/exports` kann folgendes eingetragen werden:

```
/plone/vs_buildout/var/blobstorage 192.168.110.0/24(rw)
```

Damit erlaubt der NFS-Server `zeoserver` NFS-Exporte an Server des internen Netzes `192.168.110.0/24`.

Anschließend wird der NFS-Server neu gestartet mit:

```
# service nfs restart
NFS-Daemon beenden:                                        [  OK  ]
NFS mountd beenden:                                        [  OK  ]
NFS-Dienste beenden:                                       [  OK  ]
NFS-Dienste starten:                                       [  OK  ]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
NFS-mountd starten:                                        [  OK  ]
NFS-Daemon starten:                                        [  OK  ]
```

instance1, instance-profile, instance-debug Hier kann das NFS gemountet werden, z.B. mit:

```
mount -t nfs4 192.168.110.3:/plone/vs_buildout/var/blobstorage /plone/vs_buildout/
↪var/blobstorage
```

oder `/etc/mtab` konfigurieren:

```
...
192.168.110.3:/plone/vs_buildout/var/blobstorage /plone/vs_buildout/var/
↪blobstorage nfs4 rw,addr=192.168.110.3,clientaddr=192.168.110.4 0 0
```

Zusätzliche Informationen für Windows

Für Windows sind entsprechende Dienste für den ZEO-Cluster einzurichten.

ZEO-Server

Für den ZEO-Server wird der Dienst erstellt mit:

```
> bin\zeoserver_service  install
```

bzw. in Plone-Versionen < 3.3:

```
> bin\zeoservice install
```

Üblicherweise werden die Services mit `--startup auto` intstalliert, sodass sie beim Systemstart automatisch gestartet werden.

ZEO-Clients

```
> bin\instance install
> bin\instance2 install
```

Zum Deinstallieren dieser Services können Sie einfach folgendes aufrufen:

```
> bin\instance remove
> bin\instance2 remove
```

WebDAV-Server

Um einen WebDAV-Server zu konfigurieren kann z.B. im `instance`-Abschnitt folgendes angegeben werden:

```
webdav-address = 8091
webdav-force-connection-close = off
```

enable-ms-author-via Wird der Wert auf `true` gesetzt, können auch ältere Microsoft Web Folders- und Microsoft Office-Versionen sich mit dem Zope-Server verbinden. Der Standardwert ist `off`, da hierdurch einige standardkonforme Anfragen schwierig werden. Weitere Informationen erhalten Sie unter http://www.zope.org/Collectors/Zope/1441

enable-ms-public-header Wird der Wert auf `true` gesetzt, wird ein `Public`-Header als Antwort auf eine WebDAV-OPTIONS-Anfrage gesendet:

```
zope-conf-additional =
    ...
    enable-ms-public-header on
```

Versionen von Microsofts Web Folders ab Januar 2005 benötigen diese Header-Angabe (s.a.: http://www.redmountainsw.com/wordpress/archives/webfolders-zope).

Apache für WebDAV konfigurieren

Mit dem Webserver Apache 2 lässt sich der Zugriff auf WebDAV-Ressourcen komfortabel einrichten. Benötigt werden die Module `dav` und `dav_fs`, die im Lieferumfang enthalten sind.

Die Standardkonfiguration umfasst neben der Anweisung zum Laden der Module lediglich eine Zeile zum Pfad der Datenbank für `DAVLockDB`.

Nach dem Neustart des Webservers ist die Plonesite unter der browser-üblichen Adresse mit `http://` zu erreichen. Die Angabe eines zusätzlichen Ports in der Form `http://www.veit-schiele.de:8091/` entfällt. Die Konfiguration von Clients ist im Plone-Nutzerhandbuch näher beschrieben.

Für Clients, die mit fehlenden ´´/´´ am Ende nicht korrekt umgehen, kann der Apache wie folgt konfiguriert werden:

```
<IfModule mod_setenvif.c>
 BrowserMatch "Microsoft Data Access Internet Publishing Provider" redirect-carefully
</IfModule>
```

Zu den Details der möglichen Konfiguration mit `mod_setenvif` siehe auch Apache-Umgebungsvariablen in der offiziellen Apache-Dokumentation.

Clock-Server

Um einen Clock-Server zu konfigurieren kann z.B. folgendes angegeben werden:

```
zope-conf-additional =
    <clock-server>
        method /mysite/do_stuff
        period 60
        user admin
        password secret
        host localhost
    </clock-server>
```

Für jeden `clock-server`-Abschnitt kann angegeben werden, welcher Nutzer die angegebene Methode aufrufen darf. Im einzelnen:

method Pfadangabe von Zope root zu einer ausführbaren Zope-Methode (Python-Skript, externe Methode etc.) Die Methode muss keine Argumente erhalten.

period Sekunden zwischen jedem Aufruf der Methode. Üblicherweise wird mindestens `30` angegeben.

user ein Zope-Nutzername

password Das Passwort dieses Zope-Nutzers

host Der Name des Host, der im Header eines Requests als `Host:` angegeben wird. Dies kann bei in Zope angegebenen *virtual host rules* nützlich sein.

Um zu überprüfen, ob der `clock-server` läuft, starten Sie die Instanz oder den ZEO-Client im Vordergrund und schauen, ob eine ähnliche Meldung wie die folgende ausgegeben wird:

```
2009-03-03 19:57:38 INFO ZServer Clock server for "/mysite/do_stuff" started (user:␣
↪admin, period: 60)
```

Anmerkung: Ein Clock-Server sollte immer nur für einen ZEO-Client angegeben werden.

Weitere Informationen erhalten Sie unter Clock and asyncronous tasks.

ZODBs konfigurieren

Mehrere Storages verwalten

Jede Zope-Instanz kann mehrere Zope-Datenbanken (ZODB) verwalten. Das Hinzufügen einer neuen ZODB erfolgt in der `deploy.cfg`-Datei, z.B.:

```
[zeoserver]
...
zeo-conf-additional =
    <filestorage extra>
        path  ${buildout:directory}/var/filestorage/Extra.fs
    </filestorage>

[instance]
...
zope-conf-additional =
    <zodb_db extra>
        # Extra database
        cache-size 10000
        allow-implicit-cross-references false
    # ZEOStorage database
        <zeoclient>
          server ${zeoserver:zeo-address}
          storage extra
          name zeostorage
          var ${buildout:directory}/var
          cache-size 500MB
        </zeoclient>
        mount-point /extra
    </zodb_db>

[instance2]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
...
zope-conf-additional =
    ${instance:zope-conf-additional}
    zserver-threads 1
```

`filestorage`-Abschnitt

`path` Pfadangabe der Speicherdatei. Die Pfadangaben weiterer Dateien wie `index`- und `lock`-Dateien werden daraus berechnet.

`read-only` Ist die Angabe `true`, sind nur Lesezugriffe auf diesen Speicher erlaubt. Beachten Sie, dass die `pack`-Operation nicht als schreibender Zugriff angesehen wird und weiterhin erlaubt bleibt.

`quota` Maximal zulässige Größe des Speichers. Operationen, die die angegebene Größe überschreiten würden, führen zu einem `ZODB.FileStorage.FileStorageQuotaError`.

`zodb_db`-Abschnitt

`cache-size` Die Anzahl der Objekte, die für jede Verbindung zwischengespeichert werden können.

Der Standardwert beträgt `5000`.

`pool-size` Die erwartete maximale Anzahl gleichzeitig offener Verbindungen. Dies ist keine harte Begrenzung, jedoch führt eine größere Anzahl von Verbindungen zu einem `warn`-Eintrag in der Log-Datei.

`mount-point` Pfadangabe in dem die Datenbank im Zope-Server eingehängt wird.

Unterscheidet sich die Angabe für den ZEO-Server von der des mount-point, kann folgende Notation verwendet werden:

```
path_on_client:path_on_zeo_server
```

`zeoclient`-Abschnitt

`storage` Verwaltet der ZEO-Server mehr als einen Speicher, wird für den ZEO-Client der Name des Speichers angegeben, den der benutzen möchte. Der Standardwert für ist `1`, der für die `main`-ZODB vergeben wird.

`name` Der Name des Speichers. Wird keine Angabe gemacht, wird die Adresse des Servers verwendet.

`var` Das Verzeichnis, in dem die persistenten Cache-Dateien gespeichert werden. Üblicherweise werden die Cache-Dateien im aktuellen Verzeichnis gespeichert.

`cache-size` Die maximale Größe des Client-Cache in bytes, KB oder MB.

Der Standardwert beträgt `20 MB`.

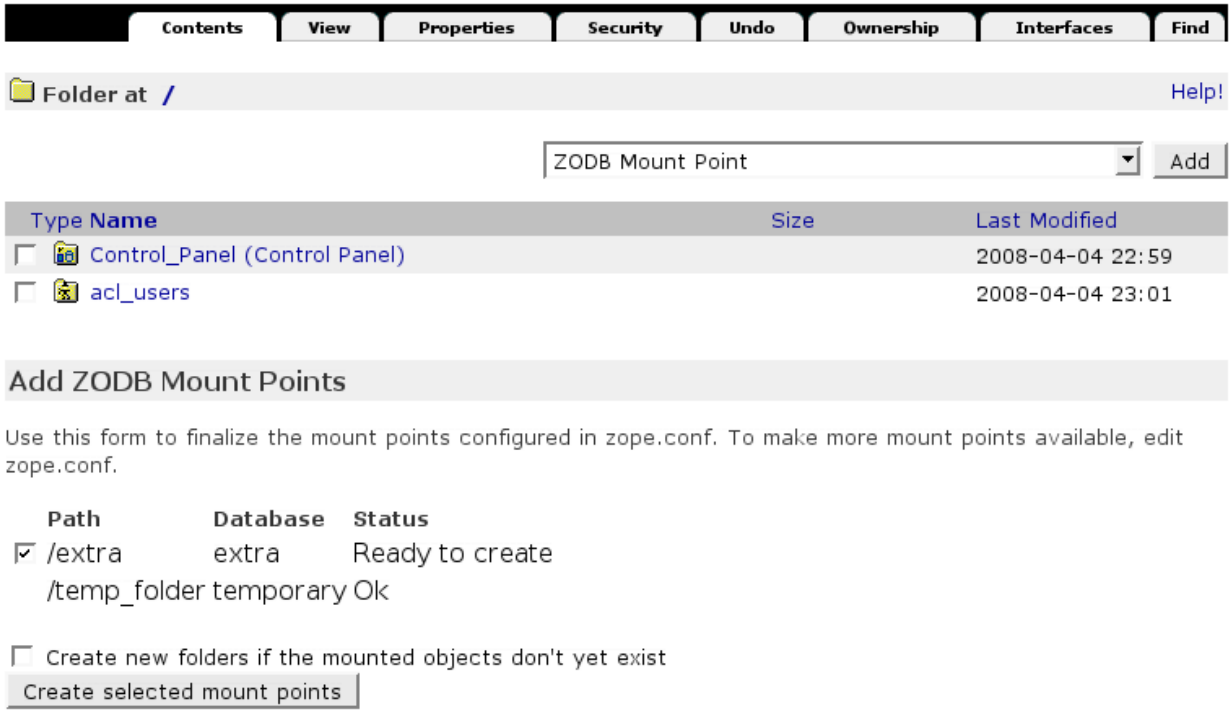
`read-only` Markierung, die anzeigt, ob dies ein Speicher mit ausschließlichem Lesezugriff ist.

Der Standardwert ist `false`.

Eine Übersicht über verfügbare Angaben erhalten Sie in `parts/zope2/lib/python/ZODB/component.xml`.

Damit die Konfiguration wirksam wird, muss nun das Buildout-Skript erneut aufgerufen werden:

```
$ ./bin/buildout -Noc deploy.cfg
```



Schließlich kann im Zope Management Interface (ZMI) ein neuer *Mount Point* hinzugefügt werden.

collective.recipe.filestorage

collective.recipe.filestorage ist ein Rezept zum Verwalten mehrerer ZODBs:

```
[buildout]
parts =
    filestorage
    ...

[filestorage]
recipe = collective.recipe.filestorage
parts =
    extra
    super
```

parts Eine Liste von filestorage-Abschnitten. Im Gegensatz zu `[buildout]`-Abschnitten muss jeder Abschnitt in einer neuen Zeile beginnen.

zeo Der Name der `plone.recipe.zope2zeoserver` oder `plone.recipe.zeoserver`-Abschnitte, für die zusätzliche Filestorages definiert werden sollen.

Der Standardwert ist der erste Abschnitt in der Buildout-Konfiguration.

zopes Eine Liste der Abschnitte, die `plone.recipe.zope2instance` verwenden und in denen die Filestorages hinzugefügt werden sollen.

Üblicherweise werden alle Abschnitte mit `plone.recipe.zope2instance` hinzugefügt, die dem zugehörigen `zeoserver`-Abschnitt zugeordnet sind.

zodb-cache-size Die Zahl der Objekte, die maximal im ZODB-Cache gehalten werden sollen.

Der Standardwert ist `5000`.

zodb-name Der Name der ZODB.

Der Standardwert ist``%(fs_part_name)s``.

zodb-mountpoint Der Pfad zu den Mount-Points.

Der Standardwert ist `/%(fs_part_name)s`.

Wollen Sie z.B. jeder Datei noch die Endung `_mountpoint` hinzufügen, ist die Angabe `/%(fs_part_name)s_mountpoint`.

zeo-address Port des zugehörigen ZEO-Servers.

Der Standardwert ist `8100`.

Sinnvollerweise wird er jedoch aus dem `zeoserver`-Abschnitt genommen:

```
${zeoserver:zeo-address}
```

zeo-client-cache-size Die maximale Größe des ZEO-Client-Cache in `bytes`, `KB` oder `MB`.

Der Standardwert ist `30MB`.

zeo-storage Die ID des ZEO-Storages.

Standardwert ist `%(fs_part_name)s`.

Soll dem Namen noch die Endung `_storage` hinzugefügt werden, so muss `%(fs_part_name)s_storage` angegeben werden.

zeo-client-name Der Name des ZEO-Client.

Der Standardwert ist `%(fs_part_name)s_zeostorage`.

Weitere Konfigurationsmöglichkeiten sind in Supported options beschrieben.

Plone 4

In Plone 4 werden Bilder und Dateien als *Binary Large Objects (BLOBs)* im Dateisystem abgelegt. Daher sind auch für die jeweiligen Mount-Points passende Speicherorte im Dateisystem anzugeben. Hierbei vereinfacht `collective.recipe.filestorage` die Buildout-Konfiguration erheblich, da die Angaben für jeden einzelnen Mount-Point nicht mehr im `zeoserver`-und jedem `instance`-Abschnitt angegeben werden müssen:

```
[buildout]
parts =
    filestorage
    ...

[filestorage]
recipe = collective.recipe.filestorage
blob-storage = ${buildout:directory}/var/blobstorage-%(fs_part_name)s
parts =
    extra

[zeoserver]
...
blob-storage = ${buildout:directory}/var/blobstorage-%(fs_part_name)s
```

ZODB packen

Die Zope Objects Database (ZODB) speichert die Daten, indem sie neue Transaktionen anhängt. Daher wächst die ZODB mit jeder Änderung weiter an, auch wenn Dateien gelöscht oder Transaktionen rückgängig gemacht werden. Durch das Packen der ZODB werden nur noch die Ergebnisse der Transaktionen bis zu einem bestimmten Zeitpunkt zusammengefasst. Dabei wird der Zeitpunkt in Tagen angegeben, für die die Transaktionen noch erhalten werden.

Mit ZEO lässt sich das Packen der Datenbank einfach Automatisieren mit dem Skript `ClientStorage.py` in `eggs/ZODB3-3.9.5-py2.6-linux-x86_64.egg/ZEO/`. Und `plone.recipe.zeoserver` stellt mit `bin/zeopack` auch einen Wrapper mit den nötigen Pfadangaben in unserem Buildout-Projekt bereit:

```
$ ./bin/zeopack days=7
```

Dabei lässt sich mit `days` die Anzahl der Tage angeben, für die alle älteren Objekte gepackt werden sollen.

Nach dem Packen ist die bisherige ZODB im Dateisystem verschoben worden nach `var/filestorage/Data.fs.old`. Soll also durch das Packen Festplattenspeicher gewonnen werden, muss diese Datei noch gelöscht werden.

Bemerkung: In Plone 3 befindet sich das Skript `ClientStorage.py` in `parts/zope2/lib/python/ZEO/`.

Cron Job definieren

Schließlich kann mit `crontab -e` noch ein Cron Job definiert werden, der das Skript regelmäßig aufruft. Fügen Sie in der Tabelle z.B. folgende Zeile hinzu um die Datenbank jeden Montag um 0:05 Uhr zusammenzupacken und alle Änderungen älter als 7 Tage zu löschen:

```
5 0 * * 1 /home/veit/myproject/bin/zeopack days=7
```

Weitere Einstellmöglichkeiten erhalten Sie mit `man 5 crontab`.

Dieser Eintrag kann auch automatisiert mit dem Rezept `z3c.recipe.usercrontab` erstellt werden. Hierzu wird in der `deploy.cfg` folgendes eingetragen:

```
[buildout]
parts =
    ...
    zeopack-crontab
...
[zeopack-crontab]
recipe = z3c.recipe.usercrontab
times = 5 0 * * 1
command = ${buildout:bin-directory}/zeopack days=7
```

Mehrere ZODBs packen

Sind zusätzliche ZODB-Mount-Points definiert worden, so sollten diese ebenfalls gepackt werden können. Hierfür ist dann jedoch ein eigenes Skript notwendig, z.B. `zeopackall`:

```
#!/home/veit/myproject/bin/zopepy

username = None
blob_dir = "/home/veit/myproject/var/blobstorage-%(fs_part_name)s"
realm = None
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
storages = '1','extra'
days = "7"
unix = None
address = "localhost:8100"
host = "localhost"
password = None
port = "8100"
import getopt; opts = getopt.getopt(sys.argv[1:], 'S:B:W1')[0];
opts = dict(opts)
storage = opts.has_key('-S') and opts['-S'] or '1'
blob_dir = opts.has_key('-B') and opts['-B'] or blob_dir

import plone.recipe.zeoserver.pack

for storage in storages:
    print 'Packing storage %s' % storage
    plone.recipe.zeoserver.pack.main(host, port, unix, days, username, password,␣
↪realm, blob_dir, storage)
```

storages Liste der Namen der zu packenden ZODBs.

1 ist der Standardwert für die `main`-Datenbank.

Bemerkung: `plone.recipe.zeoserver` steht `zopepy` üblicherweise nicht zur Verfügung. Daher sollte in der `deploy.cfg`-Datei folgendes eingetragen werden:

```
[zeoserver]
eggs = plone.recipe.zeoserver
...
[zopepy]
...
eggs =
    ${instance:eggs}
    ${zeoserver:eggs}
```

Und auch im Abschnitt `zeopack-crontab` sollte auf das neue Skript verwiesen werden:

```
[zeopack-crontab]
...
command = ${buildout:directory}/zeopackall
```

Bemerkung: Für Plone 3 sollte statt `plone.recipe.zeoserver` das Rezept `plone.recipe.zope2zeoserver` verwendet und die Zeilen mit `blob_dir` im `zeopackall`-Skript gelöscht werden.

Backup der ZODB

Zope bringt mit `repozo.py` ein Skript mit, das das Backup der ZODB im laufenden Betrieb ermöglicht. Es befindet sich wie schon `zeopack.py` in `parts/zope2/utilities/ZODBTools/`. Zudem lassen sich mit `repozo.py` auch inkrementelle Backups erstellen. `plone.recipe.zope2instance` erstellt einen Wrapper `bin/repozo`.

Um nun ein inkrementelles Backup zu erzeugen, erstellen wir zunächst das Verzeichnis `backups`, bevor wir `repozo` mit den entsprechenden Parametern aufrufen:

```
$ mkdir backups
$ ./bin/repozo -BvzQ -r backups -f var/filestorage/Data.fs
```

Soll ein Backup wieder zurückgespielt werden, sollte die Zope-Instanz zunächst gestoppt werden, eine Kopie der voraussichtlich korrupten `Data.fs` erstellt werden und erst dann `repozo` aufgerufen werden:

```
$ ./bin/repozo -Rv -r backups -o Data.fs
```

Anmerkung 1: Da `repozo` nach jedem Packen der ZODB wieder nur ein vollständiges Backup durchführen kann, empfiehlt sich das Packen deutlich weniger häufig als das Backup.

Dieser Eintrag kann auch automatisiert mit dem Rezept `z3c.recipe.usercrontab` erstellt werden. Hierzu wird in der `deploy.cfg` folgendes eingetragen:

```
[buildout]
parts =
    ...
    backup-crontab
...
[backup-crontab]
recipe = z3c.recipe.usercrontab
times = 15 0 * * *
command = ${buildout:bin-directory}/repozo  -BvzQ -r ${buildout:directory}/backups -f
↪${buildout:directory}/var/filestorage/Data.fs
```

Backup mehrerer ZODBs einer Instanz

Mit collective.recipe.backup wird ein Skript erstellt, das für mehrere ZODBs Backups erstellen kann, z.B. zusätzlich für den Katalog in eigener ZODB:

```
[buildout]
parts =
    ...
    backup
...
[backup]
recipe = collective.recipe.backup
additional_filestorages =
    Extra
    Super
```

Falls zum Anlegen mehrer Mount-Points `collective.recipe.filestorage` verwendet wurde, kann der `[backup]`-Abschnitt auch vereinfacht werden:

```
[backup]
recipe = collective.recipe.backup
additional_filestorages = ${filestorage:parts}
```

Folgende zusätzliche Optionen bietet `collective.recipe.backup`:

`location` Ort, an dem die Backups gespeichert werden.

Der Standardwert ist `var/backups` innerhalb des Buildout-Verzeichnisses.

Bei der expliziten Verwendung von `location` ist zu beachten, dass der letzte Teil der Angabe als Präfix verwendet wird. Die Angabe:

```
location = ${buildout:directory}/backups
```

werden im Ordner des Buildout-Projekts die Unterordner `backups_Catalog` und `backups_Extra` erzeugt. Diese enthalten dann die Backups der jeweiligen Datenbank.

`keep` Anzahl der vollständigen Backups, die aufbewahrt werden.

Der Standardwert ist `2`.

Alle älteren Backups einschließlich ihrer inkrementellen Backups werden automatisch gelöscht.

Wird der Wert auf `0` gesetzt, werden alle Backups aufbewahrt.

`datafs` Falls sich die Data.fs nicht im Standardordner var/filestorage/Data.fs befindet kann der Pfad mit dieser Option überschrieben werden.

`full` Üblicherweise werden inkrementelle Backups erstellt. Wird der Wert hier auf `true` gesetzt, werden jedesmal vollständige Backups erstellt.

`debug` In seltenen Fällen sollte in die Log-Datei im `debug`-Level geschrieben werden. Dann sollte hier der Wert auf `true` gesetzt werden.

`snapshotlocation` Ort, an dem die Schnappschüsse gespeichert werden sollen.

Der Standardwert ist `var/snapshotbackups` innerhalb des Buildout-Verzeichnisses. Bei expliziter Festlegung gelten bezüglich des Pfads dieselben Regeln für das Ordner-Präfix wie bei `location`.

`gzip` Der Standardwert ist `true`.

Dabei ist die Endung gezippter ZODBs `*.fsz` und nicht `*.fs.gz`.

`additional_filestorages` Hier können Sie zusätzliche Angaben machen, z.B. wenn Sie Ihren Katalog in eine eigene ZODB ausgelagert oder weitere ZODBs als Mount-Point eingebunden haben.

Bei Verwendung von `collective.recipe.backup` nach diesem Muster ändert man den `command` im Abschnitt `[backup-crontab]` auf:

```
[backup-crontab]
...
command = ${buildout:bin-directory}/backup -q
```

Löschen alter Backups

Alte Backups sollten nach einer bestimmter Zeit wieder gelöscht werden. In unserem folgenden Beispiel werden inkrementelle Backups nach zwei Wochen und vollständige Backups nach fünf Wochen gelöscht:

```
[buildout]
parts =
    ...
    remove-incremental-backups
    remove-full-backups
...
[remove-incremental-backups]
recipe = z3c.recipe.usercrontab
times = 8 0 * * *
command = find ${buildout:directory}/backups -name \*deltafs -ctime +14 -delete

[remove-full-backups]
recipe = z3c.recipe.usercrontab
times = 8 0 * * *
command = find ${buildout:directory}/backups -name \*dat -ctime +35 -delete
```

Blob-Storages

Mit `plone.recipe.backup` ab Version 2.0 lassen sich auch Sicherungskopien des Blob-Storage anlegen. Plone speichert seit ersion 4.0 üblicherweise alle Bilder und Dateien (*Binary large objects*) im Dateisystem. Daher müssen von diesen Blob-Storages ebenfalls Sicherheitskopien erstellt werden. Falls der Speicherort der Blob-Storages nicht aus `plone.recipe.zope2instance` hervorgeht, kann mit `blob_storage` auch explizit der Pfad angegeben werden:

```
[buildout]
parts =
    instance
    backup

[instance]
recipe = plone.recipe.zope2instance
user = admin:admin
blob-storage = ${buildout:directory}/var/blobstorage

[backup]
recipe = collective.recipe.backup
```

Falls erforderlich, kan Buildout verschiedene Skripts zum Erstellen der Sicherungskopien für die ZODBs und die Blob-Storages erstellen:

```
[buildout]
parts =
    ...
    filebackup
    blobbackup

[filebackup]
recipe = collective.recipe.backup
backup_blobs = false
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[blobbackup]
recipe = collective.recipe.backup
blob_storage = ${buildout:directory}/var/blobstorage
only_blobs = true
```

Folgende Attribute kamen neu hinzu:

blob-storage Verzeichnis, in dem die Blob-Storages gespeichert werden.

Diese Option wird ignoriert, falls `backup_blobs = false`.

Falls nichts für `blob-storage` angegeben wird, wird versucht, einen Wert zu ermitteln aus einem Abschnitt, in dem eines der folgenden Rezepte verwendet wird:

- `plone.recipe.zeoserver`
- `plone.recipe.zope2instance`
- `plone.recipe.zope2zeoserver`

blob_storage Alternative Schreibweise für `blob_storage` da `plone.recipe.zope2instance` ebenfalls diese Variable verwendet, in `collective.recipe.backup` jedoch Unterstriche verwendet werden.

backup_blobs Sofern ein Wert für `blob-storage` angegeben ist oder ermittelt werden kann, werden üblicherweise Sichrungskopien der Blob-Storages erstellt. Mit `backup_blobs = false` kann dies unterbunden werden.

blobbackuplocation Verzeichnis, in dem die Sicherungskopien gespeichert werden.

Der Standardwert ist `var/blobstoragebackups` innerhalb des Buildout-Verzeichnisses.

blobsnapshotlocation Verzeichnis, in dem die Schnappschüsse erstellt werden.

Der Standardwert ist `var/blobstoragesnapshots` im Buildout-Verzeichnis.

only_blobs Es wird ausschließlich ein Backup der Blob-Storages erstellt, nicht der ZODBs.

Der Standardwert ist `false`.

use_rsync Das Programm `rsync` mit *Hard Links* zum Erstellen der Blob-Backups wird verwendet.

Der Standardwert ist `true`.

Sofern `rsync` nicht installiert ist, oder *Hard Links* nicht funktionieren (*Windows*), sollte dieses Attribut auf `false` gesetzt werden. Dann wird eine einfache Kopie mit Pythons `shutil.copytree` erstellt.

Mehrere Blob-Storages

Aktuell unterstützt `collective.recipe.backup` nicht zusätzliche Blob-Storages. Für diese müsste ggf. ein eigener Buildout-Abschnitt erstellt werden, der ein zweites Set von Backup-Skripten erstellt, z.B.:

```
[extrablobbackup]
recipe = collective.recipe.backup
blob_storage = ${buildout:directory}/var/extrablobstorage
only_blobs = true
```

rsync

Üblicherweise verwendet `collective.recipe.backup` `rsync` zum Erstellen der Backups. Dabei werden sog. *hard links* erstellt um Plattenplatz zu sparen und inkrementelle Backups zu erzeugen. Dies erfordert jedoch Linux/Unix oder Mac OS X.

`rsync` kann nun auch verwendet werden um Backups auf entfernte Hosts zu erstellen: rsync-backup.sh

Für Windows liegen uns zum aktuellen Zeitpunkt keine Erfahrungen vor, auf der Basis von Cygwin sollte es jedoch auch auf Windows-Systemen lauffähig sein. Falls nicht, kann `use_rsync = false` gesetzt werden und das Blob-Storage-Verzeichnis wird nach dem Backup einfach kopiert.

collective.recipe.rsync

Alternativ kann das Rezept collective.recipe.rsync verwendet werden. Hierzu kann z.B. die Datei `rsync.cfg` mit folgendem Inhalt erstellt werden:

```
[rsync-file]
recipe = collective.recipe.rsync
source = veit-schiele.de:/srv/www.veit-schiele.de/var/filestorage/Data.fs
target = var/filestorage/Data.fs
script = true

[rsync-blob]
recipe = collective.recipe.rsync
source = veit-schiele.de:/srv/www.veit-schiele.de/var/blobstorage/
target = var/blobstorage/
script = true
```

`script` Üblicherweise ruft `collective.recipe.rsync` `rsync` während der Installation des Rezepts auf. Sofern ein entsprechendes Skript erstellt wird, kann dieses später z.B. als Cronjob aufgerufen werden um `rsync` auszuführen. Hierbei ist lediglich darauf zu achten, dass `rsync-file` vor `rsync-blob` ausgeführt wird.

`port` Optional kann ein alternativer Port für `rsync` angegeben werden.

Siehe auch:

Weitere Informationen zu `rsync` erhaltet Ihr in den Artikel von Mike Rubel: Easy Automated Snapshot-Style Backups with Linux and Rsync.

ZODB reparieren

Eine Zope-Datenbank kann z.B. durch einen Systemabsturz oder einen Festplattendefekt korrumpiert werden. Dies macht sich meist durch einen POSKeyError oder einen CorruptedError und nicht mehr bearbeitbare Objekte bemerkbar.

CorruptedError

Dieser Fehler kann verschiedene Ursachen haben, z.B. falsche Längen oder Zeiten von Transaktionen.

`fsrecover.py` ist ein Skript, das die Integrität von Transaktionen überprüft und diejenigen mit korrupten Daten entfernt. Daher ist es auch nicht für *POSKeyErrors* geeignet sondern empfiehlt sich vielmehr für *CorruptedErrors*. Darüberhinaus kann es auch zu weiteren *POSKeyErrors* führen wenn eine fehlerhafte Transaktion entfernt wird und dadurch den Verweis auf ein nicht mehr vorhandenes Objekt zurücklässt:

```
$ ./bin/zopepy -m ZODB.scripts.fsrecover -P 0 var/filestorage/Data.fs var/filestorage/
↪Data.fs.recovered &> logrecover.txt
```

In `logrecover.txt` können Sie anschließend nachschauen, wieviele Daten Ihnen verloren gingen, z.B.:

```
Recovering var/filestorage/Data.fs into var/filestorage/Data.fs.recovered
. 1 . 2 . 3 . 4 . 5 . 6 . 7 . 8 . 9 . 0
0 bytes removed during recovery
Packing ...
```

POSKeyError

Um diesen Fehler zu verstehen ist es wichtig zu wissen, dass jedes Objekt in der Datenbank eine eindeutige ID (`OID`) zugewiesen bekommen hat. Diese `OID` ist eine binäre Zahl, wie z.B. `0x40A90L`, die auf ein serialisiertes Objekt verweist. Bei einem *POSKeyError* kann nun für eine `OID` kein passendes Objekt gefunden werden. So speichert z.B. ein Ordner, der von `OFS.ObjectManager` abgeleitet ist, die enthaltenen Objekte als Werte des `_objects`-Attributs. Die daraus resultierende Liste wird beim Speichern in eine Liste von OIDs übersetzt. Kann nun beim Laden von `objectValues()` eine OID nicht mehr einem serialisierten Objekt zugewiesen werden, wird ein *POSKeyError* ausgegeben.

1. Mit zc.zodbdgc kommt ein Skript mit, das die Überprüfung von mehreren ZODBs erlaubt: `multi-zodb-check-refs`. Dabei traversiert es ab der Wurzel durch die gesamten Datenbanken. Dies soll sicherzustellen, dass alle Objekte erreichbar sind und jedes nicht-erreichbare Objekt protokolliert werden kann. Darüberhinaus wird bei Blob-Eintragen überprüft, ob ihre Dateien geladen werden können.
2. Zum Installieren von zc.zodbdgc wird zunächst eine `virtualenv`- Umgebung aufgesetzt:

   ```
   $ easy_install-2.6 virtualenv
   $ virtualenv --no-site-packages zeo_check
   ```

3. Anschließend wird in dieser `virtuelenv`-Umgebung `zc.zodbdgc` installiert:

   ```
   $ cd zeo_check
   $ ./bin/easy_install zc.zodbdgc
   ```

4. Packen Sie anschließend Ihre ZODB und kopieren diese in Ihre `virtuelenv`- Umgebung.
5. Erzeugen Sie eine Konfigurationsdatei `storages.cfg` mit folgendem Inhalt:

   ```
   <zodb>
     <filestorage my>
        path var/filestorage/my.fs
        blob-dir var/blobstorage-my
     </filestorage>
   </zodb>
   ```

6. Anschließend kann das `multi-zodb-check-refs`-Skript aufgerufen werden mit:

```
$ ./bin/multi-zodb-check-refs storages.cfg
```

Sind alle Referenzen Ihrer Datenbank gültig, so erhalten Sie keine Ausgabe. Bei POSKeyErrors sieht die Ausgabe beispielsweise so aus:

```
!!! main 26798 ?
POSKeyError: 0x68ae
```

Regelmäßige Überprüfung und E-Mail-Benachrichtigung

Dieses Skript sollte nun regelmäßig als Cronjob ausgeführt werden:

```
# Check ZEO Storages
0 6 * * * cd /home/veit/zeo_check; ./bin/multi-zodb-check-refs |mailx -s "Check␣
↪Storages" -c admin@veit-schiele.de
```

Wiederherstellen

1. Möglicherweise können die fehlenden Objekte aus dem Backup zurückgespielt werden.
2. Mit der `-r`-Option erhalten Sie eine Datenbank mit entgegengesetzten Referenzen, womit sich gegebenenfalls herausfinden lässt, welche Objekte fehlen:

   ```
   $ ./bin/multi-zodb-check-refs -r var/filestorage/refdb.fs storages.cfg
   !!! main 26798 main 16717
   POSKeyError: 0x68ae
   ```

3. Nun schreiben Sie eine `refdb.cfg` mit folgendem Inhalt:

   ```
   <zodb main>
       <filestorage 1>
             path /home/veit/zeo_check/var/filestorage/refdb.fs
       </filestorage>
   </zodb>
   ```

4. Anschließend können Sie die Datenbank öffnen:

   ```
   $ ../myproject/bin/zopepy
   >>> import ZODB.config
   >>> db = ZODB.config.databaseFromFile(open('./refdb.cfg'))
   >>> conn = db.open()
   >>> refs = conn.root()['references']
   ```

   Sie dürften nun eine Fehlermeldung wie diese bekommen:

   ```
   !!! main 13184375 ?
   POSKeyError: 0xc92d77
   ```

5. Nun können Sie die OID desjenigen Objekts herausfinden, von dem aus referenziert wird:

   ```
   >>> parent = list(refs['main'][13184375])
   >>> parent
   [13178389]
   ```

6. Wird nun dieses Objekt geladen, sollten Sie einen POSKeyError erhalten:

```
>>> app._p_jar.get('13178389')
2010-07-16 15:30:18 ERROR ZODB.Connection Couldn't load state for 0xc91615
Traceback (most recent call last):
...
ZODB.POSException.POSKeyError: 0xc92d77
```

7. Wir können jedoch die aktuellen Daten des Elternobjekts laden um eine Vorstellung von diesem Objekt zu erhalten:

```
>>> app._p_jar.db()._storage.load('\x00\x00\x00\x00\x00\xc9\x16\x15', '')
('cBTrees.IOBTree
IOBucket
q\x01.((J$KT\x02ccopy_reg
_reconstructor
q\x02(cfive.intid.keyreference
KeyReferenceToPersistent
...
```

8. Nun erzeugen wir ein Fake-Objekt, das dieselbe OID (`13184375`) wie das fehlenden Objekt hat mit:

```
$ ./bin/instance-debug debug
Starting debugger (the name "app" is bound to the top-level Zope object)
...
>>> import transaction
>>> transaction.begin()
>>> from ZODB.utils import p64
>>> p64(26798)
'\x00\x00\x00\x00\x00\x00h\xae'
>>> from persistent import Persistent
>>> a = Persistent()
>>> a._p_oid = '\x00\x00\x00\x00\x00\x00h\xae'
>>> a._p_jar = app._p_jar
>>> app._p_jar._register(a)
>>> app._p_jar._added[a._p_oid] = a
>>> transaction.commit()
```

9. Sie sollten nun wieder das Objekt selbst wie auch das Elternobjekt aufrufen können:

```
>>> app._p_jar.get('\x00\x00\x00\x00\x00\x00h\xae')
<persistent.Persistent object at 0xab7f9cc>
>>> app._p_jar.get('\x00\x00\x00\x00\x00\xc9\x16\x15')
BTrees.IOBTree.IOBucket([(39078692, <five.intid.keyreference...
```

10. Schließlich sollten Sie noch die Verbindung zur Datenbank schließen:

```
>>> conn.close()
>>> db.close()
```

Fehlende BLOB-Dateien

Falls Sie die Fehlermeldung erhalten `POSKeyError: 'No blob file'`, hat Mikko Ohtamaa das Skript fix-blobs.py geschrieben, mit dem sich Inhalte aus der ZODB löschen lassen, für die kein BLOB mehr vorhanden ist. Siehe auch Fixing POSKeyError: 'No blob file' content in Plone.

Weitere nützliche Werkzeuge

`analyze.py` zeigt Informationen wie OID, Größe etc. der Objekte in der Datenbank, z.B.:

```
$ ./Processed 123816 records in 2601 transactions
Average record size is 1276.43 bytes
Average transaction size is 60762.18 bytes
Types used:
Class Name                                      Count     TBytes    Pct AvgSize
---------------------------------------------- ------- ---------  ----- -------
AccessControl.User.UserFolder                       1        185   0.0%  185.00
App.ApplicationManager.ApplicationManager           1        189   0.0%  189.00
App.Product.ProductFolder                           1         34   0.0%   34.00
BTrees.IIBTree.IIBTree                           6705    1783379   1.1%  265.98
BTrees.IIBTree.IIBucket                          6957    4584392   2.9%  658.96
...
webdav.LockItem.LockItem                         1203     323529   0.2%  268.94
...PersistentAdapterRegistry                        2       7074   0.0% 3537.00
zope.ramcache.ram.RAMCache                          1        288   0.0%  288.00
============================================== ======= =========  ===== =======
                           Total Transactions    2601                    59.34k
                                Total Records  123816   154338k 100.0% 1276.43
                             Current Objects   74107    78439k  50.8% 1083.87
                                 Old Objects   47124    75898k  49.2% 1649.27
```

`fstest.py` überprüft die Datenbank auf korrupte Transaktionen.

`fsrecover.py` repariert Transaktionsfehler in der Datenbank.

Zum Weiterlesen

- Recovering from BTree corruption
- Inspecting a ZODB to find the causes of bloat
- Introduction to the Zope Object Database
- Finding the last changed object in a ZODB

Wiederherstellen versehentlich gelöschter Objekte

Üblicherweise lassen sich versehentlich gelöschte Objekte in der Zope Object Database (ZODB) mit dm.historical wiederherstellen sofern die ZODB in der Zwischenzeit nicht gepackt wurde.

Im folgenden Beispiel gehen wir davon aus, dass der Ordner `news` in der Plone-Site `Plone` versehentlich gelöscht und nun wiederhergestellt werden soll.

Hierzu fügen Sie zunächst dm.historical im Buildout-Abschnitt der `debug- instance` hinzu:

```
[instance-debug]
<= instance
...
eggs =
    dm.historical
    ...
```

Nachdem das Buildout-Skript durchgelaufen ist, kann die Instanz im Debug-Modus gestartet werden:

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

>>> from DateTime import DateTime
>>> from dm.historical import getObjectAt
>>> site = getObjectAt(app.Plone, DateTime('2014-03-28 14:00:00'))
>>> folder = site['news']
>>> folder.manage_exportObject()
```

1. Damit wird der Zustand der Plone-Site `Plone` zum Zeitpunkt `2014-03-28 14:00:00` aufgerufen.
2. Anschließend wird das Objekt `news` in das Dateisystem exportiert als `news.zexp`.
3. Dann kann die Datei `news.zexp` verschoben werden in den `import-` Ordner von `instance`, also in `${buildout:directory}/var/import/`.
4. Nun können Sie den Ordner `news` im Zope-Management-interface importieren.
5. Schließlich sollten Sie im `portal_catalog` die Site neu indizieren.

Logging

Zope erstellt drei verschiedene Arten von Log-Dateien:

`eventlog` Ereignisse, die Debug-Informationen über in der Zope-Instanz verwendete Produkte enthalten.

`logger access` Zugriffe, mit denen sich Site-Statistiken produzieren lassen.

`logger trace` Detaillierte Informationen über Server-Anfragen (requests).

Die Standardkonfiguration für eine Log-Datei sieht so aus:

```
<eventlog>
    level info
    <logfile>
        path  /home/veit/myproject/var/log/instance.log
        level info
    </logfile>
</eventlog>
```

Mögliche Angaben für `level` sind `critical`, `error`, `warn`, `info`, `debug` und `all`.

Handler

Jeder der drei `logger`-Abschnitte der `zope.conf` kann mehrere `handler`- Abschnitte enthalten.

Fünf verschiedene Handler lassen sich angeben:

- `logfile`
- `syslog`
- `win32-eventlog`
- `http-handler`
- `email-notifier`

Um z.B. eine E-Mail-Benachrichtigung bei Fehlern zu erhalten kann der Eintrag folgendermaßen geändert werden:

```
<eventlog>
    level info
    <logfile>
        path ${buildout:directory}/var/log/instance.log
        level info
    </logfile>
    <email-notifier>
        from zope@veit-schiele.de
        to admin@veit-schiele.de
        subject "Zope Error"
        level error
    </email-notifier>
</eventlog>
```

Mögliche Angaben für jeden Handler sind in `parts/zope2/lib/python/ZConfig/components/logger/handlers.xml` beschrieben.

Packen und Rotieren der Log-Dateien

Plone 4.2.2

Ab Plone 4.2.2 wird Plone mit einer Version von `plone.recipe.zope2instance` ausgeliefert, die die Konfiguration der Log-Rotation in der `deploy.cfg`-Datei angegeben werden, z.B.:

```
[instance-base]
...
event-log-max-size = 5 MB
event-log-old-files = 7
access-log-max-size = 20 MB
access-log-old-files = 7
```

Hiermit werden 7 Generationen der Log-Dateien mit maximal 5 MB für Event-Logs und 10 MB für Access-Logs aufbewahrt.

Plone 4.0

Ab Zope 2.11 kann die Rotation der Zope-Log-Dateien von Zope selbst vorgenommen werden. Hierzu kann in der `deploy.cfg`-Datei z.B. Folgendes angegeben werden:

```
[instance]
recipe = plone.recipe.zope2instance
...
event-log-custom =
    <logfile>
        path ${buildout:directory}/var/log/${:_buildout_section_name_}.log
        when D
        old-files 7
    </logfile>
access-log-custom =
    <logfile>
        path ${buildout:directory}/var/log/${:_buildout_section_name_}-Z2.log
        when D
        old-files 7
    </logfile>
```

Folgende Angaben sind zum Rotieren der Log-Dateien möglich:

`path` Erforderliche Pfadangabe, z.B. `${buildout:directory}/var/log/instance.log`

`old-files` Wieviele alte Log-Dateien sollen aufbewahrt werden?

Der Standardwert ist `0`.

`max-size` Maximale Größe der Log-Datei

`when` Wann sollen die Log-Dateien rotiert werden, z.B. `D` für täglich

`interval` Intervall zwischen den zu rotierenden Log-Dateien

`format` Format der Log-Dateien.

Der Standardwert ist `%(name)s %(message)s` und der Suffix `.log_format`.

Alternativ können auf *ix-Betriebssystemen die Log-Dateien auch mit `logrotate` rotiert werden:

- Ändern Sie die `deploy.cfg` folgendermaßen:

  ```
  [buildout]
  parts =
      ...
      logrotate
      ...
  [logrotate]
      recipe = collective.recipe.template
      input = templates/logrotate.conf.in
      output = ${buildout:directory}/etc/logrotate.conf
  ```

  Mit collective.recipe.template lassen sich Textdateien aus einer Vorlage generieren, wobei die `buildout`-Variablen verwendet werden können.

- Dabei können Sie ein Verzeichnis `templates` und darin eine `logrotate.conf`-Datei erstellen, z.B. mit folgendem Inhalt:

  ```
  daily
  missingok
  ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
rotate 14
mail zope-logs@veit-schiele.de
compress
delaycompress
notifempty
size 1k

${buildout:directory}/var/log/zeoserver.log {
    postrotate
        ${buildout:bin-directory}/zeoserver logreopen
    endscript
}

${buildout:directory}/var/log/instance.log ${buildout:directory}/var/log/instance-
↪Z2.log {
    sharedscripts
    postrotate
        ${buildout:bin-directory}/instance logreopen
    endscript
}

${buildout:directory}/var/log/instance2.log ${buildout:directory}/var/log/
↪instance2-Z2.log {
    sharedscripts
    postrotate
        ${buildout:bin-directory}/instance2 logreopen
    endscript
}
```

- Damit werden alle Log-Dateien in `/home/veit/myproject/var/log/` täglich rotiert;
- Log-Dateien älter als 14 Tage werden gelöscht.
- Die Log-Dateien werden an die E-Mail-Adresse `zope-logs@veit-schiele.de` gesendet.
- Weitere Informationen zu `logrotate` erhalten Sie mit `man logrotate`.

Cron

Ein Eintrag in die crontab mit `crontab -e` könnte z.B. so aussehen:

```
7 0 * * * /usr/sbin/logrotate -s  /home/veit/myproject/var/log/logrotate-status /home/
↪veit/myproject/etc/logrotate.conf
```

Damit wird täglich um 0:07 Uhr logrotate mit den Einstellungen von `/home/veit/myproject/etc/logrotate-zope` aufgerufen und ein Statusbericht in `/home/veit/myproject/var/log/logrotate-status` geschrieben.

Dieser Eintrag kann auch automatisiert mit dem Rezept `z3c.recipe.usercrontab` erstellt werden. Hierzu wird in der `deploy.cfg` folgendes eingetragen:

```
[buildout]
parts =
    ...
    logrotate-crontab
...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[logrotate-crontab]
recipe = z3c.recipe.usercrontab
times = 7 0 * * *
command = /usr/sbin/logrotate -s ${buildout:directory}/var/log/logrotate-status $
↪{buildout:directory}/etc/logrotate.conf
```

Anmerkung: Falls Sie unter Windows eine Version von Zope verwenden, die kleiner als Zope 2.11 ist, können Sie sich ein eigenes Batch-Skript schreiben, das Ihnen die Log-Dateien rotiert. Als Vorbild kann z.B. folgendes Skript genommen werden: Apache for Win32 Log file Rotation.

ZEO-Cluster automatisch starten

Damit der ZEO-Cluster beim Starten des Hosts automatisch mitgestartet wird, legen wir ein Shell-Skript an, das anschließend in das /etc/init.d-Verzeichnis eingebunden wird.

Das Skript `/srv/myproject/zeo` kann z.B. so aussehen:

```
#!/bin/sh
# /etc/rc.d/init.d/zeo
# Startup script for a ZEOCluster
#
# chkconfig: 345 80 20
# description: Zope, a scalable web application server
#
# config: /srv/myproject/deploy.cfg
#
# LSB Source function library
. /lib/lsb/init-functions

RETVAL=0
# list zeo clients in the list below
zeoclients="instance instance2"
# this is for the default install path
clusterpath="/srv/myproject"
prog="ZEOCluster"

start() {
    echo -n $"Starting $prog: "
    output=`${clusterpath}/bin/zeoserver start`
    # the return status of the zeoserver is not reliable, we need to parse
    # its output via substring match
    if echo $output | grep -q "start"; then
            # success
            touch /var/lock/zope/$prog
            log_success_msg "zeoserver started successfully"
            echo
            RETVAL=0
    else
            # failed
            log_failure_msg "zeo failed to start or was already started"
            echo
            RETVAL=1
    fi
    for client in $zeoclients
        do
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            echo -n $"Starting $client: "
            output=`${clusterpath}/bin/${client} start`
            # the return status of the instance is not reliable, we need to parse
            # its output via substring match
            if echo $output | grep -q "start"; then
                # success
                touch /var/lock/zope/${client}
                log_success_msg "$client started successfully"
                echo
                RETVAL=0
            else
                # failed
                log_failure_msg "$client failed to start or was already started"
                echo
                RETVAL=1
            fi
        done
        return $RETVAL
}

stop() {

for client in $zeoclients
    do
       echo -n $"Stopping $client: "
       output=`${clusterpath}/bin/${client} stop`
       # the return status of the instance is not reliable, we need to parse
        # its output via substring match
        if echo $output | grep -q "stop"; then
            # success
            rm /var/lock/zope/${client}
            log_success_msg "$client stopped successfully"
            echo
            RETVAL=0
        else
            # failed
            log_failure_msg "$client failed to stop or was already stopped"
            echo
            RETVAL=1
        fi
    done
    echo -n $"Stopping $prog: "
    output=`${clusterpath}/bin/zeoserver stop`
    # the return status of the instance is not reliable, we need to parse
    # its output via substring match
    if echo $output | grep -q "stop"; then
            # success
            rm /var/lock/zope/$prog
            log_success_msg "zeoserver stopped successfully"
            echo
            RETVAL=0
    else
            # failed
            log_failure_msg "zeoserver failed to stop or was already stopped"
            echo
            RETVAL=1
    fi
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        return $RETVAL
}

restart() {
   stop
   start
}

case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  status)
        echo "ZEO Server:"
        output=`${clusterpath}/bin/zeoserver status`
        echo $output
        for client in $zeoclients
        do
                echo "Zope Client" $client
                output=`${clusterpath}/bin/${client} status`
                echo $output
        done
        ;;
  restart)
    restart
    ;;
  condrestart)
    [ -e /var/lock/zope/$prog ] && restart
    ;;
  *)
    echo $"Usage: $0 {start|stop|status|restart|condrestart}"
    RETVAL=2
esac

exit $RETVAL
```

Dabei enthält das Skript folgende Optionen:

- `start`
- `stop`
- `status`
- `restart`
- `condrestart`

Anmerkung 1: Da der `effective-user` auf `zope` gesetzt wurde (s.a. Buildout für Produktivserver) sollte der Nutzer `zope` nun selbstverständlich in `/var/lock/zope/` schreiben dürfen.

Anmerkung 2: Gegebenenfalls sollte auch die Environment-Variable für den `PYTHON_EGG_CACHE` in der `deploy.cfg`-Datei festgelegt werden:

```
[instance]
...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
environment-vars =
    PYTHON_EGG_CACHE = /home/zope/.python-eggs
```

init-Prozess

Sofern symbolische Links in `/etc/rc?.d` angelegt sind, wird beim Neustart des Hosts der ZEO-Cluster ebenfalls gestartet werden. Dabei ist `?` eine Zahl zwischen `0` und `6`, die für die unterschiedlichen Runlevel des Systems stehen. Üblicherweise wird Zope in den Runlevel `3`, `4` und `5` gestartet. Hierfür wird nun zunächst in `/etc/init.d` ein symbolischer Link auf unser Skript erzeugt und dann beim Starten dieses Skripts die weiteren symbolischen Links für die genannten Runlevel erzeugt:

```
$ cd /etc/init.d
$ sudo ln -s /srv/myproject/zeo .
$ sudo /etc/init.d/zeo start
```

Und falls die symbolischen Links für die Runlevel wieder entfernt werden sollen, kann dies durch folgenden Aufruf geschehen:

```
$ sudo chkconfig --level 345 zeo off
```

User-crontab

Falls sie nicht die notwendigen Rechte besitzen sollten, um die entsprechenden `init`-Skripte zu schreiben zu können, kann der Cluster beim Neustart auch über einen Eintrag in der User-crontab gestartet werden. Der Eintrag hierfür kann in der `deploy.cfg` angegeben werden:

```
[buildout]
...
parts =
    ...
    reboot

[reboot]
recipe = z3c.recipe.usercrontab
times = @reboot
command = ${buildout:directory}/zeo start
```

Subversion

Soll das `zeo`-Skript unter Versionsverwaltung von Subversion gestellt werden, muss Subversion noch mitgeteilt werden, dass es sich um eine ausführbare Datei handeln soll:

```
$ svn propset svn:executable ON zeo
```

ZODB Replication Services

Mt den ZODB Replication Services (ZRS) können Systemadministratoren ihre ZODB auf zwei oder mehrere Storage-Server replizieren.

Erhöhte Ausfallsicherheit

Dies reduziert die Ausfallzeiten sowohl bei geplanten als auch bei ungeplanten Ausfällen, indem Administratoren:

- unternehmenskritische Daten auf zwei oder mehrere Datenbank-Server replizieren
- des Storage-Cluster in einem Wide Area Network (WAN) verteilen
- die primären und sekundären Server verwalten und überwachen, sodass ggf. ein schneller Failover vom primärem zum sekundärem Storage-Server realisiert werden kann

Backup and Maintenance

Wenn der primäre Storage-Server ausfällt, nimmt der sekundäre Server seinen Platz ein und die ZEO-Clients verbinden sich transparent mit dem neuen ZRS- Server. Die sekundären Server können jedoch auch vom Netz genommen werden, z.B. für die Überprüfung und ggf. Reparatur von `POSKeyError` und `CorruptedError` oder für Upgrades. Wenn dieser sekundäre Server wieder als Service eingebunden wird, synchronisiert er sich automatisch mit dem primären Server. Damit vereinfachen die ZODB Replication Services (ZRS) die routinemäßige Wartung ungemein.

Skalierbarkeit

Die ZODB Replication Services (ZRS) verbessern auch die Skalierbarkeit, da sekundäre Server zusätzliche schreibgeschützte ZEO-Verbindungen unter Beibehaltung ihrer Replikationsfunktionen aufbauen können.

Konzept

Die ZODB Replication Services (ZRS) erhalten einen primären Server für Schreib- und Lesezugriffe und eine beliebige Anzahl von sekundären Servern mit Nur-Lese- Zugriffen. Bei einer Transaktion auf dem primären Server wird diese auf alle verfügbaren sekundären Server repliziert. Wird ein sekundärer kurzzeitig vom Server vom Netz genommen oder ein weiterer sekundärer Server hinzugefügt, so werden diese automatisch auf den aktuellen Stand gebracht.

Installation

Requirements

zc.zrs setzt ZODB 3.9 oder größer voraus.

Buildout

1. In der Buildout-Konfiguration muss `zc.zrs` als zusätzliches Egg angegeben werden.
2. Anschließend kann der primäre Server konfiguriert werden mit:

```
[zeoserver1]
...
zeo-conf-additional =
    <zrs>
        replicate-to 5000
        <filestorage>
            path ${buildout:directory}/var/filestorage/Data.fs
        </filestorage>
    </zrs>
```

replicate-to Adresse des Replikationsservices

Dies kann nur eine Port-Nummer oder ein Hostname mit Portnummer durch einen Doppelpunkt getrennt sein.

3. Konfigurieren eines sekündären Servers:

```
[zeoserver2]
...
zeo-conf-additional =
    <zrs>
        replicate-from primary-host: 5000
        replicate-to 5000
        keep-alive-delay 60
        <filestorage>
            path ${buildout:directory}/var/filestorage/Data.fs
        </filestorage>
    </zrs>
```

replicate-from primary-host Adresse des primären Servers

replicate-to Optionale Angabe.

Wird diese Option genutzt, können andere sekundäre Server von diesem Service replizieren.

keep-alive-delay Optionale Angabe.

In einigen Netzwerkkonfigurationen werden TCP-Verbindungen nach längerer Inaktivität unterbrochen. Um dies zu verhindern, sendet der sekundäre Server regelmäßige `no-operation`-Nachrichten um die Verbindung aufrechtzuerhalten.

plone.recipe.zeoserver

Ab Version 1.2.6 von plone.recipe.zeoserver oder Plone 4.3.2 lässt sich der ZRS einfacher installieren:

```
[zeoserver]
recipe = plone.recipe.zeoserver[zrs]
...
```

replicate-to Angabe von Host und Port für den primären Server.

replicate-from Angabe von Host und Port für den sekundären Server, der die Daten repliziert.

keep-alive-delay In manchen Netzwerkkonfigurationen wird die TCP-Verbindung unterbrochen bei einer längeren Zeit der Inaktivität. Um dies zu verhindern kann der sekundäre Server periodische Nachrichten an den primären Server schicken.

Siehe auch:

- Repository

Profiling

Die Steigerung der Performance kann sehr schwierig sein. Es gibt jedoch eine Reihe von Werkzeugen, mit denen sich die Performance messen und die Flaschenhälse auffinden lassen.

Python Profilers erstellt Zope-Publisher-Profile, die im ZMI unter `http://127.0.0.1:8080/Control_Panel/DebugInfo/manage_profile` angezeigt werden:

Die Installation erfolgt mit:

```
[instance]
...
zope-conf-additional =
    publisher-profile-file ${buildout:directory}/var/instance/profile.dat
environment-vars =
    PROFILE_PUBLISHER 1
```

PTProfiler erstellt Profile für Page Templates in Zope2. Für jeden TAL-Ausdruck wird die Zeit gemessen und eine Tabelle sortiert nach den benötigten Zeiten erstellt.

Um `PTProfiler` zu installieren können Sie einfach folgendes in Ihre `buildout.cfg`-Datei schreiben:

```
eggs =
    ...
    Products.PTProfiler
```

Nachdem Sie das Buildout-Skript aufgerufen und die Instanz neu gestartet haben, können Sie an jeder Stelle Ihrer Instanz ein *PTProfiler Viewer*-Objekt hinzufügen und anschließend auf *Enable* klicken. Nun erhalten Sie eine Liste der Pfade aller aufgerufenen Page Templates und beim Klicken auf einen der Pfade auch die Zeiten für alle Aufrufe.

Call Profiler zeigt den Ablauf von DTML, ZSQL, ZPT, Python-Methoden und -Skripten an, wobei sowohl die absoluten als auch die relativen Zeiten ermittelt werden. Zum Installieren lässt sich in der `buildout.cfg`-Datei im Abschnitt `[productdistros]` als URL folgendes angegeben:

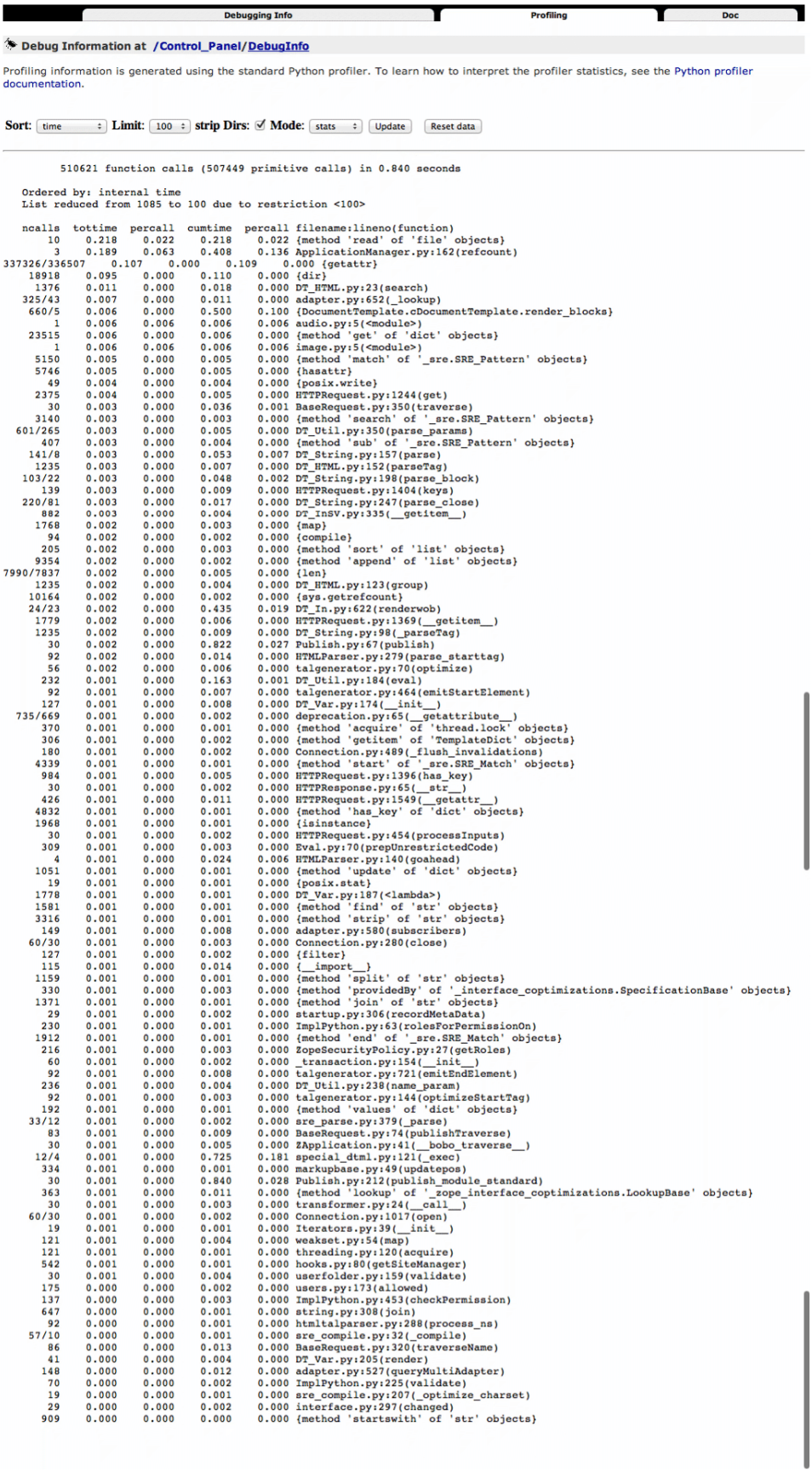
```
http://plone.org/products/callprofiler/releases/1.4-fixed/CallProfiler-1.4-fixed.
↪tar.gz
```

Anschließend kann in `/Control_Panel/CallProfiler` angegeben werden, für welche Aufrufe ein Profil erstellt werden soll.

ZopeProfiler liefert sowohl *Zope object call level*- als auch *Python function call level*-Statistiken, z.B.:

```
      167 function calls (160 primitive calls) in 5.229 CPU seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    2.927    2.927    2.960    2.960 document_view:0(__call__)
     1    0.754    0.754    3.715    3.715 mysite:0(Request)
```

(Fortsetzung auf der nächsten Seite)



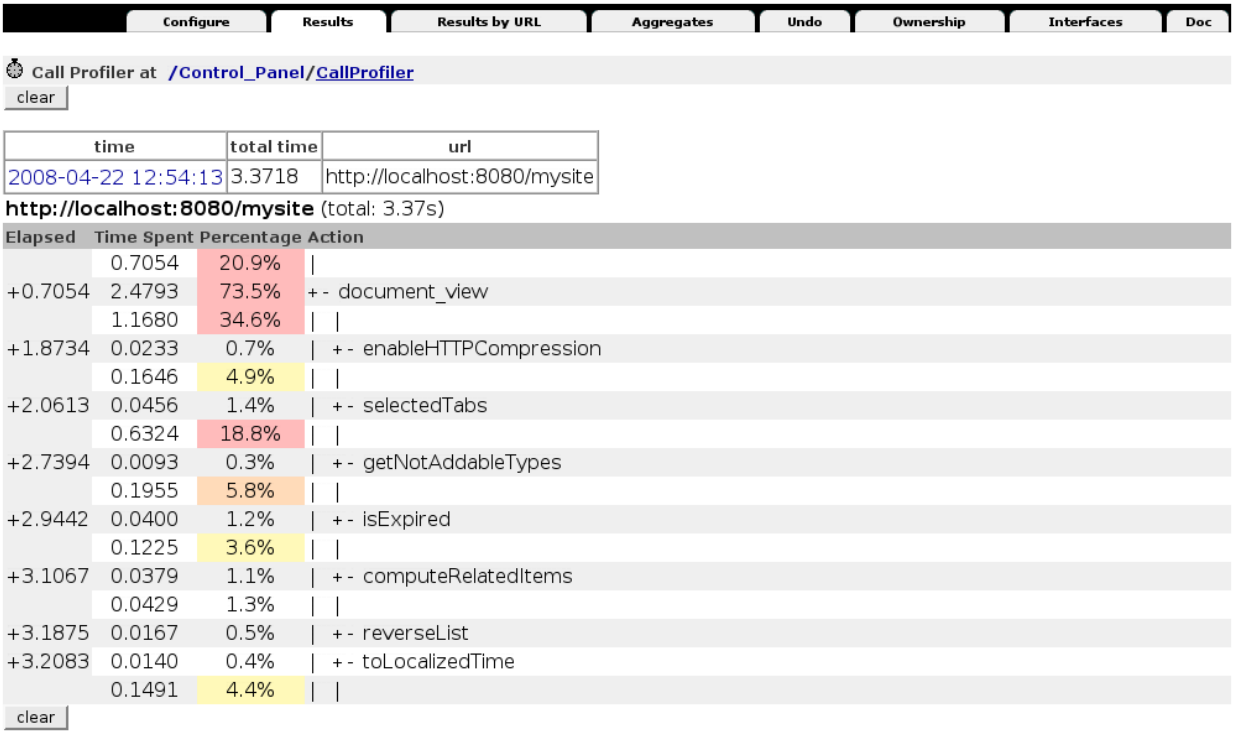
Debugging Info | Profiling | Doc

Debug Information at /Control_Panel/DebugInfo

Profiling information is generated using the standard Python profiler. To learn how to interpret the profiler statistics, see the Python profiler documentation.

Sort: time Limit: 100 strip Dirs: Mode: stats Update Reset data

```
         510621 function calls (507449 primitive calls) in 0.840 seconds

   Ordered by: internal time
   List reduced from 1085 to 100 due to restriction <100>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       10    0.218    0.022    0.218    0.022 {method 'read' of 'file' objects}
        3    0.189    0.063    0.408    0.136 ApplicationManager.py:162(refcount)
337326/336507    0.107    0.000    0.109    0.000 {getattr}
    18918    0.095    0.000    0.110    0.000 {dir}
     1376    0.011    0.000    0.018    0.000 DT_HTML.py:23(search)
   325/43    0.007    0.000    0.011    0.000 adapter.py:652(_lookup)
    660/5    0.006    0.000    0.500    0.100 {DocumentTemplate.cDocumentTemplate.render_blocks}
        1    0.006    0.006    0.006    0.006 audio.py:5(<module>)
    23515    0.006    0.000    0.006    0.000 {method 'get' of 'dict' objects}
        1    0.006    0.006    0.006    0.006 image.py:5(<module>)
     5150    0.005    0.000    0.005    0.000 {method 'match' of '_sre.SRE_Pattern' objects}
     5746    0.005    0.000    0.005    0.000 {hasattr}
       49    0.004    0.000    0.004    0.000 {posix.write}
     2375    0.004    0.000    0.005    0.000 HTTPRequest.py:1244(get)
       30    0.003    0.000    0.036    0.001 BaseRequest.py:350(traverse)
     3140    0.003    0.000    0.003    0.000 {method 'search' of '_sre.SRE_Pattern' objects}
  601/265    0.003    0.000    0.005    0.000 DT_Util.py:350(parse_params)
      407    0.003    0.000    0.004    0.000 {method 'sub' of '_sre.SRE_Pattern' objects}
    141/8    0.003    0.000    0.053    0.007 DT_String.py:157(parse)
     1235    0.003    0.000    0.007    0.000 DT_HTML.py:152(parseTag)
   103/22    0.003    0.000    0.048    0.002 DT_String.py:198(parse_block)
      139    0.003    0.000    0.009    0.000 HTTPRequest.py:1404(keys)
   220/81    0.003    0.000    0.017    0.000 DT_String.py:247(parse_close)
      882    0.003    0.000    0.004    0.000 DT_InSV.py:335(__getitem__)
     1768    0.002    0.000    0.003    0.000 {map}
       94    0.002    0.000    0.002    0.000 {compile}
      205    0.002    0.000    0.003    0.000 {method 'sort' of 'list' objects}
     9354    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
7990/7837    0.002    0.000    0.005    0.000 {len}
     1235    0.002    0.000    0.004    0.000 DT_HTML.py:123(group)
    10164    0.002    0.000    0.002    0.000 {sys.getrefcount}
    24/23    0.002    0.000    0.435    0.019 DT_In.py:622(renderwob)
     1779    0.002    0.000    0.006    0.000 HTTPRequest.py:1369(__getitem__)
     1235    0.002    0.000    0.009    0.000 DT_String.py:98(_parseTag)
       30    0.002    0.000    0.822    0.027 Publish.py:67(publish)
       92    0.002    0.000    0.014    0.000 HTMLParser.py:279(parse_starttag)
       56    0.002    0.000    0.006    0.000 talgenerator.py:70(optimize)
      232    0.001    0.000    0.163    0.001 DT_Util.py:184(eval)
       92    0.001    0.000    0.007    0.000 talgenerator.py:464(emitStartElement)
      127    0.001    0.000    0.008    0.000 DT_Var.py:174(__init__)
  735/669    0.001    0.000    0.002    0.000 deprecation.py:65(__getattribute__)
      370    0.001    0.000    0.001    0.000 {method 'acquire' of 'thread.lock' objects}
      306    0.001    0.000    0.002    0.000 {method 'getitem' of 'TemplateDict' objects}
      180    0.001    0.000    0.002    0.000 Connection.py:489(_flush_invalidations)
     4339    0.001    0.000    0.001    0.000 {method 'start' of '_sre.SRE_Match' objects}
      984    0.001    0.000    0.005    0.000 HTTPRequest.py:1396(has_key)
       30    0.001    0.000    0.002    0.000 HTTPResponse.py:65(__str__)
      426    0.001    0.000    0.011    0.000 HTTPRequest.py:1549(__getattr__)
     4832    0.001    0.000    0.001    0.000 {method 'has_key' of 'dict' objects}
     1968    0.001    0.000    0.001    0.000 {isinstance}
       30    0.001    0.000    0.002    0.000 HTTPRequest.py:454(processInputs)
      309    0.001    0.000    0.003    0.000 Eval.py:70(prepUnrestrictedCode)
        4    0.001    0.000    0.024    0.006 HTMLParser.py:140(goahead)
     1051    0.001    0.000    0.001    0.000 {method 'update' of 'dict' objects}
       19    0.001    0.000    0.001    0.000 {posix.stat}
     1778    0.001    0.000    0.001    0.000 DT_Var.py:187(<lambda>)
     1581    0.001    0.000    0.001    0.000 {method 'find' of 'str' objects}
     3316    0.001    0.000    0.001    0.000 {method 'strip' of 'str' objects}
      149    0.001    0.000    0.008    0.000 adapter.py:580(subscribers)
    60/30    0.001    0.000    0.003    0.000 Connection.py:280(close)
      127    0.001    0.000    0.002    0.000 {filter}
      115    0.001    0.000    0.014    0.000 {__import__}
     1159    0.001    0.000    0.001    0.000 {method 'split' of 'str' objects}
      330    0.001    0.000    0.003    0.000 {method 'providedBy' of '_interface_coptimizations.SpecificationBase' objects}
     1371    0.001    0.000    0.001    0.000 {method 'join' of 'str' objects}
       29    0.001    0.000    0.002    0.000 startup.py:306(recordMetaData)
      230    0.001    0.000    0.001    0.000 ImplPython.py:63(rolesForPermissionOn)
     1912    0.001    0.000    0.001    0.000 {method 'end' of '_sre.SRE_Match' objects}
      216    0.001    0.000    0.003    0.000 ZopeSecurityPolicy.py:27(getRoles)
       60    0.001    0.000    0.002    0.000 _transaction.py:154(__init__)
       92    0.001    0.000    0.008    0.000 talgenerator.py:721(emitEndElement)
      236    0.001    0.000    0.004    0.000 DT_Util.py:238(name_param)
       92    0.001    0.000    0.003    0.000 talgenerator.py:144(optimizeStartTag)
      192    0.001    0.000    0.001    0.000 {method 'values' of 'dict' objects}
    33/12    0.001    0.000    0.002    0.000 sre_parse.py:379(_parse)
       83    0.001    0.000    0.009    0.000 BaseRequest.py:74(publishTraverse)
       30    0.001    0.000    0.005    0.000 ZApplication.py:41(__bobo_traverse__)
     12/4    0.001    0.000    0.725    0.181 special_dtml.py:121(_exec)
      334    0.001    0.000    0.001    0.000 marshalbase.py:49(updatepos)
       30    0.001    0.000    0.840    0.028 Publish.py:212(publish_module_standard)
      363    0.001    0.000    0.011    0.000 {method 'lookup' of '_zope_interface_coptimizations.LookupBase' objects}
       30    0.001    0.000    0.003    0.000 transformer.py:24(__call__)
    60/30    0.001    0.000    0.002    0.000 Connection.py:1017(open)
       19    0.001    0.000    0.001    0.000 Iterators.py:39(__init__)
      121    0.001    0.000    0.004    0.000 weakset.py:54(map)
      121    0.001    0.000    0.001    0.000 threading.py:120(acquire)
      542    0.001    0.000    0.001    0.000 hooks.py:80(getSiteManager)
       30    0.001    0.000    0.004    0.000 userfolder.py:159(validate)
      175    0.000    0.000    0.002    0.000 users.py:173(allowed)
      137    0.000    0.000    0.003    0.000 ImplPython.py:453(checkPermission)
      647    0.000    0.000    0.001    0.000 string.py:308(join)
       92    0.000    0.000    0.001    0.000 htmltalparser.py:288(process_ns)
    57/10    0.000    0.000    0.001    0.000 sre_compile.py:32(_compile)
       86    0.000    0.000    0.013    0.000 BaseRequest.py:320(traverseName)
       41    0.000    0.000    0.004    0.000 DT_Var.py:205(render)
      148    0.000    0.000    0.012    0.000 adapter.py:527(queryMultiAdapter)
       70    0.000    0.000    0.002    0.000 ImplPython.py:225(validate)
       19    0.000    0.000    0.001    0.000 sre_compile.py:207(_optimize_charset)
       29    0.000    0.000    0.002    0.000 interface.py:297(changed)
      909    0.000    0.000    0.000    0.000 {method 'startswith' of 'str' objects}
```



Configure | Results | Results by URL | Aggregates | Undo | Ownership | Interfaces | Doc

Call Profiler at /Control_Panel/CallProfiler

clear

time	total time	url
2008-04-22 12:54:13	3.3718	http://localhost:8080/mysite

http://localhost:8080/mysite (total: 3.37s)

Elapsed	Time Spent	Percentage	Action
	0.7054	20.9%	|
+0.7054	2.4793	73.5%	+- document_view
	1.1680	34.6%	| |
+1.8734	0.0233	0.7%	| +- enableHTTPCompression
	0.1646	4.9%	| |
+2.0613	0.0456	1.4%	| +- selectedTabs
	0.6324	18.8%	| |
+2.7394	0.0093	0.3%	| +- getNotAddableTypes
	0.1955	5.8%	| |
+2.9442	0.0400	1.2%	| +- isExpired
	0.1225	3.6%	| |
+3.1067	0.0379	1.1%	| +- computeRelatedItems
	0.0429	1.3%	| |
+3.1875	0.0167	0.5%	| +- reverseList
+3.2083	0.0140	0.4%	| +- toLocalizedTime
	0.1491	4.4%	| |

clear

(Fortsetzung der vorherigen Seite)

```
      1    0.186    0.186    0.187    0.187 search_icon.gif:0(Request)
      1    0.132    0.132    0.132    0.132 folder_icon.gif:0(Request)
      1    0.080    0.080    0.080    0.080 file_icon.gif:0(Request)
      1    0.076    0.076    0.076    0.076 member-cachekey9666.css:0(Request)
      1    0.071    0.071    0.073    0.073 document_icon.gif:0(Request)
...
```

Installieren lässt sich der ZopeProfiler durch die Angabe der URL im Abschnitt `[productdistros]` der `buildout.cfg`-Datei:

```
http://www.dieter.handshake.de/pyprojects/zope/ZopeProfiler.tgz
```

Eine Erläuterung der Tabelle finden Sie in The Python Profilers.

hotshot Python-Profiler, der für jede einzelne Funktion ein Profil ausgibt. Er kann einfach aufgerufen werden, z.B. mit:

```
./bin/instance test -pvvv --profile -m vs.registration
```

Dabei wird eine Liste der 50 zeitaufwendigsten Funktionen ausgegeben und eine Datei geschrieben wie z.B. `myproject/tests_profile.bMAEin.prof`. Diese Datei lässt sich editieren, z.B. mit:

```
$ python
>>> from hotshot.stats import load
>>> s = load('tests_profile.bMAEin.prof')
s.print_stats('.*some_function.*')
```

profilehooks Decorator für das Profiling einzelner Funktionen:

```
from profilehooks import profile

  @profile
  def my_function(args, etc):
      pass
```

Die Installation erfolgt einfach mit `easy_install profilehooks`.

Mit `pydoc profilehooks` erhalten Sie eine Liste aller verfügbaren Decorator-Optionen.

Weitere Informationen erhalten Sie in der profilehooks-Dokumentation

collective.stats gibt low level ZODB-Statistiken aus.

Installation:

```
[instance]
...
eggs =
    ...
    collective.stats
```

Beim Starten der Instanz im Vordergrung mit `./bin/instance fg` erhalten Sie dann z.B. folgende Ausgabe auf der Konsole:

```
2014-02-17 12:25:30 INFO Zope Ready to handle requests
2014-02-17 12:25:50 INFO collective.stats | 0.0021 0.0014 0.0018 0.0004 0.0000␣
↪0000 0000 0000 | GET:/favicon.ico | t: 0.0000, t_c: 0.0000, t_nc: 0.0000 | RSS:␣
↪116708 - 116744
2014-02-17 12:25:55 INFO collective.stats | 0.1783 0.0021 0.1779 0.0004 0.0000␣
↪0000 0000 0000 | GET:/manage_main | t: 0.0000, t_c: 0.0000, t_nc: 0.0000 | RSS:␣
↪116756 - 116948
```

Die Werte bedeuten dann im Einzelnen:

Header Detail

time Dauer innerhalb des Zope Publisher

t traverse Zeit, zu dem der Zope Publisher

t commit Dauer für `transaction.commit()`

t transchain Dauer für `plone.transformchain.applyTransform`

setstate Dauer in `Connection.setstate`

total Anzahl der *zodb object loads*

total cached Anzahl der Cache loads

modified Anzahl modifizierter Objekte

rss before RAM-Verbrauch vor dem Request

rss after RAM-Verbrauch nach dem Request

Products.LongRequestLogger Sog. *stack traces* lang laufender Requests an eine Zope2-Instanz werden periodisch in eine Log-Datei geschrieben. Die Konfiguration des `Products.LongRequestLogger` erfolgt über Umgebungsvariablen für diese Instanz:

```
[instance]
...
eggs =
    ...
    Products.LongRequestLogger

environment-vars =
    longrequestlogger_file = ${buildout:directory}/var/log/${:_buildout_section_
↪name_}-longrequest.log
    longrequestlogger_timeout = 4
    longrequestlogger_interval = 2
```

longrequestlogger_file Erforderliche Pfadangabe zu der Datei, in die das Log geschrieben werden soll.

longrequestlogger_timeout Die Anzahl in Sekunden als Fließkommazahl, nachdem das Logging beginnen soll.

Der Standardwert ist``2``.

longrequestlogger_interval Die Frequenz, mit der der *stack trace* geschrieben werden soll.

Der Standardwert ist `1`.

HAProxy

- Debugging web application performance with HAProxy
- Debugging web application performance with HAProxy – Part 2

ab Apache HTTP-Server-Benchmarking-Werkzeug, das einfache Performance-Tests erlaubt.

Unter Debian und Ubuntu lässt sich `ab` installieren mit:

```
$ apt-get install apache2-utils
```

Anschließend kann es z.B. mit folgenden Optionen aufgerufen werden:

```
$ ab -n 100 -c 3 http://www.veit-schiele.de/
```

Damit wird 100-mal die Seite angefragt, wobei immer je 3 Anfragen gleichzeitig gestellt werden.

Um die jeweiligen Einstellungen zu testen, empfiehlt es sich, zunächst die einfache Plone-Site, dann mit Cache-Fu und schließlich mit Varnish zu testen.

Mehr über Apache Benchmark erfahren Sie mit:

```
$ ab -h
```

Beachten Sie, dass Apache Benchmark nur die angegebene URL prüft, nicht die gesamte Seite mit Bildern und CSS-Dateien.

TinyLogAnalyzer Mit TinyLogAnalyzer lassen sich die Antowrtzeiten des HTTP-Access-Log auswerten. Hierzu muss zunächst die Log-Datei so konfiguriert werden, dass sie auch die Antwortzeiten protokolliert:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %T/%D"␣
↪combined
```

ergibt z.B. folgende Ausgabe:

```
[31/Oct/2011:13:36:07 +0000] "GET / HTTP/1.1" 200 7918 "" "..." 0/95491
```

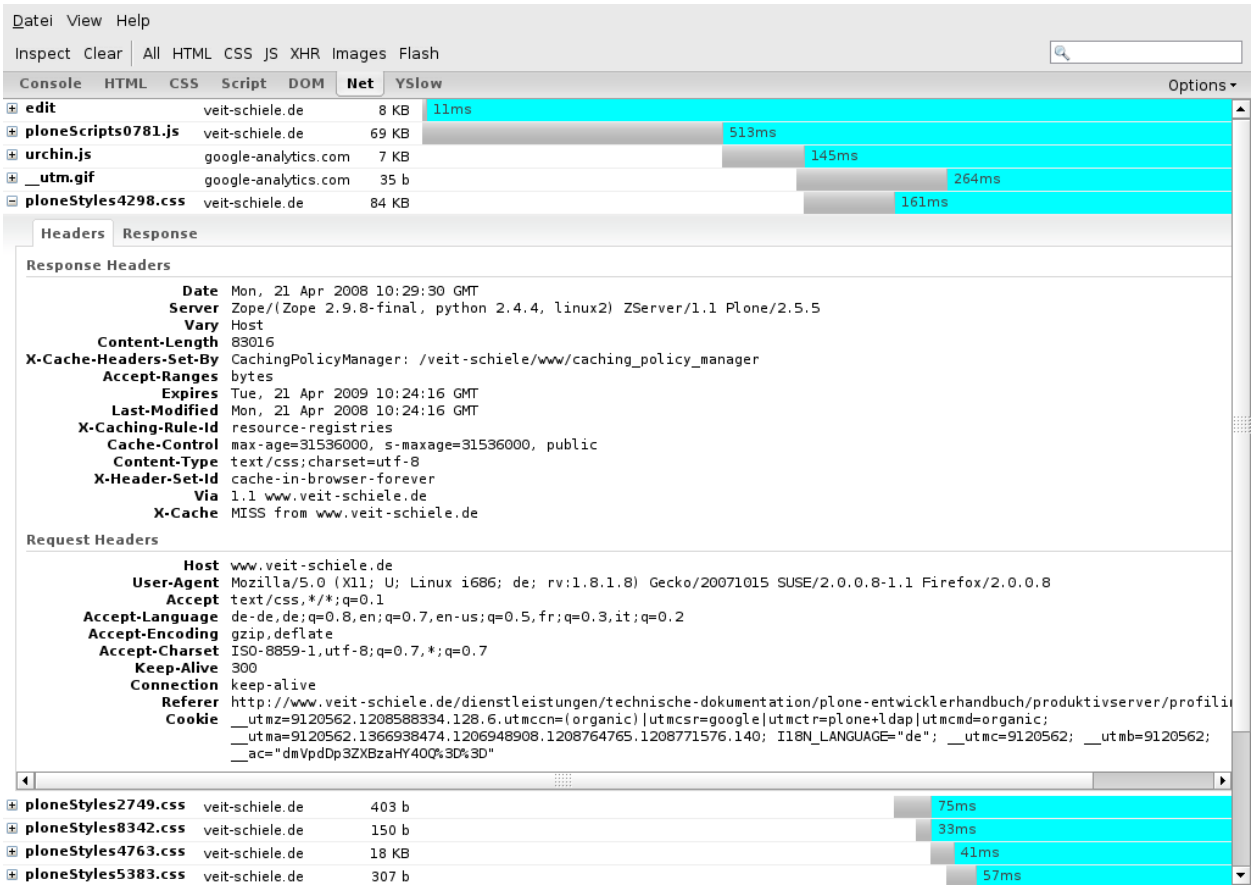
Dabei ist `0/95491` die Zeit in Sekunden und Mikrosekunden, die de Beantwortung des Requests dauerte.

Eine solche Log-Datei kann nun von TinyLogAnalyzer ausgewertet werden.

Apache JMeter Apache JMeter wurde entwickelt um Last und Performance von funktionalen Tests zu messen.

FunkLoad Mit FunkLoad lassen sich ebenfalls Lasttests auf Basis von funktionalen Tests erstellen

Firebug Mit Firebug lässt sich der Traffic zwischen Ihrem Browser und der Website beobachten:



Dabei werden alle Requests zum Darstellen der gesamten Seite analysiert. Zudem kann man sich die *Response Headers* für jedes Objekt anzeigen lassen und so herausfinden, ob, von wem und in welchem Umfang tatsächlich gecached wird.

YSlow analysiert Webseiten und teilt Ihnen mit, warum Ihre Seiten langsam dargestellt werden. YSlow ist ein Firefox Add-on, das in Firebug integriert ist.

GTmetrix Website, die die Ergebnisse der Messungen mit PageSpeed und YSlow anzeigt, ohne dass diese Plugins installiert sein müssten. Darüberhinaus lassen sich auch *Timeline* und *History* anzeigen.

varnishstat Varnishstat erstellt kontinuierlich aktualisierte Statistiken einer laufenden `varnishd`-Instanz, wobei zwischen `Hit` und `MISS` unterschieden wird.

Caching

Plone kann üblicherweise 6 bis 10 Anfragen je Sekunde verarbeiten. Einfache Web-Server, die nicht jedesmal die Seite dynamisch neu erzeugen, können häufig hunderte von Anfragen in der Sekunde verarbeiten.

plone.app.caching

Voraussetzungen

plone.app.caching hat folgende Voraussetzungen:

- plone.caching
  - z3c.caching
- plone.cachepurging
- plone.app.registry
  - plone.registry
- plone.protect

Installation

plone.app.caching lässt sich einfach mit Buildout installieren indem in der Buildout-Konfigurationsdatei folgendes angegeben wird:

```
[buildout]
...
extends =
    ...
    http://good-py.appspot.com/release/plone.app.caching/1.0b2
...
eggs =
    ...
    plone.app.caching
```

Anschließend rufen Sie das Buildout-Skript auf und starten Ihre Instanz neu.

Aktivieren

Gehen Sie nun in das Kontrollfeld *Erweiterungen* der Plone-Konfiguration und aktivieren dort *HTTP caching support*.

Dies aktiviert automatisch auch die *Configuration registry* und *Plone z3c.form support*.

plone.app.caching: Konfiguration

Nach dem Aktivieren finden Sie ein Caching-Kontrollfeld in der Plone-Konfiguration. Dieses Kontrollfeld enthält die folgenden Reiter:

Change Settings

Global settings

Enable caching Falls diese Option nicht aktiviert ist, wird nicht gecacht.

Enable GZip compression Ist diese Option aktiv, werden die Inhalte momprimiert bevor sie zum Browser gesendet werden.

Caching proxies

Enable purging Ist diese Option aktiviert, schickt Plone ggf. HTTP-PURGE-Anfragen an den Caching-Proxy. Dieser muss PURGE erlauben, damit der Cache ggf. geleert werden kann.

Caching proxies Geben Sie hier den Domain-Namen jedes Caching Proxy an, einen je Zeile.

Content types to purge Ist diese Option aktiviert, kann Plone die Views von Artikeln löschen wenn sie verändert oder gelöscht wurden.

Virtual host rewriting takes place front of the caching proxy Aktivieren Sie diese Option, wenn Sie virtuelle Hosts mit Rewriting vor dem Caching Proxy verwenden.

Externally facing domains Falls Sie Rewriting mit virtuellen Hosts konfiguriert haben und Ihre Site über mehrere Domains erreichbar ist (z.B. `http://veit-schiele.de` und `http://www.veit-schiele.de`), sollten alle verfügbaren Domains eingetragen werden, eine je Zeile. Dies gewährleistet, dass die PURGE-Anfragen für alle Domains gesendet werden.

In-memory cache

Maximum entries in the cache Wie viele Artikel sollen im Cache gespeichert werden?

Maximum age of entries in the cache Die maximale Zeit in Sekunden, die ein Artikel im Cache gespeichert wird bevor es gelöscht wird.

Cleanup interval Zeit in Sekunden, bevor der Cache gereinigt werden soll. Geringere WErte verringern den Speicherverbrauch, erhöhen jedoch die Last.

Caching operations

Ruleset mappings

Regelsätze zur Verknüpfung von Views, PageTemplates und Ressourcen mit Caching-Operatoren. `plone.app.caching` enthält sechs verschiedene Regelsätze, die zusammen mit den Beschreibungen im Kontrollfeld ausgewählt werden können:

Content Feed (`plone.content.feed`) Regelsatz für Feeds, z.B. RSS oder ATOM.

Content files and images (`plone.content.file`) Regelsatz für Dateien und Bilder im Inhaltsbereich.

Content folder view (`plone.content.folderView`) Öffentliche Ansicht eines Artikels, der andere Artikel enthalten kann.

Content item view (`plone.content.itemView`) Öffentliche Ansicht eines Artikels, der keine anderen Artikel enthalten kann.

File and image resources (`plone.resource`) Bilder und Dateien, die enweder über das *Portal Skin Tool* ausgeliefert werden oder in registrierten Verzeichnissen im Dateisystem bereitgestellt werden.

Stable file and image resources (`plone.stableResource`) Über die *Resource Registries* verwaltete Ressourcen wie CSS-, Javascript- und KSS-Dateien. Dies sind Dateien, die dauerhaft gespeichert werden können, da sich die URL ändert sobald sich das Objekt selbst ändert.

Legacy template mappings

Für jeden dieser Regelsätze können Sie einen der Operatoren auswählen, der mit `plone.app.caching` geliefert wird:

Strong caching (`plone.app.caching.strongCaching`) cacht im Browser und Proxy für 24 Stunden und setzt folgende Header:

```
Last-Modified: <last-modified-date>
Cache-Control: max-age=<seconds>, proxy-revalidate, public
```

Achtung: Dies sollte nur für *Stable Ressources** verwendet werden, die sich nie ändern ohne dass sich auch ihrre URL ändert.

Moderate caching (`plone.app.caching.moderateCaching`) cacht im Browser, aber läuft sofort ab und cacht im Proxy für 24 Stunden.

Der Operator wird verwendet für die Regelsätze `plone.content.feed` und `plone.content.file`. Für Feeds wird dann z.B. folgender Header ausgeliefert:

```
ETag: <etag-value>
Cache-Control: max-age=0, s-maxage=<seconds>, must-revalidate
```

Und für Dateien wird folgender Header gesetzt:

```
Last-Modified: <last-modified-date>
Cache-Control: max-age=0, s-maxage=<seconds>, must-revalidate
```

Achtung: Wird ein Proxy verwendet, dessen Cache nicht zuverlässig geleert werden kann, können veraltete Inhalte ausgeliefert werden.

Weak caching (`plone.app.caching.weakCaching`) cacht im Browser aber läuft sofort ab und gibt anschließend den HTTP-Status-Code `304 Not Modified` aus.

Im Caching-Profil `with-caching-proxy` wird der operator verwendet für die Regelsätze `plone.content.itemView` und `plone.content.folderView`.

No caching (`plone.app.caching.noCaching`) die Antwort verfällt sofort im Browser und verhindert die Validierung.

Chain (`plone.caching.operations.chain`) erlaubt die Verwendung mehrerer Operatoren zusammen.

Standard-Mapping von `plone.app.caching`:

Regelsatz	Profile		
	without-caching-proxy	with-caching-proxy	with-caching-proxy-splitviews
Content Feed	weakCaching	moderateCaching	moderateCaching
Content files and images	weakCaching	moderateCaching	moderateCaching
Content folder view	weakCaching	weakCaching	moderateCaching
Content item view	weakCaching	weakCaching	moderateCaching
File and image resources	strongCaching	strongCaching	strongCaching
Stable file and image resources	strongCaching	strongCaching	strongCaching

Detailed settings

Hier können Sie Parameter für individuelle Caching-Operatoren angeben:

Maximum age (`maxage`) Zeit in Sekunden, die die Antwort im Browser oder Caching-Proxy gespeichert werden soll.

Fügt der Antwort einen `Cache-Control: max-age=<value>`-Header und einen passenden `Expires`-Header hinzu.

Shared maximum age (`smaxage`) Zeit in Sekunden, die die Antwort im Caching-Proxy gespeichert wird.

Fügt der Antwort einen `Cache-Control: s-maxage=<value>`-Header hinzu.

ETags (`etags`) Eine Liste der ETag-Komponenten, die mit dem ETag-Header ausgegeben werden sollen.

Darüberhinaus wird eine `304 Not Modified`-Antwort generiert für Antworten auf `If-None-Match`-Anfragen.

Last-modified validation (`lastModified`) Fügt der Antwort einen `Last-Modified`-Header hinzu und `304 Not Modified`-Antworten auf `If-Modified-Since`-Anfragen.

RAM cache (`ramCache`) cacht im Zope-RAM-Cache. Ist die URL nicht eindeutig, können entweder ETags oder Last-Modified der Liste der Parameter hinzugefügt werden um einen Unique Cache Key zu erzeugen.

Vary (`vary`) Namen der HTTP-Headers in der Anfrage einer URL, die der Caching Proxy für das Ausliefern einer gecachten Antwort benötigt.

Anonymous only (`anonOnly`) Wird der Wert auf `True` gesetzt, so erhalten angemeldete Nutzer immer die aktuellen Inhalte.

Dies funktioniert am besten zusammen mit dem *moderate caching*-Operator.

Beachten Sie bitte, dass im Caching Proxy der Vary-Header für `X-Anonymous` gesetzt ist.

Request variables that prevent caching (`cacheStopRequestVariables`) Eine Liste von Variablen in der Anfrage (einschließlich Cookies), die das Caching verhindern sollen.

Import settings

Hier können vordefinierte Profile mit Caching-Einstellungen importiert werden.

`plone.app.caching` kommt mit drei Standard-Caching-Profilen:

- Ohne Caching Proxy

  Diese Einstellungen sind hilfreich, wenn kein Caching Proxy verwendet wird.

- Mit Caching Proxy

  Diese Einstellungen sind hilfreich wenn ein Caching Proxy wie Squid oder Varnish verwendet wird. Dieses Profil unterscheidet sich im wesentlichen dadurch vom Profil *Ohne Caching Proxy*, dass Dateien und Bilder im Proxy Cache gespeichert werden können.

- Mit Caching Proxy (und Split-View-Caching)

  Ein Beispiel für ein Profil, das unterschiedliche Ansichten bereitstellt.

Purge Caching-Proxy

Hier können manuell Inhalte des Caching-Proxy gelöscht werden.

Dieser Reiter wird nur angezeigt, wenn Sie in *Change Settings* Purging erlaubt haben.

RAM-Cache

Hier können Sie Statistiken zu Purging und RAM-Cache betrachten.

plone.app.caching: Eigene Profile erstellen

Zunächst wird ein GenericSetup-Profil registriert für das `ICacheProfiles`-Marker-Interface registriert:

```
<genericsetup:registerProfile
    name="with-caching-proxy"
    title="With caching proxy"
    description="Settings useful for setups with a caching proxy such as Squid or␣
↪Varnish"
    directory="profiles/with-caching-proxy"
    provides="Products.GenericSetup.interfaces.EXTENSION"
    for="plone.app.caching.interfaces.ICacheProfiles"
    />
```

Dies bewirkt zugleich, dass dieses Profil nicht als Profil im Erweiterungen-Kontrollfeld von Plone ausgewählt werden kann.

Das Verzeichnis `profiles/with-caching-proxy` enthält eine `registry.xml`-Datei mit Einstellungen für die Regelsätze zum Verknüpfen von Objekten mit Caching-Operationen, z.B.:

```
<record name="plone.caching.interfaces.ICacheSettings.operationMapping">
    <value purge="False">
        <element key="plone.resource">plone.app.caching.strongCaching</element>
        <element key="plone.stableResource">plone.app.caching.strongCaching</element>
        <element key="plone.content.itemView">plone.app.caching.weakCaching</element>
        <element key="plone.content.feed">plone.app.caching.moderateCaching</element>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        <element key="plone.content.folderView">plone.app.caching.weakCaching</
↪element>
        <element key="plone.content.file">plone.app.caching.moderateCaching</element>
    </value>
</record>
```

Um z.B. RAM-Caching für die *weak caching*-Operation von Ressourcen zu erlauben, wird der Regelsatz `plone.content.itemView` verwendet:

```
<record name="plone.app.caching.weakCaching.plone.content.itemView.ramCache">
    <field ref="plone.app.caching.weakCaching.ramCache" />
    <value>True</value>
</record>
```

CacheFu

CacheFu ist eine Sammlung von Produkten, die die Darstellung von Seiten einer Plone-Site deutlich beschleunigen kann, wobei auch Caching-Proxies wie Squid und Varnish unterstützt werden.

CacheFu lässt sich einfach mit Buildout installieren indem unter `eggs` im `[buildout]`-Abschnitt `Products.CacheSetup` angegeben wird.

Nachdem `./bin/buildout` aufgerufen und die Instanz neu gestartet wurde, lässt sich *CacheSetup* in der Website-Konfiguration unter Zusatzprodukte für die Plone-Site installieren. Anschließend finden Sie in der Konfiguration von Zusatzprodukten *Cache Configuration*.

- Im *Main*-Reiter kann CacheFu eingeschaltet und die verwendete Cache Policy angegeben werden. Außerdem kann zwischen verschiedenen Server-Konfigurationen gewählt werden.
- Im *Policy*-Reiter kann zwischen verschiedenen Policies gewählt oder neue erstellt werden.
- Im *Rules*-Reiter können Regeln angegeben werden, wobei zwischen drei verschiedenen Arten unterschieden wird:

  Content Cache Rule Eine Regel für Artikeltypen.

  PolicyHTTPCacheManager Cache Rule Regel für Objekte, die mit einem PolicyHTTPCachingManager assoziiert sind.

  Template Cache Rule Regel für Page Templates.

  Dabei wird immer die erste passende Regel verwendet.
- Im *Headers*-Reiter können HTTP-Header-Angaben für eine spezifische Policy angegeben werden.
  - Mit `max_age` wird der *Expires Header* gesetzt, der dem Browser mitteilt, ob die Seite ohne Revalidierung erneut geladen werden darf. Wenn `max_age` in einem der Header Sets von CacheSetup auf `0` gesetzt wird, werden in der Ausgabe der Response Headers 10 Jahre abgezogen, um nicht-synchronisierte Clients abzufangen.
  - `s-maxage` gibt die Zeit an, die eine Seite im Proxy-Cache gehalten werden darf.

CacheFu für `vs.registration`-Artikeltypen

Um die beiden in `vs.registration` definierten Artikeltypen für CacheFu zu konfigurieren, gehen Sie im *Rules*-Reiter zunächst zur *Content*-Cache-Regel und fügen dort *Registrant* zu den *Content Types* hinzu. Damit werden die Inhalte für nicht-angemeldete Nutzer bis zu einer Stunde im Proxy gespeichert.

Für angemeldete Nutzer werden die Inhalte mit einem *ETag* versehen, das verschiedenen Angaben wie Member ID und Modifikationsdatum enthält. *ETags* werden an den Browser gesendet, damit dieser entscheiden kann, ob eine Seite aus dem Cache geladen werden darf oder vom Server neu angefordert wird. Dabei nutzen verschiedene Nutzer jedoch nicht denselben Cache, damit auch das Caching personalisierter Seiten möglich ist. Der *ETag* hat normalerweise eine Laufzeit von einer Stunde (3600 Sekunden), danach wird das Objekt erneut abgerufen.

Schließlich ist noch *Registration* in die *Container*-Cache-Regel einzutragen.

Übernahme der Cache-Settings in unser Policy-Produkt

Die Cache-Settings lassen sich exportieren und in andere Plone-Sites importieren indem im *Import*-Reiter des *Generic Setup Tool* das *CacheSetup*-Profil ausgewählt wird und anschließend im *Export*-Reiter die Einstellungen des *CacheFu Settings Tool* exportiert werden. Die `cachesettings.xml`-Datei kann dann in das Standardprofil von `vs.policy` übernommen werden.

Siehe auch:

- Initial setup and background
- Cachefu strategy
- Debugging cache settings

Varnish

Varnish ist ein sog. Caching Reverse Proxy, d.h. er sitzt unmittelbar nach dem Web-Server und bildet einen Zwischenspeicher für ausgehende Inhalte.

Einführung

Diese Seite beschreibt, wie der Varnish caching Proxy mit Plone verwendet werden kann.

Installation

Zur Installation von Varnish empfehlen wir denPaket-Manager der jeweiligen Distribution.

- Für Debian/Ubuntu:

```
$ sudo apt-get install varnish
```

- Für CentOS/Fedora:

```
$ yum install varnish
```

- Für Mac OS X:

```
$ sudo port install varnish
```

oder:

```
$ brew install varnish
```

Administrieren

zum Überprüfen einer Varnish-Instanz steht Ihnen `varnishadm` zur Verfügung:

```
# varnishadm -T localhost:6082 -S /etc/varnish/secret
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,4.4.0-92-generic,x86_64,-junix,-smalloc,-smalloc,-hcritbit
varnish-4.1.1 revision 66bb824

Type 'help' for command list.
Type 'quit' to close CLI session.

varnish>
```

Beenden der Konsole

```
quit
```

Cache löschen

Cache vollständig löschen:

```
# varnishadm "ban req.url ~".
```

Alle `.jpg`-Dateien aus dem Cache löschen:

```
# varnishadm "ban req.url ~ .jpg"
```

Konfiguration

Eine neue `*.vcl`-Datei kann geladen werden mit:

```
# vcl.load <name> <file>
```

also z.B. mit:

```
# vcl.load ploneconf /etc/varnish/plone.vcl
```

Anschließend kann sie aktiviert werden mit:

```
# vcl.use ploneconf
```

Anschließend kann Varnish mit Buildout konfiguriert werden. Hierzu tragen Sie folgendes in Ihre `deploy.cfg`-Datei ein:

```
[buildout]
parts =
...
varnish-config
...
[varnish-config]
recipe = collective.recipe.template
input = templates/plone.vcl.in
output = ${buildout:directory}/etc/plone.vcl
backend-host = 127.0.0.1
backend-port = 8010
```

Eine examplarische Varnish-Konfigurationsdatei finden Sie hier: plone.vcl.in. In ihr wird das Backend spezifiziert, das auf `localhost` an Port `8080` läuft und erlaubt Anfragen via HTTP-Basic Authentication oder Cookie-basierte Authentifizierung.

Weitere Informationen zur Varnish-Konfiguration erhalten Sie in Varnish Configuration Language - VCL.

Log-Dateien

Um einen Einträge in den Log-Dateien in Echtzeit zu sehen, können Sie folgendes eingeben:

```
# varnishlog
```

Statistiken

Um sich in Echtzeit die Varnish-Statistik anzuzeigen ähnlich `top`, rufen Sie `varnishstat` auf:

```
# varnishstat
Uptime mgt:     1+23:43:52
Hitrate n:       10       46       46
Uptime child:   1+23:43:53                                                  ␣
↪avg(n):   0.0000   0.0000   0.0000
    NAME                              CURRENT        CHANGE       AVERAGE        AVG_
↪10       AVG_100      AVG_1000
MAIN.uptime                         1+23:43:53
MAIN.sess_conn                           53834          0.00           .            0.
↪07          0.07          0.07
MAIN.client_req                          53834          0.00           .            0.
↪07          0.07          0.07
MAIN.cache_hit                           14119          0.00           .            0.
↪00          0.00          0.00
MAIN.cache_miss                          39568          0.00           .            0.
↪07          0.07          0.07
MAIN.backend_reuse                       38602          0.00           .            0.
↪07          0.07          0.07
...
```

Üblicherweise schreibt Varnish keine Log-Datei sondern hält die Informationen nur im Arbeitsspeicher. Wenn Apache-ähnliche Protokolle aus Varnish geschrieben werden sollen, kann dies mit `varnishncsa` geschehen.

Lastverteilung

Mit der Verteilung der Last auf verschiedene Applikationsserver können die angefragten Objekte schneller ausgeliefert werden.Das `vmod_directors`-Modul ermöglicht diese Lastverteilung auf verschiedene Weisen:

`round-robin` (Rundlauf-Verfahren) greift nacheinander auf die einzelnen Instanzen zu.

`fallback` versucht jede Instanz aus und wählt die erste, die antwortet.

`hash` wählt die Instanz durch Berechnen des Hash eines Strings.

Dies wird häufig verwendet mit ``client.ip``oder einem Session-Cookie, um sog. *sticky sessions* zu bekommen.

`random` verteilt die Last über die Instanzen mit einer gewichteten zufälligen Wahrscheinlichkeitsverteilung.

Das folgende Beispiel zeigt, wie Sie die Round-Robin-Lastverteilung von zwei Plone-Instanzen konfigurieren können:

```
backend instance1 {
    .host = "127.0.0.1";
    .port = "8081";
    .connect_timeout = 0.4s;
    .first_byte_timeout = 300s;
    .between_bytes_timeout  = 60s;
    .probe = {
        .url = "/";
        .timeout = 5s;
        .interval = 15s;
        .window = 10;
        .threshold = 8;
    }
}

backend instance2 {
    .host = "127.0.0.1";
    .port = "8082";
    .connect_timeout = 0.4s;
    .first_byte_timeout = 300s;
    .between_bytes_timeout  = 60s;
    .probe = {
        .url = "/";
        .timeout = 5s;
        .interval = 15s;
        .window = 10;
        .threshold = 8;
    }
}

...

import directors;

sub vcl_init {
    new vdir = directors.fallback();
    vdir.add_backend(instance1);
    vdir.add_backend(instance1);
}

...

import std;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
sub vcl_recv {

    set req.backend_hint = vdir.backend();
    ...
    if (! std.healthy(req.backend_hint))  {
        set req.backend_hint = sorryserver();
        return(pass);
    }
    ...
}
```

probe gibt in unserem Fall an, dass Varnish das `/`-Objekt alle 5 Sekunden aufruft. Falls die Antwort länger als eine Sekunde ausbleibt, nimmt Varnish an, dass das Backend nicht erreichbar ist. Umgekehrt nimmt Varnish an, dass das Backend erreichbar ist wenn drei der letzten fünf Verbindungsversuche erfolgreich waren. Weitere Informationen hierzu erhalten sie in backend health polling.

Bemerkung: Allgemeine Informationen zur Lastverteilung mit Varnish erhalten Sie unter Backend servers und vmod_directors.

Migration zu Varnish 4.x

- Plone Documentation: Varnish 4.x
- What's new in Varnish 4.0
- github.com/fgsch/varnish3to4

Memoize

Decorators zum Caching der Werte von Funktionen und Methoden.

Der generische `memoize`-Decorator verwendet das `GenericCache`-Modul als Storage. Üblicherweise enthält es maximal tausend Objekte, die maximal eine Stunde gecached werden.

`view` und `instance`

`view` und `instance` sind zwei Cache-Decorators, die festgelegte cache keys und storages haben. Hier ein Beispiel für den `instance`-Decorator:

```
from plone.memoize import instance

class MyClass(object):

   @instance.memoize
   def some_function(self, arg1, arg2):
       ...
```

Wird die Funktion `some_function()` das erste Mal aufgerufen, wird der zurückgegebene Wert gespeichert. Anschließend wird bei Aufrufen mit den gleichen Argumenten die gecachte Version ohne erneute Berechnung ausgeliefert.

Mit dem `view`-Decorator wird für denselben Zope3-View im selben Kontext gecached, z.B. mit:

```
from plone.memoize import view

class MyView(BrowserView):

    @view.memoize
    def some_function(self, arg1, arg2):
        ...
```

Schließlich kann mit `@view.memoize_contextless` das Caching des Views unabhängig vom Kontext veranlasst werden. Dabei muss die Anfrage jedoch zwingend `zope.annotation` verwenden.

Marshalling von Schlüsseln und Parametern

Das Marshallers-Modul bietet mehrere Marshaller. Für Beispiele schauen Sie sich am besten die Docstrings in marshallers.py an.

`volatile`-Modul

Das `volatile`-Modul definiert einen Decorator, der uns Angaben darüber erlaubt, wie der cache key berechnet wird und wo er gespeichert wird.

Hier ein einfaches Beispiel für das Caching eines Five Views mit `volatile`- Caching im `ram`-Modul:

```
from Products.Five import BrowserView
from plone.memoize import ram

def _render_cachekey(method, self, brain):
    return (brain.getPath(), brain.modified)

class View(BrowserView):
    @ram.cache(_render_cachekey)
    def render(self, brain):
        obj = brain.getObject()
        view = obj.restrictedTraverse('@@obj-view')
        return view.render()
```

Die Ergebnisse der `render`-Methode werden über Anfragen hinweg und unabhängig vom Nutzer gecached. Der Cache wird den Angaben in `_render_cachekey` entsprechend aktualisiert sobald sich das Änderungsdatum oder der Pfad des Brains ändern.

Sollen die Werte der Funktion nur für maximal eine Stunde gespeichert werden, kann derselbe Decorator verwendet werden:

```
from time import time
...
class View(BrowserView):
    @ram.cache(lambda *args: time() // (60 * 60))
    ...
```

Weitere Beispiele und Erläuterungen des `volatile`-Decorator finden Sie in volatile.py.

Unterstützung für memcached

memcached ist ein *distributed memory caching system*, das ebenfalls von `plone.memoize` unterstützt wird. Hier ein einfaches Beispiel, wie ein solches *Utility* in der `caching.py` definiert werden kann:

```
from zope.interface import directlyProvides
import zope.thread
from plone.memoize.interfaces import ICacheChooser
from plone.memoize.ram import MemcacheAdapter
import os
import memcache

thread_local = zope.thread.local()

def choose_cache(fun_name):
    global servers

    client=getattr(thread_local, "client", None)
    if client is None:
        servers=os.environ.get(
            "MEMCACHE_SERVER", "127.0.0.1:11211").split(",")
        client=thread_local.client=memcache.Client(servers, debug=0)

    return MemcacheAdapter(client)

directlyProvides(choose_cache, ICacheChooser)
```

Anschließend wird die bestehende Konfiguration überschrieben, sodass unser neues Utility für das `ICacheChooser`-Interface verwendet wird. Dies wird in der `overrides.zcml`-Datei angegeben:

```
<configure
        xmlns="http://namespaces.zope.org/zope">

    <utility
        component=".caching.choose_cache"
        provides="plone.memoize.interfaces.ICacheChooser"
        />

</configure>
```

Weitere Informationen zu `memcached` erhaltet Ihr in Python + Memcached: Efficient Caching in Distributed Applications.

archetypes.schematuning

archetypes.schematuning bietet Caching für die Archetypes-Schemata. archetypes.schematuning verwendet plone.memoize um die Archetypes-Schema-Methoden des `BaseObject` zu cachen. Üblicherweise werden diese faktorisiert und für jede Verbindung entsprechend modifiziert. So werden in einer normalen Plone-Site beim Zugriff auf ein ATDocument-Schema durchschnittlich 80 Requests benötigt. Mit `archetypes.schematuning` können solche Zugriffe durchschnittlich um das 18-fache beschleunigt werden.

Für Anwendungen, die auf Archetypes aufsetzen und das Schema dynamisch ändern, steht mit `invalidateSchema` eine Methode zur Verfügung, die das alte Schema aus dem Cache löscht. Hierzu müssen diese Anwendungen jedoch entsprechend angepasst werden.

Apache

Module

Damit der Apache-Webserver Anfragen an Zope weiterleiten kann, muss das `mod_rewrite` Modul mit seinen Abhängigkeiten in Apache's `httpd.conf` angegeben werden:

```
LoadModule alias_module    /usr/lib/apache2/modules/mod_alias.so
LoadModule proxy_module    /usr/lib/apache2/modules/mod_proxy.so
LoadModule proxy_http      /usr/lib/apache2/modules/mod_proxy_http.so
LoadModule rewrite_module  /usr/lib/apache2/modules/mod_rewrite.so
```

Bei Debian- und Ubuntu-Distributionen kann die Konfiguration der Module vereinfacht werden mit `a2enmod`. Die oben angegebenen Module lassen sich dann einfach aktivieren mit:

```
$ a2enmod alias proxy proxy_http rewrite
```

In anderen Distributionen werden die Module meist schon verwendet oder die einzelnen Zeilen müssen nur noch auskommentiert werden.

Virtual Hosts

Anschließend können Sie in `httpd.conf` oder einer eingebundenen Datei einen Virtual Host für die Domain `www.veit-schiele.de` angeben:

```
NameVirtualHost 83.223.91.163:80
<VirtualHost 83.223.91.163:80>
    ServerName    www.veit-schiele.de
    RewriteEngine on
    RewriteRule   ^/(.*) http://83.223.91.163:8082/VirtualHostBase/http/%{SERVER_NAME}
↪:80/vs/VirtualHostRoot/$1 [P]
</VirtualHost>
```

Dies definiert den *Virtual Host* für die Domain `www.veit-schiele.de` wenn Anfragen am Port `80` hereinkommen. Die IP-Adresse `83.223.91.163` sollte derjenigen in der `Listen`-Anweisung entsprechen:

```
Listen 83.223.91.163:80
```

Bei Debian- und Ubuntu-Distributionen kann die Erstellung von Virtual Hosts vereinfacht werden mit `a2ensite`. Der oben angegebene Virtual Host lässt sich dann einfach aktivieren mit:

```
# a2ensite www.veit-schiele.de
Site www.veit-schiele.de installed; run /etc/init.d/apache2 reload to enable.
```

Schließlich wird unter Verwendung der RewriteEngine eine RewriteRule definiert, die für alle dem regulären Ausdruck `^(.*)` entsprechenden URLs weiterleitet:

1. `http://83.223.91.163:8082` verweist auf den Zope-Server, auf den weitergeleitet wird;
2. `VirtualHostBase` informiert das VirtualHostMonster über Protokoll und Host auf den umgeschrieben werden soll. In diesem Fall auf `http` und `veit-schiele.de:80`.
3. Als nächstes wird der Pfad auf das Objekt angegeben, das die *Site Root* sein soll, also unsere Plone-Site. `VirtualHostRoot` beendet die Pfadangabe
4. Mit `$1` wird Apache nun mitgeteilt, dass alle, dem regulären Ausdruck entsprechenden Teile der URL hier angehängt werden sollen.

5. `L` weist Apache an, wenn diese Regel zutreffend war, nicht nach weiteren Regeln zu suchen und `P` aktiviert das `mod_proxy`-Modul, das das URL-Mapping übernimmt.

Anschließend kann Apache mit `apachectl graceful` neu gestartet werden und die Plone-Site sollte dann unter `http://www.veit-schiele.de` erreichbar sein.

Anmerkung: Das root-Verzeichnis des ZMI ist nicht über den virtuellen Host erreichbar. hierzu muss weiterhin `83.223.91.163:8082` aufgerufen werden.

Verschlüsselte Verbindungen

Üblicherweise liefern wir die gesamte Site `https`-verschlüsselt aus. Dabei wird beim Zugriff auf `http://www.veit-schiele.de` weitergeleitet mit:

```
Redirect permanent / https://www.veit-schiele.de/
```

Für `https://www.veit-schiele.de` benötigt der Apache dann das SSL-Modul:

```
LoadModule ssl_module      /usr/lib/apache2/modules/mod_ssl.so
```

Anschließend können Sie einen weiteren Virtual Host für die Domain `www.veit-schiele.de` am SSL-Port `443` angeben:

```
NameVirtualHost 83.223.91.163:443
<VirtualHost 83.223.91.163:443>
    ServerName            www.veit-schiele.de
    SSLEngine             on
    SSLCertificateFile    /etc/apache2/ssl.crt/server.crt
    SSLCertificateKeyFile /etc/apache2/ssl.key/server.key
</VirtualHost>
```

Schließlich muss noch die `Listen`-Anweisung für Anfragen am SSL-Port `443` eingetragen werden.

Login via SSL

Falls z.B. aus Performance-Gründen die Kommunikation der anonymen Nutzer nicht verschlüsselt werden soll, kann bei der Anmeldung auf einen anderen `VirtualHost` mit SSL-Verschlüsselung weitergeleitet werden. Umgekehrt sollen die Nutzer bei der Abmeldung wieder unverschlüsselt auf die Site zugreifen können. Entsprechend kommt für den VirtualHost an Port `80` folgende Rewrite-Regel hinzu:

```
# Rewrites the came_from in the URL for https
RewriteCond %{QUERY_STRING} came_from=http(.*)
RewriteRule ^/(.*)login_form$ https://edit.veit-schiele.de/$1login_form?came_
↪from=https%1 [NE,L]
```

Beachten Sie, dass `?came_from=` nicht direkt in einer `RewriteRule` angegeben werden kann und daher der `QUERY_STRING` zunächst in der `RewriteCond` ausgelesen wird.

Wird `login_form` direkt oder auf einem ungültigen Template (z.B. `logged_out`) aufgerufen, wird folgende zusätzliche Regel benötigt:

```
# Switches to https when hit login_form or login_success
RewriteRule ^/login_(.*) https://edit.veit-schiele.de/login_$1 [NE,L]
```

Schließlich können in der Konfigurationsdatei des VirtualHost an Port `443` noch folgende RewriteRules angegeben werden:

```
# Switches to http upon logout
RewriteRule ^/(.*)logged_out http://www.veit-schiele.de/$1logged_out [L,P]

# Keeps on https until log out
RewriteRule ^/(.*) http://83.223.91.163:8080/VirtualHostBase/https/%edit.veit-schiele.
↪de:443/vs/VirtualHostRoot/$1 [L,P]
```

Amerkung 1: Die Anleitung verweist bewusst auf einen anderen `ServerName`, da die Browser in ihrer Cookie-Verwaltung nicht zwischen `http` und `https` unterscheiden und daher versehentlich doch die Zugangsdaten unverschlüsselt übertragen werden könnten.

Amerkung 2: Die Anleitung zur Anmeldung via SSL funktioniert nicht für das Login-Portlet.

Management-Ansicht im öffentlichen Netz verbieten

Hierzu wird die Konfiguration des VirtualHost folgendermaßen erweitert:

```
# Forbidden HTTP status for all path components beginning with manage
RedirectMatch 403 /manage
```

Multi-Site-Konfiguration

zc.recipe.macro liefert eine Reihe von Macros, womit einzelne Buildout-Abschnitte dynamisch aus einem Macro- und einem Parameter-Abschnitt generiert werden können. Dies ermöglicht Buildout, Konfigurationsdaten unabhängig vom Ausgabeformat zu halten und ermöglicht so die Konfiguration verschiedener Dienste für mehrere Sites.

Üblicherweise wird in einem zc.recipe.macro-Abschnitt ein Macro mit den Parametern dieses Abschnitts aufgerufen. Dies läßt sich anschaulich an einer Apache-Konfiguration zeigen.

Apache-Konfiguration

```
[buildout]
parts =
    ...
    apache

[apache]
recipe = zc.recipe.macro
macro = apache-macro
result-recipe=collective.recipe.template
targets =
    my-apache:mysite-parameters
```

Im folgenden werden dann die Abschnitte `[apache-macro]` und `[mysite-parameters]` definiert:

```
[apache-macro]
domain=$${:domain}
host=$${:host}
port=$${:port}
input = ${buildout:directory}/template/apache-vhost.in
output = ${buildout:parts-directory}/$${:__name__}
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[mysite-parameters]
domain=mysite.org
host=83.223.91.163
port=8080
```

Die Datei `templates/apache-vhost.in` sieht z.B. so aus:

```
${host}
${port}
${domain}
```

Das Ergebnis ist dann:

```
[apache]
recipe = zc.recipe.macro:empty
result-sections = my-apache

[my-apache]
recipe = collective.recipe.template
domain = mysite.org
host = 83.223.91.163
input = /home/veit/myproject/template/apache-vhost.in
output = /home/veit/myproject/parts/my-apache
port = 8080
```

Das Rezept `zc.recipe.macro` ändert sich zu `zc.recipe.macro:empty`, hat jedoch keine Wirkung mehr. Dennoch muss der Abschnitt vorhanden sein, da er in `[buildout]` als Abschnitt angegeben wurde.

Und `parts/my-apache` sieht schließlich so aus:

```
83.223.91.163
8080
mysite.org
```

Jede weitere Site kann dann einfach folgendermaßen eingetragen werden:

```
[apache]
...
targets =
    ...
    vs-apache:vs-parameters

...
[vs-parameters]
domain=veit-schiele.de
host=83.223.91.163
port=8090
```

awstats-Konfiguration

Analog kann mit `tc.recipe.macro` auch awstats konfiguriert werden:

```
[buildout]
parts =
    ...
    mysite-awstats

[awstats]
recipe = zc.recipe.macro
macro = awstats-macro
result-recipe=collective.recipe.template
targets =
    mysite-awstats:mysite-parameters
    vs-awstats:vs-parameters

[awstats-macro]
domain=$${:domain}
host=$${:host}
port=$${:port}
input = ${buildout:directory}/template/awstats-conf.in
output = ${buildout:parts-directory}/$${:__name__}
```

Die Datei `templates/awstats-conf.in` sieht z.B. so aus:

```
${host}
${port}
${domain}
```

Das Ergebnis ist dann:

```
[buildout]
parts =
    ...
    awstats
    mysite-awstats
    vs-awstats

...

[awstats]
recipe = zc.recipe.macro:empty
result-sections =
    mysite-awstats
    vs-awstats

[mysite-awstats]
domain = mysite.org
host = 83.223.91.163
input = /home/veit/myproject/template/awstats-conf.in
output = /home/veit/myproject/parts/mysite-awstats
port = 8080
recipe = collective.recipe.template

[vs-awstats]
domain = veit-schiele.de
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
host = 83.223.91.163
input = /home/veit/myproject/template/awstats-conf.in
output = /home/veit/myproject/parts/vs-awstats
port = 8090
recipe = collective.recipe.template
```

Und `parts/my-awstats` sieht schließlich so aus:

```
83.223.91.163
8080
mysite.org
```

Siehe auch:

- Martin Aspeli: Tools for a successful Plone project
- Martin Aspeli: An über-buildout for a production Plone server
- Martin Aspeli: The Über-Buildout Mark II - Windows (IIS) and Unix (nginx), production and development

Monitoring

Munin-plugins

redturtle.munin stellt aktuell vier Plugins für Munin bereit:

`zopethreads` protokolliert die freien Zope-Threads

`zopecache` protokolliert die Datenbank-Cache-Parameter

`zodbactivity` protokolliert die Aktivität der ZODB

`zopememory` protokolliert die Speicherverwendung von Zope unter Linux

Dabei wird gocept.munin für die Registrierung dr Plugins verwendet.

Installation und Konfiguration

1. Zur Installation kann folgendes in der `deploy.cfg`-Datei angegeben werden:

   ```
   [instance]
   ...
   eggs =
       ...
       redturtle.munin
   zcml =
       ...
       redturtle.munin
   ```

2. Um das Plugin-Hilfsskript zu erstellen, wird zusätzlich noch ein `[munin]`-Abschnitt in die `deploy.cfg` eingetragen:

   ```
   [buildout]
   parts =
       ...
       munin1
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[munin1]
recipe = zc.recipe.egg
eggs = redturtle.munin
arguments = http_address='${instance:http-address}', user='${instance:user}'
```

Die Werte in der `arguments`-Option werden zur Generierung des Hilfsskripts verwendet, das dann als Munin-Plugin verwendet wird. Die Angaben für `ip_address`, `http_address`, `port_base` und `user` können hier kommasepariert angegeben werden.

3. Für jeden weiteren ZEO-Client wird dann ein weiterer Abschnitt eingefügt.
4. Nun sollte das Plugin folgendermaßen aufgerufen werden können:

```
http://localhost:8081/@@redturtle.munin.plugins/zopethreads
```

Der Name des View `zopethreads` entspricht dabei dem Namen des Plugins. Der View selbst kann nur mit den Rechten `View management screens` aufgerufen werden.

5. Nun werden noch Symlinks für die Hilfsskripte in `myproject/bin` zum Munin-Plugin-Verzeichnis gelegt wobei das Hilfsskript selbst diese Symlinks setzen kann:

```
$ bin/munin install /opt/munin/etc/plugins [<prefix>] [<suffix>]
```

Alternativ können die Symlinks auch selbst erstellt werden:

```
$ cd /opt/munin/etc/plugins
$ ln -s ~/myproject/bin/munin vs_zodbactivity_mysite1
```

`/opt/munin/etc/plugins` ist dabei Ihr `munin`-Verzeichnis

`~/myproject/` ist das Verzeichnis des Buildout-Projekts

`zodb_activity` ist der Name des Plugins

`vs` ist der Prefix-Wert

`mysite1` ist der Suffix-Wert, der in Munin angezeigt wird.

6. Schließlich können Sie noch überprüfen, ob das Plugin in Munin ordentlich konfiguriert ist:

```
$ cd /opt/munin/etc/plugin-conf.d/
$ vim redturtle.conf
... [vs_*_mysite1]
... env.AUTH admin:secret
... env.URL http://localhost:8081/@@redturtle.munin.plugins/%s
```

haufe.requestmonitoring

Detaillierte Request-Logging-Funktionalität für die ab Zope 2.10 verfügbaren Publication Events.

Home

http://pypi.python.org/pypi/haufe.requestmonitoring

Anforderungen

- Zope 2.12

  Für Zope 2.10 ist ein Backport der *Publication Events* verfügbar in

ZPublisherEventsBackport. Dieser wird benötigt und lässt sich mit Buildout einfach durch plone.postpublicationhook unter Angabe von `[Zope2.10]` installieren:

```
eggs =
    ...
    plone.postpublicationhook [Zope2.10]
```

Installation

Um `haufe.requestmonitoring` zu installieren, muss es einfach in der `buildout.cfg`-Datei den `zcml`-Optionen im `Instanz`-Abschnitt hinzugefügt werden:

```
[instance-base]
...
eggs +=
    ...
    haufe.requestmonitoring
    threadframe
    zope.app.appsetup
zcml +=
    ...
    haufe.requestmonitoring
```

Aktivierung

Zur Aktivierung von `haufe.requestmonitoring` erhält die Instanz einen Abschnitt `product-config` mit dem Namen `successlogging` und dem Schlüssel `filebase`. Dieser gibt den Basisnamen der Log-Dateien an, in unserem Fall `request`:

```
[instance-base]
...
zope-conf-additional =
    <product-config successlogging>
        filebase /home/veit/vs_buildout/var/log/request
    </product-config>

    %import haufe.requestmonitoring
    <requestmonitor requestmonitor>
        period 5s
        <monitorhandler dumper>
            factory haufe.requestmonitoring.DumpTraceback.factory
            # 0 means no repetition.
            # A negative value means indefinitely.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            repeat -1
            time 10s
        </monitorhandler>
    </requestmonitor>
```

Beim Logging werden dann zwei Dateien unterschieden: `<base>_good.<date>` und `<base>_bad.<date>`. Üblicherweise werden Antwortzeiten über 500 ms als *unsuccessful* bezeichnet. Falls dieser Standardwert geändert werden soll, kann ein spezieller `ISuccessFull`-Adapter registriert werden.

Nachdem das Buildout-Skript durchlaufen und der ZEO-Cluster neu gestartet wurde, sind die Subscriber `IPubStart` und `IPubSuccess` bzw. `IPubFailure` registriert. Für jedes dieser Events wird nun ein Eintrag in die Log-Datei geschrieben in der Form:

```
timestamp status request_time type request_id request_info
```

Monitoring

Mit `DumpTraceback` werden diejenigen Anfragen protokolliert, die nach der in `period` angegebenen Zeit nach Anfragen sucht, die länger laufen als die in `time` angegebene Zeit:

```
%import haufe.requestmonitoring
<requestmonitor requestmonitor>
    period 5s
    <monitorhandler dumper>
        factory Haufe.RequestMonitoring.DumpTraceback.factory
        # 0 --> no repetition
        repeat -1
        time 10s
    </monitorhandler>
</requestmonitor>
```

Eine typische Ausgabe sieht dann so aus:

```
2009-08-11 14:29:09 INFO Zope Ready to handle requests
2009-08-11 14:29:09 INFO RequestMonitor started
2009-08-11 14:29:14 INFO RequestMonitor monitoring 1 requests
2009-08-11 14:29:19 INFO RequestMonitor monitoring 1 requests
2009-08-11 14:29:24 INFO RequestMonitor monitoring 1 requests
2009-08-11 14:29:24 WARNING RequestMonitor.DumpTrace Long running request
Request 1 "/foo" running in thread -497947728 since 14.9961140156s
Python call stack (innermost first)
Module /home/junga/sandboxes/review/parts/instance/Extensions/foo.py, line 4, in foo
Module Products.ExternalMethod.ExternalMethod, line 231, in __call__
- __traceback_info__: ((), {}, None)
Module ZPublisher.Publish, line 46, in call_object
Module ZPublisher.mapply, line 88, in mapply
Module ZPublisher.Publish, line 126, in publish
Module ZPublisher.Publish, line 225, in publish_module_standard
Module ZPublisher.Publish, line 424, in publish_module
Module Products.ZopeProfiler.ZopeProfiler, line 353, in _profilePublishModule
Module Products.ZopeProfiler.MonkeyPatcher, line 35, in __call__
Module ZServer.PubCore.ZServerPublisher, line 28, in __init__
```

haufe.ztop

haufe.ztop erlaubt die Analyse von Zope-Requests zur Laufzeit.

Home

http://pypi.python.org/pypi/haufe.ztop

Anforderungen

- haufe.requestmonitoring

Installation

Um `haufe.ztop` zu installieren, muss es einfach in der `buildout.cfg`-Datei hinzugefügt werden:

```
[buildout]
parts =
    ...
   ztop
...
[ztop]
recipe = zc.recipe.egg
eggs = haufe.ztop
```

Nachdem Buildout durchlaufen wurde, stehen Ihnen die beiden Skripts `ztop` und `zanalyse` zur Verfügung.

`ztop` stellt die Request-Informationen dar indem es die Zope-Request-Logfiles auswertet. Diese werden identifiziert durch `requestsBasename` und `startDate`.

`zanalyse` gibt regelmäßig Angaben der Request-Informationen auf der Konsole aus indem es die Zope-Request-Logfiles auswertet. Diese werden identifiziert durch `requestsBasename` und `startDate`.

haufe.monitoring

Monitoring von Sets von ZEO-Clients mit Aggregation von Error Logs, Instanzen, Threads Loads und VM-Datengröße.

Home

http://pypi.python.org/pypi/haufe.monitoring

Installation

```
[instance-base]
...
eggs +=
    haufe.monitoring
zcml +=
    haufe.monitoring
```

Nachdem das Buildout-Skript durchlaufen und der ZEO-Cluster neu gestartet wurde, können Sie in den Monitoring-View `@@monitor` aufrufen, z.B. `http://mysite/@@monitor`.

Bemerkung: Da momentan noch einige Sicherheitsangaben offen sind, sollte `haufe.monitoring` aktuell nur in internen Netzen verwendet werden.

ZopeHealthWatcher

Mit ZopeHealthWatcher können die Threads der Zope-Anwendung beobachtet werden.

- ZopeHealthWatcher kann die Threads sowohl von ZEO-Clients als auch von einfachen Zope-Servern anzeigen.
- Für jeden Thread wird angegeben, ob dieser beschäftigt ist, und wenn ja, wird der *execution stack* angezeigt.
- ZopeHealthWatcher kann auch zum Debuggen von *locked threads* verwendet werden.
- ZopeHealthWatcher basiert auf dem Code von DeadlockDebugger.
- Die Darstellung erfolgt entweder auf der Konsole oder im Web-Browser.

Installation

Erweitern Sie Ihre `buildout.cfg`-Datei folgendermaßen:

```
[buildout]

parts =
     ...
    zhw

eggs =
     ...
    ZopeHealthWatcher

[zhw]
recipe = zc.recipe.egg
eggs = ZopeHealthWatcher
scripts = zope_health_watcher
```

Konfiguration

Nachdem das Egg installiert ist, müssen in `eggs/ZopeHealthWatcher-0.3-py2.6.egg/Products/ZopeHealthWatcher/custom.py` die Werte für `ACTIVATED` und `SECRET` geändert werden, z.B.:

```
ACTIVATED = True
SECRET = 'secret'
```

Verwendung

ZopeHealthWatcher kann sowohl in der Konsole als auch im Web-Browser verwendet werden.

`zope_health_watcher`-Skript

Sie können nun `zope_health_watcher` aufrufen mit der URL Ihres Zope-Servers oder ZEO-Clients:

```
$ zope_health_watcher http://localhost:8080
Idle: 3        Busy: 1
OK - Everything looks fine
```

Ist der Server mit hohem `load` ausgelastet, z.B. mit 4 Threads, werden die relevanten Informationen wie Zeit, `sysload`, Speicherinformationen und für jeden ausgelasteten Thread Stack, Query, URL als auch den *User Agent* anzeigt:

```
$ zope_health_watcher http://localhost:8080
Information:
    Time: 2011-06-02T10:52:31.522557
    Sysload: 0.25 0.18 0.20 4/1003 32523
    Meminfo: MemTotal:       11759712 kB
    MemFree:         4799368 kB
    Buffers:             204 kB
    Cached:          2933200 kB
    SwapCached:            0 kB
    Active:          4051368 kB
    Inactive:        1678532 kB
    Active(anon):    2830948 kB
    Inactive(anon):    46644 kB
    Active(file):    1220420 kB
    Inactive(file):  1631888 kB
    Unevictable:        2764 kB
    Mlocked:            2764 kB
    SwapTotal:       4000176 kB
    SwapFree:        4000176 kB
    ...
Dump:
Thread -162882704
QUERY: GET /VirtualHostBase/http/www.plone-entwicklerhandbuch.de:80/pen/
 ↪VirtualHostRoot/@@downloadPDF?
URL: http://www.plone-entwicklerhandbuch.de/@@downloadPDF
HTTP_USER_AGENT: Mozilla/5.0 (X11; U; Linux i686; de; rv:1.9.2.17) Gecko/20110422␣
 ↪Ubuntu/10.04 (lucid) Firefox/3.6.17
File "/home/veit/plone40_buildout/eggs/Zope2-2.12.17-py2.6-linux-x86_64.egg/ZServer/
 ↪PubCore/ZServerPublisher.py", line 31, in __init__
   response=b)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
...
Thread -184550544
...
```

Ist der Server nicht erreichbar, gibt das Skript folgendes aus:

```
$ ./bin/zope_health_watcher http://localhost:8080
Idle: 0 Busy: 0
FAILURE - [Errno socket error] (61, 'Connection refused')
```

ZopeHealthWatcher gibt dabei Fehlercodes so aus, dass sie auch von Nagios o.ä. weiterverarbeitet werden können:

- `OK = 0`
- `WARNING = 1`
- `FAILURE = 2`
- `CRITICAL =3`

Web-Browser

Die Ausgabe kann auch als HTML auf einen Web-Browser erfolgen, wenn z.B. folgende Adresse eingegeben wird:

```
http://www.plone-entwicklerhandbuch.de/manage_zhw?secret
```

Nagios

nagios Open-Source-Monitoring von Hosts, Services etc.

pnp4nagios PNP ist ein Addonfür Nagios für einfach zu konfigurierende, auf RRDTools basierende Performance-Charts.

nagios-check-webpage Nagios-Plugin zum Überprüfen von Web-Seiten

Weitere Informationen finden Sie in der nagios-check-webpage Documentation

Sentry

Mit Sentry lassen sich in Echtzeit Fehler agregieren und protokollieren. Dabei kann Sentry plattform-unabhängig eingesetzt werden.

Das Sentry-Paket ist in seinem Kern nur ein einfacher Web-Server mit speziellem UI. Es behandelt die Authentifizierung von Clients (wie Raven) und die gesamte Logik zur Speicherung und Aggregation. Dabei liefert Sentry eine vollständige API zum Senden von Ereignissen aus jeder Sprache in jede Anwendung.

Buildout-Konfiguration

```
[buildout]
...
eggs =
    ...
    raven

[instance1]
...
event-log-custom =
    %import raven.contrib.zope
    <logfile>
      path ${buildout:directory}/var/{:_buildout_section_name_}.log
      level INFO
    </logfile>
    <sentry>
      dsn YOUR_DSN
      level ERROR
    </sentry>
```

Siehe auch:

- Documentation

Supervisor

Supervisor ist ein Client/Server-System, das die Prozessüberwachung und -kontrolle auf Unix-Betriebssystemen erlaubt.

Supervisor Installation und Konfiguration

Supervisor ist ein Client/Server-System, das die Prozessüberwachung und -kontrolle auf Unix-Betriebssystemen erlaubt.

Dieses Python-Programm erlaubt `start`, `stop` und `restart` anderer Programme auf UNIX-Systemen wobei es auch abgestürzte Prozesse erneut starten kann.

Supervisor-Komponenten

`supervisord` *daemon*-Prozess, der andere Programme als Kind-Prozesse laufen lässt.

`supervisorctl` Client-Programm, das den Status der `supervisord`-Kind-Prozesse kontrolliert und mitloggt

Web-Interface für start, stop, restart und Ansicht der Log-Dateien.

Supervisor-Konfiguration

Die Supervisor-Konfigurationsdatei `etc/supervisord.conf` kann z.B. folgendermaßen aussehen:

```
[unix_http_server]
file=%(here)s/../var/supervisor.sock
chmod=0600

[supervisorctl]
serverurl=unix://%(here)s/../var/supervisor.sock

[rpcinterface:supervisor]
supervisor.rpcinterface_factory=supervisor.rpcinterface:make_main_rpcinterface

[supervisord]
logfile=%(here)s/../var/log/supervisord.log
logfile_maxbytes=5MB
logfile_backups=10
loglevel=info
pidfile=%(here)s/../var/supervisord.pid ;
childlogdir=%(here)s/../var/log
nodaemon=false              ; (start in foreground if true; default false)
minfds=1024                 ; (min. avail startup file descriptors; default 1024)
minprocs=200                ; (min. avail process descriptors; default 200)
directory=%(here)s

[program:zeoserver]
command = %(here)s/../bin/zeoserver fg
autostart= true
autorestart = true
startsecs = 10
priority = 100

[program: instance]
command = %(here)s/../bin/instance console
startsecs = 60
priority = 2
redirect_stderr = true
autostart= true
autorestart = true
priority = 500

[groups]
programs = instance

[program:varnish]
command = %(here)s/../bin/varnish -F
autostart= true
autorestart = true
priority = 1000
```

Supervisor kann einfach mit Buildout installiert werden:

```
[buildout]
parts =
   ...
   supervisor
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[supervisor]
recipe = zc.recipe.egg
eggs = supervisor
```

Nun kann Buildout aufgerufen und anschließend Supervisor gestartet werden:

```
$ ./bin/buildout
$ ./bin/supervisord
```

Schließlich können Sie Zeo-Server und -Client mit einem Aufruf starten und stoppen:

```
$ ./bin/supervisorctl start all
$ ./bin/supervisorctl stop all
```

Monitoring

In `etc/supervisord.conf` lässt sich auch ein Monitoring-Server konfigurieren:

```
[inet_http_server]
port=127.0.0.1:9001
username=admin
password=secret
```

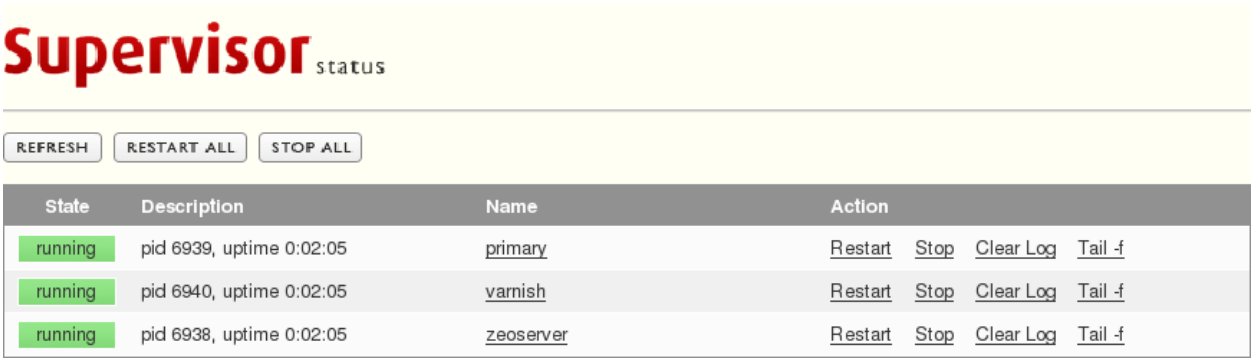
Und im Abschnitt `[supervisorctl]` muss noch die `serverurl` geändert werden:

```
[supervisorctl]
serverurl = http://localhost:9001
```

Dann starten Sie Supervisor und geben Nutzername und Passwort ein:

```
$ ./bin/supervisord
$ ./bin/supervisorctl
Server requires authentication
Username:admin
Password:

instance                         STARTING
varnish                          RUNNING    pid 21683, uptime 0:00:10
zeoserver                        STARTING
```

In Ihrem Web-Browser können Sie nun unter `http://localhost:9001` die Prozesse steuern und die Log-Dateien ansehen:



Der `supervisord`-Prozess lässt sich beenden mit:

```
supervisor> shutdown
Really shut the remote supervisord process down y/N? y
Shut down
```

Und anschließend kann auch der `supervisorctl`-Prozess beendet werden mit `Strg-D`.

superlance

superlance ist ein Plugin für supervisord zum Monitoring und Controlling der unter supervisor laufenden Prozesse.

Installation

superlance kann einfach mit Buildout installiert werden:

```
[supervisor]
recipe = collective.recipe.supervisor
plugins =
    superlance

eventlisteners =
    memmon  TICK_60  ${buildout:bin-directory}/memmon [-g app=1GB]
    httpok1  (startsecs=600)  TICK_3600  ${buildout:bin-directory}/httpok  [-p␣
↪app:instance1 -t 30 http://127.0.0.1:8010/]
    httpok2  (startsecs=600)  TICK_3600  ${buildout:bin-directory}/httpok  [-p␣
↪app:instance2 -t 30 http://127.0.0.1:8020/]
```

Nach dem Durchlaufen von Buildout sollten u.a. zusätzlich folgende Plugins im `bin`-Verzeichnis installiert sein:

`crashmail` schickt eine E-Mail-Benachrichtigung an eine vorkonfigurierte Adresse wenn ein Prozess hängt.

`memmon` startet einen Prozess neu, wenn dieser zu viel Arbeitsspeicher verbraucht. Damit kann aktiv sog. `MemoryErrors` vorgebeugt werden.

`httpok` falls Threads hängen bleiben, wird der Prozess automatisch neu gestartet

Siehe auch:

- Superlance documentation

Software Configuration Management (SCM)

Fabric

Fabric ist eine Python-Bibliothek, die die Verwaltung von Deployments und Aufgaben der Systemadministration deutlich vereinfacht.

Sie lässt sich einfach mit Buildout installieren. Hierzu erstellen wir z.B. eine `fabric.cfg`-Datei mit folgendem Inhalt:

```
[buildout]
parts += fabric

[versions]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
fabric = 1.0.0
paramiko = 1.7.6
pycrypto = 2.3

[fabric]
recipe = zc.recipe.egg
eggs =
    setuptools
    fabric
```

Anschließend kann das Skript `fabfile.py` erstellt werden:

```
from __future__ import with_statement
from fabric.api import cd, env, local, run, sudo

def www():
    env.hosts = ['www.veit-schiele.de']
    env.shell = '/bin/sh -c'
    env.sudo_prefix = "sudo -S -p '%s' -H "
    env.code_root = '/srv/www.veit-schiele.de'
    env.code_user = 'zope'

def update():
    with cd(env.code_root):
        sudo('nice svn up', user=env.code_user)

def rebuild():
    with cd(env.code_root):
        sudo('nice bin/buildout -c deploy.cfg', user=env.code_user)

def restart():
    with cd(env.code_root):
        sudo('nice bin/supervisorctl reload', user=env.code_user)

def start():
    with cd(env.code_root):
        sudo('nice bin/supervisord', user=env.code_user)

def stop():
    with cd(env.code_root):
        sudo('nice bin/supervisorctl shutdown', user=env.code_user)

def deploy():
    update()
    rebuild()
    restart()
```

Ein weiteres Beispiel zur lokalen Installation eines auf dem Server erstellten Snapshots finden Sie in diesem fabric.py.

Siehe auch:

- Fabric documentation

Performance

Performance-Tests

zope-testbrowser Mit `zope-testbrowser` lassen sich auch Performance-Tests durchführen. Dabei lässt sich für jedes Browser-Objekt angeben, wieviel Zeit jeder Request benötigte. Dies kann verwendet werden, um für einen Request eine tolerierbare Antwortzeit festzulegen. Dabei sollte jedoch nicht `lastRequestSeconds` verwendet werden, da dies unterschiedliche Zeiteinstellungen der Maschinen mitberücksichtigt, sondern `lastRequestPystones`:

```
>>> browser.open('http://localhost:8080/mysite/')
>>> browser.lastRequestPystones < 5
True
```

funkload erlaubt es komplexe Tests mit komplexen Zyklen zu schreiben und gibt ansehnliche Reporte aus.

Siehe auch:

- Connexions Developers Blog: Switching from Squid to Varnish (and getting some nice benchmarking tools along the way)
- Tarek Ziadé: Funkload + Fabric = quick and dirty distributed load system

jMeter Sie können Performance-Tests für Ihr Diazo-Theme erstellen mit JMeter

1. Installation:

   ```
   $ sudo apt-get install jmeter
   ```

2. Erstellen eines Testplans

   Nachdem Sie JMeter gestartet haben, z.B. durch die Eingabe von ``jmeter `` in der Konsole, sehen Sie einen leeren Testplan.

   Ein Testplan besteht mindestens aus den folgenden Elementen:

   1. *Thread Group* Dies ist das Wurzelelement eines Testplans. Es simuliert die Nutzer, als die Anfragen ausgeführt werden. Dabei simuliert jeder Thread einen Nutzer.
   2. ***HTTP Request Default*** Die Standardwerte für alle HTTP-Requests innerhalb einer *Thread Group*.
   3. ***HTTP Request*** Eine Stichprobe (Sampler), die verwendet werden kann um die Zeit für eine Antwort an eine bestimmte URL zu messen.
   4. **Aggregate Graph** Statistiken zu den *HTTP Request* können als aggregierte Graphen dargestellt werden.

Siehe auch:

- Getting Started with jMeter

Performance-Monitoring

pnp4nagios ist ein Addon für nagios für einfach zu konfigurierende, auf RRDTools basierende Performance-Charts.

ZCatalog

Der ZCatalog erlaubt die Indizierung und Suche in der ZODB. Darüberhinaus bietet er vielfältige Erweiterungsmöglichkeiten, mit denen sich deutlich effizientere Methoden implementieren lassen.

collective.indexing

collective.indexing ersetzt die Standard-Indizierung des CMF durch einen asynchronen Mechanismus, der redundante Indizierungen vermeidet. Damit wird die Performance zur Aktualisierung der Indizes deutlich gesteigert.

Installation

Um collective.indexing zu installieren, muss es einfach als Egg im `[buildout]`-Abschnitt eingetragen werden:

```
[buildout]
...
eggs =
    ...
    collective.indexing
```

Anschließend kann es in der `configure.zcml`-Datei eines Produkts eingetragen werden:

```
<include package="collective.indexing" />
```

Alternativ kann es auch als Wert für `zcml` im Abschnitt der Zope-Instanz eingetragen werden:

```
[instance]
...
zcml =
    collective.indexing
```

Anschließend kann `collective.indexing` für die Plone-Site installiert werden.

Siehe auch:

- Clean and fast indexing in Plone

Katalog in eigener ZODB

Wird der Katalog in einer eigenen ZODB gespeichert, können unterschiedliche Cache-Einstellungen für die Inhalte der Plone-Site und den Katalog angegeben werden. Damit werden bei einer umfangreichen Anfrage am Katalog keine Inhaltsobjekte mehr aus dem Cache verdrängt. Wird darüberhinaus die ZODB mit dem Katalog noch auf einer eigenen Platte gespeichert, lässt sich die Performance noch weiter steigern.

In ZODBs konfigurieren wird allgemein beschrieben, wie zusätzlich ZODBs angegeben werden können.

1. Fügen Sie in der `deploy.cfg`-Datei eine neue ZODB hinzu:

```
[zeoserver]
...
zeo-conf-additional =
    <filestorage 2>
        path ${buildout:directory}/var/filestorage/CatalogData.fs
    </filestorage>

[instance1]
...
zope-conf-additional =

    <zodb_db catalog>
        mount-point /mysite/portal_catalog
        container-class Products.CMFPlone.CatalogTool.CatalogTool
        cache-size 300000
        <zeoclient>
            server ${zeo:zeo-address}
            storage 2
            name catalogstorage
            var ${buildout:parts-directory}/instance1/var
            cache-size 400MB
        </zeoclient>
    </zodb_db>
```

2. Rufen Sie `./bin/buildout` aufund starten anschließend den ZEO-Cluster.
3. Nun erstellen wir eine neue Plone-Site mit der ID `mysite`.
4. Löschen Sie `portal_catalog` in dieser Plone-Site.

   Beachten Sie, dass die Plone-Site anschließend nicht mehr funktioniert.
5. Gehen Sie in das ZMI dieser Site und wählen dort *ZODB Mount Point* aus.
   1. Im folgenden Formular sollte der `portal_catalog`-Mount-Point verfügbar sein.
   2. Aktivieren Sie *create missing folders...*
6. Wechseln Sie anschließend in das `portal_catalog`-Objekt.
7. Wechseln Sie in den *advanced*-Reiter und aktivieren dann *clear and rebuild*. Beachten Sie, dass dies einige Zeit dauern kann.

Schließlich ist der Katalog Ihrer Plone-Site in einem eigenen Mount-Point verfügbar.

Query-Plan

Üblicherweise iteriert die Suchmethode des ZCatalog über alle beteiligten Indizes und bildet anschließend jedesmal die Schnittmenge. Dabei bleibt die Reihenfolge der Indizes, an die die Anfrage gestellt wird, unbestimmt. Dieses Vorgehen lässt sich nun optimieren wenn die Anfrage zunächst an Indizes gestellt wird, dessen Trefferwahrscheinlichkeiten am geringsten sind.

experimental.catalogqueryplan berechnet die durchschnittliche Trefferwahrscheinlichkeit jedes Index. Darauf aufbauend werden die Indizes so sortiert, dass zunächst die Anfragen mit den wenigsten Ergebnissen abgearbeitet werden.

Installation

Um `experimental.catalogqueryplan` zu installieren, wird es im `[buildout]`-Abschnitt eingetragen:

```
[buildout]
...
eggs =
    ...
    experimental.catalogqueryplan
```

Anschließend kann es in der `configure.zcml`-Datei eines Produkts eingetragen werden:

```
<include package="experimental.catalogqueryplan" />
```

Alternativ kann es auch als Wert für `zcml` im Abschnitt der Zope-Instanz eingetragen werden:

```
[instance]
...
zcml =
    experimental.catalogqueryplan
```

Bogus-Index-Names

Um unterschiedliche Query-Plans für ähnlich Anfragen zu erhalten, ermöglicht `experimental.catalogqueryplan` auch zusätzliche *Bogus Index Names*. Diese werden zwar vom Katalog ignoriert, sind jedoch Teil des Schlüssels zur Erstellung des *Query Plans*. So wird z.B. die Suche nach Seiten im Entwurfsstadium in einer anderen Reihenfolge die Indizes abarbeiten als die suche nach veröffentlichten Seiten, da sich wohl nur sehr wenige Artikel im Entwurfsstadium auf einer Site finden dürften.

Logging langsamer Kataloganfragen

In Version 1.4 wird `experimental.catalogqueryplan` auch langsame Anfragen in das *Event-Log* schreiben. Hierzu werden die Umgebungsvariablen im Abschnitt für den oder die Zope-Server in die Buildout-Konfigurationsdatei eingetragen:

```
[instance]
...
zope-conf-additional =
    <environment>
        LOG_SLOW_QUERIES True
        LONG_QUERY_TIME  0.05
    </environment>
```

`LOG_SLOW_QUERIES` Wird der Wert auf `True` gesetzt, werden langsame Anfragen in das *Event-Log* geschrieben.

`LONG_QUERY_TIME` Nur diejenigen Anfragen, die länger als die hier angegebene Zeit in Sekunden dauern, werden in das *Event-Log* geschrieben.

Der Standardwert ist `0.01`.

Die Ausgabe im *Event-Log* kann dann z.B. so aussehen:

```
2009-04-17T16:56:47 INFO experimental.catalogqueryplan portal_catalog, query: 0.11ms␣
↪(hits: 0), mean 128.41ms (key hits: 11),  priority: ('path', 'review_state', 'is_
↪default_page', 'allowedRolesAndUsers', 'portal_type')
```

Siehe auch:

- Wikipedia: Query plan
- Jarn: Catalog query plan
- experimental.catalogqueryplan: General ideas

unimr.catalogqueryplan

Üblicherweise iteriert die Suchmethode des ZCatalog über alle beteiligten Indizes und bildet anschließend jedesmal die Schnittmenge. Dabei bleibt die Reihenfolge der Indizes, an die die Anfrage gestellt wird, unbestimmt. Dieses Vorgehen lässt sich nun optimieren wenn die Anfrage zunächst an Indizes gestellt wird, dessen Trefferwahrscheinlichkeiten am geringsten sind.

Bemerkung: Ab Zope 2.13, das mit Plone 4.1 kommt, unterstützt der ZCatalog Query Plans, s.a. ZCatalog.

Der Monkey Patch des unimr.catalogqueryplan berechnet die durchschnittliche Trefferwahrscheinlichkeit und Verarbeitungsdauer jedes Index. Darauf aufbauend werden die Indizes so sortiert, dass zunächst die effektivsten Anfragen abgearbeitet werden. Hierdurch kann die durchschnittliche Dauer von Anfragen auf die Hälfte reduziert werden.

Installation

Um `unimr.catalogqueryplan` zu installieren, wird ein Abschnitt `[eggcheckouts]` im `[buildout]`-Abschnitt eingetragen, anschließend dieser Abschnitt definiert und schließlich die Eggs dem ZEO-Client zur Verfügung gestellt:

```
[buildout]
parts =
    ...
    eggcheckouts
...
[eggcheckouts]
recipe = infrae.subversion
urls =
    https://svn.plone.org/svn/collective/unimr.catalogqueryplan/trunk unimr.
↪catalogqueryplan
location = src
as_eggs = true

[instance]
...
eggs =
    ...
   ${eggcheckouts:eggs}
```

Logging langsamer Kataloganfragen

Darüberhinaus kann `unimr.catalogqueryplan` langsame Anfragen in das *Event-Log* schreiben. Hierzu werden die Umgebungsvariablen im Abschnitt für den oder die Zope-Server in die Buildout-Konfigurationsdatei eingetragen:

```
[instance]
...
zope-conf-additional =
    <environment>
        LOG_SLOW_QUERIES True
        LONG_QUERY_TIME  0.05
    </environment>
```

LOG_SLOW_QUERIES Wird der Wert auf `True` gesetzt, werden langsame Anfragen in das *Event-Log* geschrieben.

LONG_QUERY_TIME Nur diejenigen Anfragen, die länger als die hier angegebene Zeit in Sekunden dauern, werden in das *Event-Log* geschrieben.

Der Standardwert ist `0.0`.

Die Ausgabe im *Event-Log* kann dann z.B. so aussehen:

```
2009-04-17T16:56:47 INFO unimr.catalogqueryplan portal_catalog, query: 0.11ms (hits:␣
↪0), mean 128.41ms (key hits: 11),  priority: ('path', 'review_state', 'is_default_
↪page', 'allowedRolesAndUsers', 'portal_type')
```

Aktualisierungsfrequenz

Wie oft der *Query-Plan* aktualisiert wird, lässt sich ebenfalls über Umgebungsvariablen in der Buildout-Konfigurationsdatei angeben:

```
[instance]
...
zope-conf-additional =
    ...
    <environment>
        REFRESH_RATE 500
    </environment>
```

REFRESH_RATE Die Zeitspanne, nach der ein neuer *Query Plan* berechnet wird.

Der Standardwert ist `100`.

Bogus-Index

Um unterschiedliche *Query-Plans* für ähnliche Anfragen zu erhalten, können zusätzliche *Bogus-Iindizes* bereitgestellt werden. Diese werden zwar im Katalog ignoriert, können aber dennoch zur Erstellung eines *Query-Plan* verwendet werden.

Composite Index

CompositeIndex ist ein Index für den ZCatalog, der mehr als ein Attribut je Objekt enthalten kann.Solche Indizes sollten erstellt werden, wenn Anfragen mit mehreren Attributen in einer Suche zu erwarten ist und die kombinierte Suche deutlich weniger Ergebnisse erwarten lässt als die einzelnen Suchen mit nur einem Attribut. Hiervon profitieren vor allem große Sites mit mehr als 100.000 Objekten, bei denen die Anfragen zwei- bis dreimal schneller abgearbeitet werden können.

Viele Kataloganfragen basieren auf er Kombination indizierter Attribute. Üblicherweise arbeitet der ZCatalog jede dieser Anfragen sequentiell ab und berechnet die Schnittmenge zwischen jedem dieser Ergebnisse. Für große Sites mit vielen Objekten reduziert diese Strategie die Performance von Kataloganfragen deutlich. Der `CompositeIndex` von unimr.compositeindex hingegen kann jedoch bereits die Schnittmenge von Indizees der Typen `FieldIndex` und `KeywordIndex` bilden.

Installation

Um `unimr.compositeindex` zu installieren, muss es einfach als Egg im `[buildout]`-Abschnitt eingetragen werden:

```
[buildout]
...
eggs =
    ...
    unimr.compositeindex
```

Anschließend kann es in der `configure.zcml`-Datei eines Produkts eingetragen werden:

```
<include package="unimr.compositeindex" />
```

Alternativ kann es auch als Wert für `zcml` im Abschnitt der Zope-Instanz eingetragen werden:

```
[instance]
...
zcml =
    unimr.compositeindex
```

Verwendung

Für das *Generic Setup*-Profil wird die `catalog.xml`-Datei erstellt mit folgendem Inhalt:

```
<?xml version="1.0"?>
<object name="portal_catalog" meta_type="Plone Catalog Tool">
    <index name="comp01" meta_type="CompositeIndex">
        <indexed_attr value="is_default_page"/>
        <indexed_attr value="review_state"/>
        <indexed_attr value="portal_type"/>
        <indexed_attr value="allowedRolesAndUsers"/>
    </index>
    <column value="comp01"/>
</object>
```

`index name` Eine gültige ID Ihrer Wahl.

`indexed_attr value` Name des Attributs eines Objekts, das in einer Anfrage verkettet werden soll.

Anschließend sollte im ZMI im *Indexes*-Reiter des Catalog Tools für diesen *CompositeIndex* noch *Reindex* angegeben werden. Jede Anfrage mit zwei oder mehr Komponenten des `Composite key` wird nun automatisch umgewandelt in eine Anfrage an den *CompositeIndex*.

Chameleon

Chameleon ist eine Reimplementierung der ZopePageTemplates, die durch Precompiling eine deutliche Performance-Optimierng erlaubt. Durchschnittlich dürfte isch so eine Performance-Steigerung für ungecachte Inhalte um ca. 20% ergeben.

Chameleon kann einfach für Plone 4 installiert werden indem als einzige Abhängigkeit five.pt in einer Version 2.1 installiert wird:

```
[instance]
...
eggs =
    ...
    five.pt
```

Die automatische Paketkonfiguration von Plone installiert dann automatisch Chameleon nach.

Eine ausführliche Dokumentation zu Chameleon erhalten Sie in Chameleon documentation.

Lastverteilung mit HAProxy

Mit *Varnish* lässt sich zwar eine einfache Lastverteilung im *Round-Robin*-Verfahren realisieren, für zuverlässige Hochverfügbarkeit sind jedoch aufwendigere Verfahren notwendig.

Dies kann mit HAProxy realisiert werden. Dabei lässt sich HAProxy einfach mit Buildout installieren:

```
[haproxy-build]
recipe = plone.recipe.haproxy
url = http://www.haproxy.org/download/1.4/src/haproxy-1.4.26.tar.gz
target = linux26
cpu = i686
pcre = 1

[haproxy-config]
recipe = collective.recipe.template
input = ${buildout:directory}/templates/haproxy.conf.in
output = ${buildout:directory}/etc/haproxy.conf
user = haproxy
group = haproxy
frontend-host = 127.0.0.1
frontend-port = 8001
```

Die `haproxy.conf.in`-Datei in `templates` sieht dann z.B. so aus:

```
global
  maxconn  ${haproxy-conf:maxconn}
  user ${haproxy-config:user}
  group ${haproxy-config:group}
  daemon
  nbproc 1
  spread-checks 3
  ulimit-n 65536
```

(Fortsetzung auf der nächsten Seite)

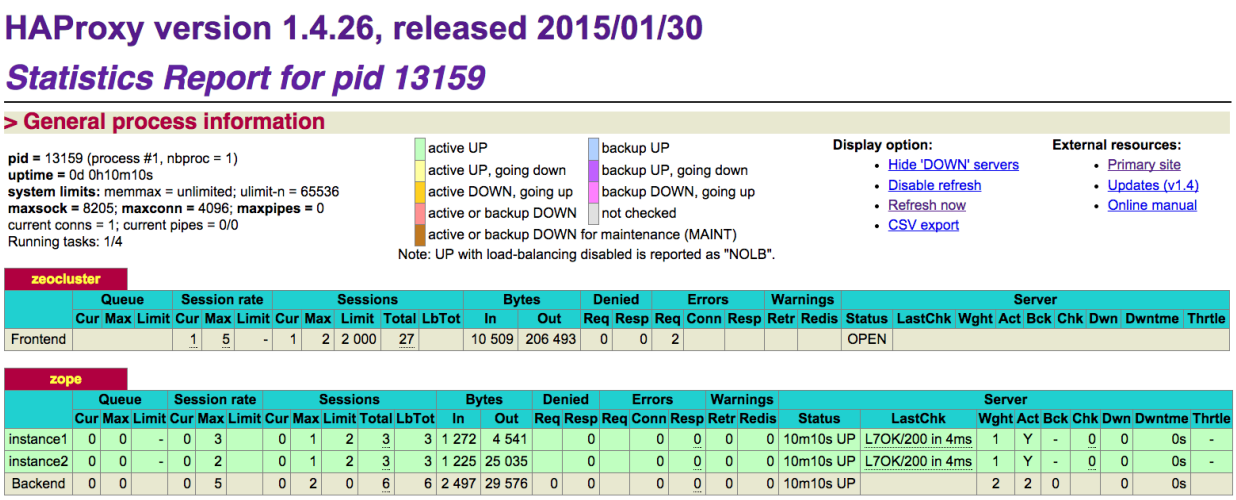
(Fortsetzung der vorherigen Seite)

```
defaults
  mode http

  # The zope instances have maxconn 1, and it is important that
  # unused/idle connections are closed as soon as possible.
  option httpclose

  # Remove requests from the queue if people press stop button
  option abortonclose

  retries 3
  option redispatch
  monitor-uri /haproxy-ping

  timeout connect 5s
  timeout queue 30s
  timeout client 50s
  timeout server 120s
  timeout check 50s
  stats enable
  stats uri /haproxy-status
  stats refresh 60s
  stats realm Haproxy\ statistics

frontend zeocluster
  bind ${haproxy-config:frontend-host}:${haproxy-config:frontend-port}
  default_backend zope

  option httplog
  log 127.0.0.1:1514 local6


  # Load balancing over the zope instances
  backend zope
  # Use Zope's __ac cookie as a basis for session stickiness if present.
  appsession __ac len 32 timeout 1d
  # Otherwise add a cookie called "serverid" for maintaining session stickiness.
  # This cookie lasts until the client's browser closes, and is invisible to Zope.
  cookie serverid insert nocache indirect
  # If no session found, use the roundrobin load-balancing algorithm to pick a␣
↪backend.
  balance roundrobin
  # Use / (the default) for periodic backend health checks
  option httpchk GET /

  # Server options:
  # "maxconn" is how many connections can be sent to the server at once
  # "check" enables health checks
  # "rise 1" means consider Zope up after 1 successful health check
  server instance1 127.0.0.1:${instance1:http-address} weight 1 check inter 15s rise␣
↪2 fall 1 maxconn 2
  server instance2 127.0.0.1:${instance1:http-address} weight 1 check inter 15s rise␣
↪2 fall 1 maxconn 2
```

HAProxy kann dann gestartet werden mit:

```
${buildout:directory}/bin/haproxy -f ${buildout:directory}/etc/haproxy.conf -db
```

Unter http://localhost:8001/haproxy-status können Sie sich dann den aktuellen Status des HAProxy anschauen:



HAProxy version 1.4.26, released 2015/01/30

Statistics Report for pid 13159

> General process information

pid = 13159 (process #1, nbproc = 1)
uptime = 0d 0h10m10s
system limits: memmax = unlimited; ulimit-n = 65536
maxsock = 8205; **maxconn =** 4096; **maxpipes =** 0
current conns = 1; current pipes = 0/0
Running tasks: 1/4

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.4)
- Online manual

zeocluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	5	-	1	2	2 000	27		10 509	206 493	0	0	2					OPEN							

zope

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
instance1	0	0	-	0	3		0	1	2	3	3	1 272	4 541		0		0	0	0	0	10m10s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
instance2	0	0	-	0	2		0	1	2	3	3	1 225	25 035		0		0	0	0	0	10m10s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-
Backend	0	0		0	5		0	2	0	6	6	2 497	29 576	0	0		0	0	0	0	10m10s UP		2	2	0		0	0s	

Siehe auch:

- Analyze ALOHA's HAProxy logs with halog

Security

Security Advisories

Plone Announce Mailinglist-Mailingliste Eine Mailingliste mit extrem geringem Datenaufkommen, für neue Releases und Sicherheitshinweise. Die Liste ist moderiert und nur das Plone-Team kann auf dieser Liste veröffentlichen.

RSS Feed of Plone Security Advisories RSS 1.0-Feed

Das Abonnieren der Mailingliste oder des RSS-Feed wird unbedingt empfohlen.

Plone-Hotfixes

Einen Überblick, welche Plone-Versionen welche Hotfixes benötigen, erhalten Sie unter Plone Hotfixes.

plone.protect

plone.protect bietet Methoden um die Sicherheit von Web-Formularen in Plone zu erhöhen.

Einschränken der Requests auf HTTP-POST

```
from plone.protect import PostOnly
from plone.protect import protect

@protect(PostOnly)
def something(self, param, REQUEST=None):
    pass
```

Form authentication (CSRF)

Cross-Site-Request-Forgery CSRF ist ein Angriffsverfahren, bei dem ein ein HTTP-Formular an einen anderen Ort übermittelt wird und anschließend die Parameter ausgewertet werden. *Form authentication* soll verhindern, dass diese Parameter ausgewertet werden können da zunächst die Authentizität überprüft wird.

Der erforderliche Token kann einfach generiert werden mit:

```
<span tal:replace="structure context/@@authenticator/authenticator"/>
```

Zur Überprüfung des Tokens können verschiedene Methoden verwendet werden:

1. ZCA:

   ```
   authenticator=getMultiAdapter((context, request), name=u"authenticator")
   if not authenticator.verify():
       raise Unauthorized
   ```

2. mit einem Decorator:

   ```
   from plone.protect import CheckAuthenticator
   from plone.protect import protect

   @protect(CheckAuthenticator)
   def something(self, param, REQUEST=None):
       pass
   ```

3. Anfrage an einen Funktionsvalidator weiterreichen:

   ```
   from plone.protect import CheckAuthenticator
   ...
   CheckAuthenticator(self.context.REQUEST)
   ...
   ```

Automatischer CSRF-und Clickjacking-Schutz

Seit Version 3 bietet `plone.protect` einen automatischen CSRF-Schutz indem automatisch ein Auth-Token in allen internen Formularen verwendet wird, wenn der Benutzer angemeldet ist oder in die ZODB geschrieben werden soll.

Zum Schutz vor Clickjacking setzt Plone zudem den X-Frame-Options-Header auf `SAMEORIGIN`.

Um diesen Wert zu ändern gibt es drei Möglichkeiten:

1. In einem View überschreiben, z.B. mit:

```
self.request.response.setHeader ('X-Frame-Options "," AllowAll'))
```

2. Im Proxy-Server überschreiben
3. Die Umgebungsvariable `PLONE_X_FRAME_OPTIONS` ändern

s.a. Debugging CSRF Protection False Positives in Plone

Siehe auch:

Plone Developer Documentation: Security Zope security facilities, Sandboxing and SELinux

PySprint: Sicherheit und Datenschutz bei Zope-Anwendungen Anhand des deutschen Bundesdatenschutzgesetz (BDSG) wird überprüft, wie Zope-Anwendungen diesen Anforderungen gerecht werden können.

WebLion: Secure Zope Declare IP-Addresses and *iptables* config

Steve McMahon, Eric Rose: Protecting Plone From The Big Bad Internet Presentation from Plone Conference 2008 in Washington, D.C.

Security overview of Plone The ten most common security issues in web applications, and how Plone addresses them.

zopyx.plone.cassandra Show all assigned local roles within a subtree for any Plone 4 site.

iptables

iptables-Konfiguration

Auf der virtuellen Maschine mit dem ZEO-Server ist der Port 9000 zu öffnen:

```
# system-config-firewall-tui

Firewall-Konfiguration: Anpassen
Trusted Dienste: WWW (HTTP)
Andere Ports: Hinzufügen
Port/Port-Bereich: 9000
Protokoll: tcp
```

Dies generiert die Datei `/etc/sysconfig/iptables`:

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9000 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Eingehende Anfragen lassen sich überprüfen mit:

```
# watch iptables --list -v
Every 2,0s: iptables --list -v                                                  ␣
↪           Wed Mar 20 18:30:34 2013

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1476  202K ACCEPT     all  --  any    any     anywhere             anywhere      ␣
↪  state RELATED,ESTABLISHED
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
    1    60 ACCEPT     all  --  lo     any     anywhere             anywhere
    1    60 ACCEPT     tcp  --  any    any     anywhere             anywhere      ␣
↪  state NEW tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere      ␣
↪  state NEW tcp dpt:nfs
    6   360 ACCEPT     tcp  --  any    any     anywhere             anywhere      ␣
↪  state NEW tcp dpt:cslistener
   18   576 REJECT     all  --  any    any     anywhere             anywhere      ␣
↪  reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REJECT     all  --  any    any     anywhere             anywhere      ␣
↪  reject-with icmp-host-prohibited

Chain OUTPUT (policy ACCEPT 1405 packets, 117K bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Nun können wir uns die aktiven Internetverbindungen anschauen mit:

```
# netstat -tulpen
Aktive Internetverbindungen (Nur Server)
Proto Recv-Q Send-Q Local Address               Foreign Address             State      ␣
↪  Benutzer   Inode      PID/Program name
...
...        0      0 0.0.0.0:9000                0.0.0.0:*                   LISTEN     ␣
↪  502        1302676    23572/python
...
```

Analog sollten nun auch `iptables` für die Instanzen konfiguriert werden.

3.12 Authentifizierung

3.12.1 Emergency user

Sollten bei einer bereits existierenden Zope-Datenbank die Zugangsdaten verlorengegangen oder die Sicherheitseinstellungen so verändert haben, dass Sie selbst nicht mehr die notwendigen Änderungen vornehmen können, so gibt es ein Skript für die Zope-Instanz, mit dem Sie erneut Zugang zum ZMI erhalten können. Dieses Skript kann im Buildout-Verzeichnis folgendermaßen aufgerufen werden:

```
$ cd parts/instance/
$  python ../../parts/zope2/utilities/zpasswd.py access
Username: veit
Password:
Verify password:
Please choose a format from:

SHA - SHA-1 hashed password (default)
CRYPT - UNIX-style crypt password
CLEARTEXT - no protection

Encoding: SHA
Domain restrictions:
```

encoding gibt die Art der Verschlüsselung an.

domains Domainnamen, von denen aus dieser Nutzer sich anmelden kann.

Anschließend sollte sich die Instanz starten lassen und der soeben erzeugte Nutzer sollte sich nun im ZMI anmelden können. Innerhalb des ZMI hat dieser sog. *emergency user* dann allerdings nicht die vollen Administrationsrechte sondern nur diejenigen zum Anlegen neuer Nutzer und Ändern der Sicherheitseinstellungen. Sie können also nun z.B. einen neuen Nutzer mit Management-Rechten anlegen, anschließend die Instanz stoppen. Nach dem erneuten Starten der Instanz sollten Sie als der im ZMI angelegt Nutzer wieder anmelden können. Vergessen Sie bitte nicht, die `access`-Datei wieder zu löschen.

3.12.2 LDAP

Installieren und Konfigurieren des LDAP-Servers

Die LDAP-Verbindung wird anhand eines OpenLDAP-Servers demonstriert.

Installation

OpenLDAP kann heruntergeladen werden von http://www.openldap.org/. Es ist jedoch auch in vorkonfigurierten Pakete für viele Betriebssysteme verfügbar. Für Debian und Ubuntu können Sie diese installieren mit:

```
# apt-get install slapd ldapscripts
```

Konfiguration

Falls der LDAP-Server noch nicht gestartet wurde, kann dies mit folgender Angabe erfolgen:

```
# /etc/init.d/slapd restart
```

Anschließend lassen sich einige der von Ubuntu im `LDIF`-Format mitgelieferten Schemata hinzufügen:

```
# ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/cosine.ldif
# ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/inetorgperson.ldif
# ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/nis.ldif
```

Nachdem diese allgemeinen Schemata hinzugefügt worden sind, wird nun noch eine initiale `cn=config`-Datenbank aufgesetzt, die die gesamte Konfiguration des OpenLDAP-Servers enthält. Hierzu erstellen wir zunächst die Datei `db.ldif` mit folgendem Inhalt:

```
#######################################################
# DATABASE SETUP
#######################################################

# Load modules for database type
dn: cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb

# Create directory database
dn: olcDatabase={1}hdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcHdbConfig
olcDatabase: {1}hdb
olcDbDirectory: /var/lib/ldap
olcSuffix: dc=veit-schiele,dc=de
olcRootDN: cn=admin,dc=veit-schiele,dc=de
olcRootPW: 1234
olcAccess: {0}to attrs=userPassword,shadowLastChange by dn="cn=admin,dc=veit-schiele,
↪dc=de" write by anonymous auth by self write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to * by dn="cn=admin,dc=veit-schiele,dc=de" write by * read
olcLastMod: TRUE
olcDbCheckpoint: 512 30
olcDbConfig: {0}set_cachesize 0 2097152 0
olcDbConfig: {1}set_lk_max_objects 1500
olcDbConfig: {2}set_lk_max_locks 1500
olcDbConfig: {3}set_lk_max_lockers 1500
olcDbIndex: uid pres,eq
olcDbIndex: cn,sn,mail pres,eq,approx,sub
olcDbIndex: objectClass eq


#######################################################
# DEFAULTS MODIFICATION
#######################################################
# Some of the defaults need to be modified in order to allow
# remote access to the LDAP config. Otherwise only root
# will have administrative access.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
dn: cn=config
changetype: modify
delete: olcAuthzRegexp

dn: olcDatabase={-1}frontend,cn=config
changetype: modify
delete: olcAccess

dn: olcDatabase={0}config,cn=config
changetype: modify
add: olcRootPW
olcRootPW: {CRYPT}7hzU8RaZxaGi2

dn: olcDatabase={0}config,cn=config
changetype: modify
delete: olcAccess
```

Diese Konfiguration wird nun mit folgendem Befehl eingelesen:

```
# ldapadd -Y EXTERNAL -H ldapi:/// -f db.ldif
```

Nun sollten noch minimale Einträge für den *LDAP DIT* (Directory Information Tree) angelegt werden. Hierzu erstellen wir eine weitere Datei `base.ldif` mit folgendem Inhalt:

```
# Top level - the organization
dn: dc=veit-schiele,dc=de
objectClass: dcObject
objectclass: organization
o: veit-schiele.de
dc: veit-schiele
description: Top level

# LDAP admin
dn: cn=admin,dc=veit-schiele,dc=de
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
userPassword: 1234
description: LDAP admin
```

Dadurch wird der Benutzer `"cn=admin,dc=veit-schiele,dc=de"` mit dem Passwort `1234` erstellt, der anschließend alle Rechte am LDAP-Server hat. Das Passwort sollte natürlich angepasst werden.

Diese Datei wird eingelesen mit:

```
# ldapadd -x -D cn=admin,dc=veit-schiele,dc=de -W -f base.ldif
```

Zum Testen kann nun der Directory Information Tree ausgelesen werden mit:

```
ldapsearch -xLLL -b dc=veit-schiele,dc=de
```

Nun sollten wir noch einige Einträge in das LDAP-Repository erstellen. Hierzu erzeugen wir die Datei `veit-schiele.ldif`:

```
# Second level - organizational units
dn: ou=people, dc=veit-schiele,dc=de
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
ou: people
description: All people in organisation
objectclass: organizationalunit

dn: ou=groups, dc=veit-schiele,dc=de
ou: groups
description: All groups in the organization
objectclass: organizationalunit

# Third level - people
dn: uid=vschiele,ou=people,dc=veit-schiele,dc=de
objectClass: pilotPerson
objectClass: uidObject
uid: vschiele
cn: Veit Schiele
sn: Schiele
userPassword:: e1NIQX01ZW42RzZNZXpScm9UM1hLcWtkUE9tWS9CZlE9
mail: kontakt@veit-schiele.de

# Third level - groups
dn: cn=Staff,ou=groups,dc=veit-schiele,dc=de
objectClass: top
objectClass: groupOfUniqueNames
cn: Staff
uniqueMember: uid=vschiele,ou=people,dc=veit-schiele,dc=de
```

- Die Organisation `dc=veit-schiele,dc=de` erhält zunächst zwei Organisationseinheiten (*organizational units*): *people* und *groups*.
- Nun wird ein Nutzer `vschiele` der Organisationseinheit *people* und eine Gruppe `staff` der Organisationseinheit *groups* hinzugefügt.
- Die Eigenschaft `userPassword` ist durch die beiden Doppelpunkte als SHA1 Hash-Wert gekennzeichnet.
- Schließlich kann die LDIF-Datei importiert werden mit:

```
# ldapadd -xWD 'cn=admin, dc=veit-schiele,dc=de' -f veit-schiele.ldif
```

Plone mit LDAP-Server verbinden

Installation des `python-ldap`-Moduls

Um LDAP mit Python nutzen zu können, muss das `python-ldap`-Modul installiert sein. Sie können testen, ob dieses Modul der Zope-Instanz zur Verfügung steht mit:

```
$ ./bin/zopepy
$ import _ldap
```

Sollten Sie keine Fehlermeldung erhalten. können Sie den Python-Interpreter wieder mit `Strg-D` (unter Windows `Strg-Z`) verlassen.

Falls Sie eine Fehlermeldung erhalten, können Sie das `python-ldap`-Modul einfach mit Buildout installieren. Überprüfen Sie zunächst, ob die erforderlichen Pakete bereits installiert sind:

OpenLDAP-Client Version 2.3

OpenSSL (optional) http://www.openssl.org/

cyrus-sasl (optional) http://asg.web.cmu.edu/sasl/sasl-library.html

Kerberos-Bibliotheken (optional) *MIT* oder *heimdal*

Für Debian und Ubuntu können Sie diese installieren mit:

```
$ sudo apt-get install libldap2-dev libsasl2-dev libssl-dev
```

Anschließend können Sie Ihre `base.cfg`-Datei folgendermaßen ändern:

```
[buildout]
parts =
    python-ldap
    ...

[python-ldap]
recipe = zc.recipe.egg:custom
eggs = python-ldap
find-links =
include-dirs = /usr/include /usr/lib/sasl2
library-dirs = /usr/lib
```

Die hier für `include-dirs` und `library-dirs` eingetragenen Pfade sind die für Ubuntu erforderlichen Angaben.

Weitere Informationen zur Installation des `python-ldap`-Moduls erhalten Sie unter http://www.python-ldap.org/doc/html/installing.html.

Installation von `plone.app.ldap`

Hierzu tragen wir in der `setup.py`-Datei unseres Policy-Produkts folgendes ein:

```
setup(name='vs.policy',
    ...
    install_requires=[
        ...
          'plone.app.ldap',
    ],
```

Das `plone.app.ldap`-Egg installiert automatisch die erforderlichen Produkte LDAPMultiPlugins, LDAPUserFolder, PloneLDAP, dataflake.fakeldap und python-ldap mit.

Anschließend wird noch in der `metadata.xml`-Datei des Policy-Produkts `plone.app.ldap` als Abhängigkeit eingetragen:

```
<?xml version="1.0"?>
<metadata>
    ...
    <dependencies>
        ...
        <dependency>profile-plone.app.ldap:ldap</dependency>
    </dependencies>
</metadata>
```

Nachdem das Buildout-Projekt aktualisiert, die Zope-Instanz neu gestartet und eine neue Plone-Site mit `vs.policy` hinzugefügt wurde, sollte sich *LDAP Connection* in *Website-Konfiguration* konfigurieren lassen.

Konfiguration

Anschließend erscheint in der *Website-Konfiguration* die Konfiguration der *LDAP Connection*:

LDAP-server type unterscheidet zwischen *LDAP* und *Active Directory*.

rDN attribute *relative distinguished name*, wird als erster Abschnitt des `DN` für in Plone angelegte Nutzer verwendet.

Für *LDAP* wird hier meist `uid` angegeben, für *Active Directory* hingegen `cn`.

user id attribute Attribut, das für die ID von Nutzern angegeben werden soll.

Auch hier wird dasselbe wie für den *relative distinguished name* angegeben.

login name attribute wird während der Authentifizierung verwendet und ist üblicherweise dieselbe Angabe wie bei *user id attribute*.

Bei einem *Active Directory*-Server kann entweder `userPrincipalName` oder `sAMAccountName` als *user id* und *login name* verwendet werden. Dabei wird für `sAMAccountName` nur der einfache Nutzername angegeben während `userPrincipalName` auch den Domainnamen enthält. `sAMAccountName` ist daher gut geeignet, wenn das Active Directory nur für eine Domain konfiguriert werden muss.

LDAP object classes Strukturelle Objektklasse und weitere Objektklassen, durch Komma getrennt. Diese Objektklassen werden in Plone neu hinzugefügten Nutzern zugewiesen. Das hier verwendete `PilotPerson, uidObject` ergibt eine einfaches Objekt mit einer `uid`.

Bind DN ist der *distinguished name*, der von Plone verwendet wird um Zugang zum LDAP-Server zu erhalten. Wir verwenden den `rootdn`:

```
"cn=admin,dc=veit-schiele,dc=de"
```

Bind password ist das Passwort für den in *Bind DN* angegebenen Nutzer, in unserem Fall also das in der `slpad.conf`-Datei angegebene Passwort `1234`.

Base DN for users Ort im LDAP-Repository, an dem die Nutzer verwaltet werden. In unserem Fall in der *organizational unit* `people`:

```
ou=people,dc=veit-schiele,dc=de
```

Search scope for users Angabe, ob Nutzer direkt in der *organizational unit* gefunden werden oder auch in Untereinheiten. In unserem Fall könnten wir zwar `one level` angeben, `subtree` ist jedoch flexibler und wird daher üblicherweise verwendet.

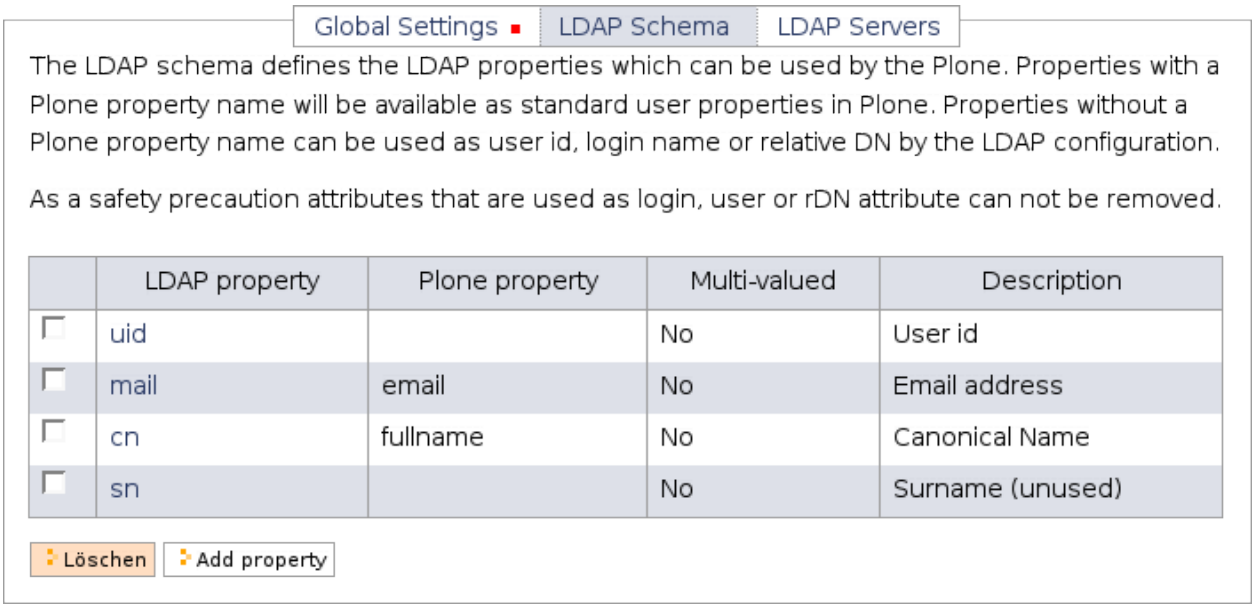
Base DN for groups Ort im LDAP-Repository, an dem die Gruppen verwaltet werden. In unserem Fall in der *organizational unit* `groups`:

```
ou=groups,dc=veit-schiele,dc=de
```

Search scope for groups Angabe, ob Gruppen direkt in der *organizational unit* gefunden werden oder auch in Untereinheiten. Auch hier geben wir wie in *Search scope for users* `subtree` an.

LDAP-Schema

Im *LDAP Schema*-Reiter können LDAP-Attribute mit Eigenschaften der Plone-Nutzer verknüpft werden.



Üblicherweise wird das `uid`-Attribut nicht mit einer Plone-Eigenschaft verknüpft, da es im *Global Settings*-Reiter frei als *rDN*, *user id* und *login name* angegeben werden soll.

Auch *sn* wird hier angegeben, jedoch nicht mit einer Plone-Eigenschaft verknüpft. Dies führt dazu, dass für jeden neu angelegten Plone-Nutzer ein leeres Feld im LDAP-Repository angelegt wird. Dennoch darf *sn* nicht aus der Plone-LDAP-Konfiguration gelöscht werden, da die *sn*-Eigenschaft zwingend für die *pilotPerson*-Objektklasse erforderlich ist.

Plone-Eigenschaften können mit LDAP-Attributen gemischt werden wobei der *Pluggable Authentication Service* (PAS) die LDAP-Eigenschaften bevorzugt.

LDAP-Server

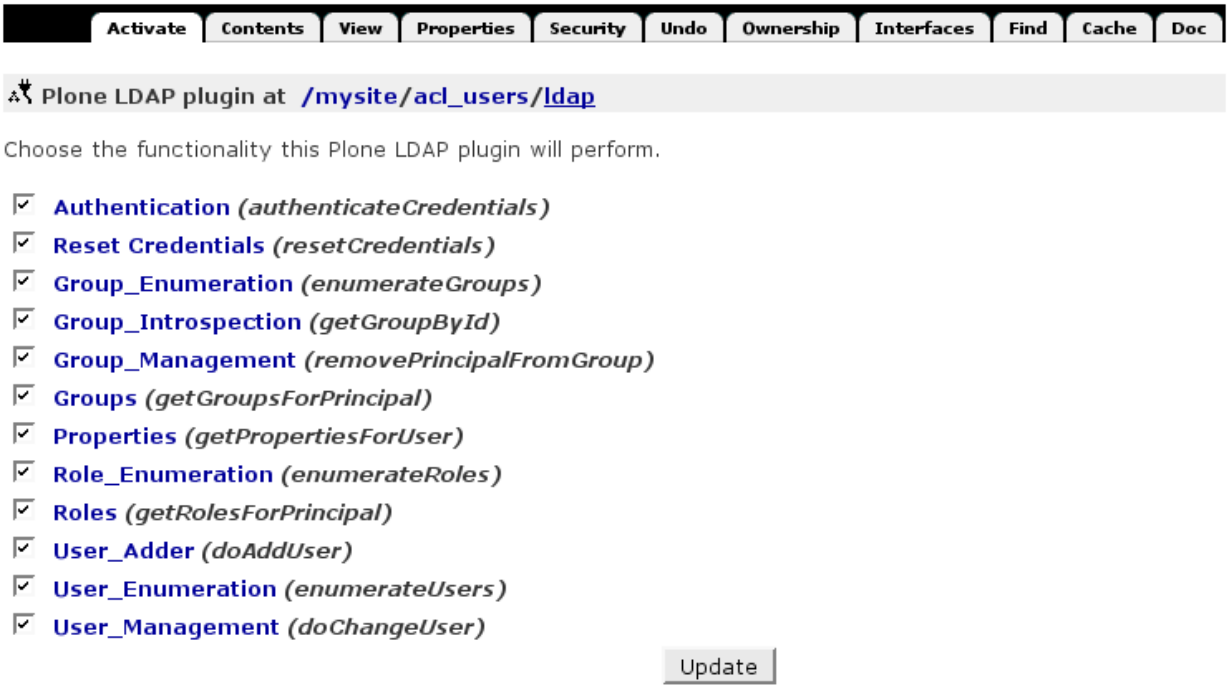
Im *LDAP Server*-Reiter können ein oder mehrere LDAP-Server angegeben werden. Sind mehrere Server konfiguriert werden sie von oben nach unten durchsucht.

Um nun zu testen, ob die Plone-Site mit dem LDAP-Server kommuniziert, klicken Sie auf *Benutzer* in der Plone-Konfiguration und lassen sich anschließend alle Nutzer anzeigen. Nun sollten Ihnen die Nutzer aus dem LDAP-Server korrekt angezeigt werden.

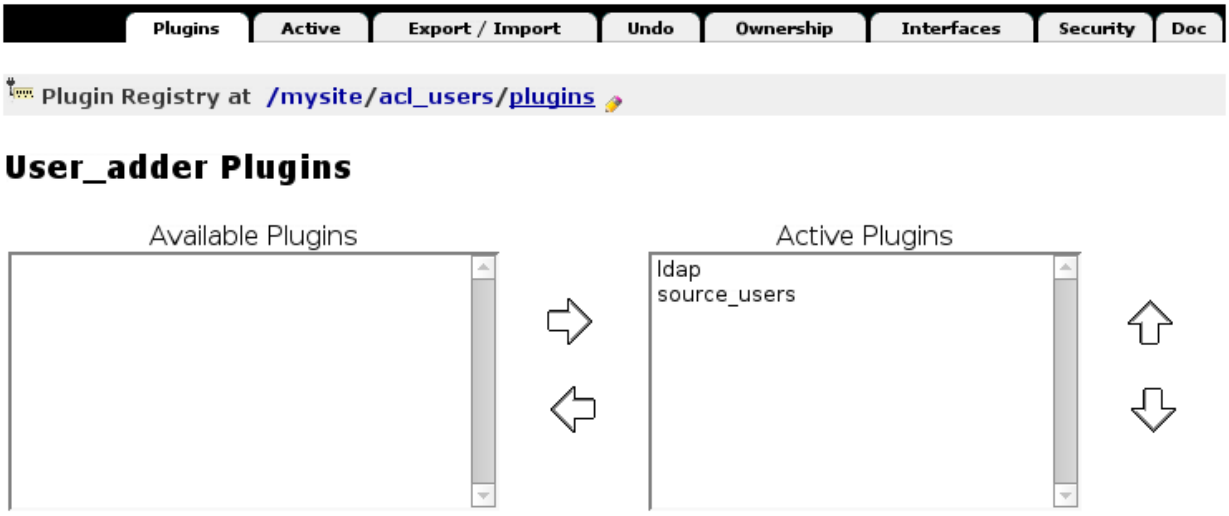
Anmerkung: Der Nutzername für die Zope-Instanz, z.B. `admin`, sollte nicht in Ihrem LDAP-Verzeichnis vorhanden sein.

Manuelle Konfiguration der PAS-Plugins

Die oben gezeigte LDAP-Konfiguration konfiguriert das `ldap`-Plugin in `acl_users`.



Hier lassen sich die einzelnen Funktionen der Nutzerverwaltung aktivieren bzw. deaktivieren. Beim Klicken in die Funktionen können diese auch konfiguriert werden. Schauen wir uns nun *User Adder* genauer an:



Auch hier werden die aktiven Plugins von oben nach unten durchsucht, d.h. die Nutzereigenschaften werden zunächst mit dem `ldap`-Plugin und dann erst mit dem `source_users`-Plugin gesucht.

Sollen in Plone angelegte Nutzer nicht in das LDAP-Repository übernommen werden, muss die Reihenfolge der beiden Plugins geändert werden.

Sollen neue Plone-Nutzer nicht in das LDAP-Repository übernommen werden, verschieben Sie das `ldap`-Plugin in die *Available Plugins*.

Alternative Ansätze

- Intranets with huge LDAPs
- chaoflow (Florian Friesdorf)

  Three tier architecture:

  1. ldapy

     Low-level Python-Bibliothek, die libldap via cffi undasynchrone Aufrufe via Generatoren unterstützt.

  2. ldapalchemy

     entsprechend sqlalchemy modelliert zur Unterstützung von Session-Management mit Connection-Pools und Abfrage der LDAP-Einträge mit Attributnamen und Typ-Mapping.

  3. pas.plugins.ldapalchemy

     PAS-Plugin, das unter Verwendung von ldapalchemy mit dem LDAP redet

Reproduzierbare LDAP- und AD-Konfigurationen

Mit vs.genericsetup.ldap haben wir ein Produkt entwickelt, das den Im- und Export der LDAP- oder AD-Konfiguration von Plone-Sites erlaubt.

`vs.genericsetup.ldap` nutzt das *Generic Setup Tool*, um die Konfiguration einer LDAP- oder Active Directory-Anbindung aus einer Plone-Site exportieren und in einer anderen Plone-Site wieder importieren zu können. Damit werden reproduzierbare und programmatische LDAP- und AD-Anbindungen möglich.

Installation

Um `vs.genericsetup.ldap` zu installieren, wird in der `buildout.cfg-Datei` folgendes eingetragen:

```
[buildout]
eggs =
    ...
    vs.genericsetup.ldap

[instance]
...
zcml =
    ...
    collective.genericsetup.ldap
```

Anwendung

Nach der Installation von `vs.genericsetup.ldap` können Sie in das Generic Setup Tool Ihrer Plone-Site gehen und die Konfiguration Ihrer LDAP- oder AD-Anbindung exportieren, indem Sie im *Export*-Reiter *LDAP Settings Export* auswählen und anschließend auf *Export selected steps* klicken. Anschließend können Sie die `ldap_plugin.xml`-Datei in `src/vs.policy/vs/policy/profile/default/` kopieren. Beim Installieren des `vs.policy`-Produkts wird das *Plone LDAP plugin* oder *ActiveDirectory Multi Plugin* automatisch im *Pluggable Auth Service* erstellt. Ist `vs.policy` bereits für Ihre Website installiert, können Sie eines der beiden Plugins auch nachträglich hinzufügen, indem Sie im *Import*-Reiter des *Generic Setup Tool* den *LDAP Settings Import* auswählen und anschließend auf *Import selected steps* klicken.

3.12.3 Membrane und Remember

Membrane Bibliothek, die es erlaubt, Nutzer und Gruppen als Archetypes-Inhaltstypen zu modellieren.

Remember Reihe von PAS-Plugins, die, basierend auf Membrane, das übliche Verhalten der Plone-Mitgliederverwaltung reimplementieren.

3.12.4 Single Sign On mit Kerberos

Um eine lauffähige Kerberos-Umgebung sowie eine Plone bzw. Zope-Website mit Single Sign On zu erhalten, sind Anpassungen auf mehreren Ebenen notwendig:

1. Einrichtung eines zentralen Kerberos-Servers (KDC, Key Distribution Center)
2. Einrichtung des Betriebssystems des vorgelagerten Webservers, sodass Kerberos-Tickets vom KDC bezogen werden können
3. Einrichtung des Betriebssystems des Website-Benutzers, sodass es beim Maschinen-Login Kerberos-Tickets vom KDC bezieht
4. Einrichtung des vorgelagerten Webservers, in diesem Falle Apache
5. Installation eines Zope-Zusatzproduktes, damit der User Folder die vom Webserver zusätzlich zur Verfügung gestellten Authentifizierungsdaten auswertet

Dieses Dokument befasst sich lediglich mit der serverseitigen Einrichtung. Das Betriebssystem des Benutzers der Website muss also bereits so konfiguriert sein, dass der Logon für den Benutzer bereits über Kerberos abgewickelt wird.

Unser Beispielsystem setzt auf einem Linux-Server auf, der als Kerberos-Server konfiguriert wird. Der Apache-Server läuft ebenfalls unter Linux und agiert als Kerberos-Client. Beim Anmeldevorgang wird automatisch ein Kerberos-Ticket bezogen, welches die spätere Abwicklung der Kerberos-Authentifizierung zur Website ermöglicht. Das Einrichten einer solchen Konstellation ist auf dem Internet vielfach beschrieben, siehe auch die Weiterführenden Links am Ende des Dokumentes.

3.12.5 Installation des `mod_auth_kerb`-Authentication-Modul

Für Apache wird das Zusatzmodul `mod_auth_kerb` benötigt, welches unter Linux mit den Bordmitteln des Betriebssystems installiert werden kann, wie z.B. `apt-get` unter Debian und Ubuntu oder `yum` unter RedHat-Systemen:

```
# apt-get install libapache2-mod-auth-kerb
```

oder:

```
# yum install mod_auth_kerb
```

Falls das Modul anschließend nicht automatisch geladen wird, kann dies manuelle geschehen mit:

```
# LoadModule auth_kerb_module /usr/lib/apache2/modules/mod_auth_kerb.so
```

Je nach OS muss der Pfadname des Moduls noch angepasst werden.

Da die Header der HTTP-Anfrage von Apache modifiziert werden, muss auch `mod_headers` installiert sein. In unserer Beispielkonfiguration wird Apache als Proxy für Plone verwendet, womit auch `mod_rewrite` und `mod_proxy` aktiv sein müssen.

3.12.6 Erstellen eines *Service Principal*

Der *Service Principal* kann erstellt werden mit folgendem kadmin-Befehl:

```
# kadmin -p bofh/admin -q "addprinc -randkey HTTP/www.example.com"
```

3.12.7 Erstellen einer Schlüsseltabelle (`keytab`)

Ein keytab ist eine Datei zum Speichern der Schlüssel für einen oder mehrere Kerberos-Prinzipals. `mod_auth_kerb` benötigt diese Tabelle um den oben erstellten Service Principal nutzen zu können.

1. Die Schlüsseltabelle kann mit `kadmin` erstellt werden:

   ```
   # kadmin -p bofh / admin -q "ktadd -k /etc/apache2/http.keytab HTTP / www.example.
   ↪com"
   ```

   oder auf RedHat-basierten Systemen im Pfad `/etc/httpd/http.keytab`.

   Die `-k`-Option spezifiziert den Pfadnamen der `keytab`-Datei, die erstellt wird sofern sie noch nicht existiert.

   Anschließend muss der Eigentümer so geändert werden, dass der Apache-Prozess darauf zugreifen kann:

   ```
   # chown www-data /etc/apache2/http.keytab
   ```

   oder auf RedHat-basierten Systemen:

   ```
   # chown apache /etc/httpd/http.keytab
   ```

   Um zu überprüfen, ob der Schlüssel korrekt in die `keytab` eingetragen wurde, sollten wir uns als *Service Principal* authentifizieren und uns anschließend das resultierende Granting-Ticket mit `klist` anzeigen lassen:

   ```
   # kinit -k -t /etc/apache2/http.keytab HTTP/www.example.com
   # klist
   ```

3.12.8 Konfigurieren des Apache-Webservers

In der unten gezeigten Apache-Konfiguration lauscht die Zope-Instanz auf IP `10.0.0.2` am Port `8080`, und Apache auf `10.0.0.1` am Port `80`. Das Plone-Portal befindet sich in der Zope-Instanz unter dem Pfad `/portal`. In Kerberos wird der Realm-Wert `MYDOMAIN` verwendet:

```
<VirtualHost 10.0.0.1:80>
    ServerAdmin webmaster@mydomain.com
    ServerName intranet.mydomain.com

    <Location />
        AuthName "Intranet"
        AuthType  Kerberos
        KrbAuthoritative on
        KrbAuthRealms  MYDOMAIN
        KrbServiceName HTTP
        Krb5Keytab /etc/krb5.keytab
        KrbMethodNegotiate on
        KrbMethodK5Passwd off
        KrbSaveCredentials on
        require valid-user
        RequestHeader set X_REMOTE_USER %{remoteUser}e
    </Location>

    RewriteEngine On
    RewriteRule ^/(.*) http://10.0.0.2:8080/VirtualHostBase/http/intranet.mydomain.
↪com:80/portal/VirtualHostRoot/$1 [L,P,E=remoteUser:%{LA-U:REMOTE_USER}]
</VirtualHost>
```

Um diese Konfiguration nun verwenden zu können, muss der Apache-Webserver neu gestartet werden mit:

```
# service apache2 force-reload
```

Wie man erkennen kann, reicht das Einfügen der `mod_auth_kerb`-Direktiven in eine `Location`-Direktive. Kerberos wird als Authentifizierungsmechanismus festgelegt, und es wird eine positive Identifikation des Benutzers zum Zugriff erfordert (require valid-user). Wichtig hierbei ist das Abschalten von `KrbMethodK5Passwd`, um eine Abfrage und Übertragung des Kerberos-Logins zwischen Browser und Apache zu verhindern. Es wird ausschliesslich die Negotiate-Methode zugelassen (`KrbMethodNegotiate`), bei der keine Logins über das Netz geschickt werden, sondern nur Kerberos-Ticket-Informationen.

Die Plone-Instanz selber muss kein Kerberos verstehen. Wie man in der Apache-Konfiguration ersehen kann, wird der ermittelte Benutzername in einen zusätzlichen HTTP-Header `X_REMOTE_USER` geschrieben und so weitergeleitet.

Beim Einsatz von `mod_auth_kerb` in Apache muss man beachten, dass man Kerberos-Authentifizierung nicht mit anderen Authentifizierungen kombinieren kann. Es ist nicht möglich, bei erfolgloser Kerberos-Authentifizierung auf z.B. normale Basic Auth zurückzufallen. Ferner ist es nicht möglich, diese Authentifizierung optional zu gestalten, sodass auch bei erfolglosem Kerberos- Versuch der Besuch der Website gestattet wird. Das heisst, man kann auf einem für Kerberos-Authentifizierung eingerichteten Hostnamen keine Besucher bedienen, die anonym durchgelassen werden sollen oder die auf andere Weise authentifiziert werden können.

3.12.9 Plone-Konfiguration

1. Auf der Plone-Seite reicht die Installation und Konfiguration eines Zusatzproduktes, welches den von Apache gesetzten zusätzlichen HTTP-Header versteht und auswertet. Für unser Beispiel benutzen wir Products.WebserverAuth (siehe auch Weiterführende Links unten). Das Produkt kann als Python Egg einfach in einen bestehenden Plone-Buildout eingebunden werden:

```
[instance]
...
eggs =
    ...
    Products.WebServerAuth
```

2. Nachdem das Buildout-Skript durchlaufen und die Instanz neu gestartet wurde, sollte in portal-url → site setup → Add-on Products *WebServerAuth* aktiviert werden können.

3. Damit wird im `PluggableAuthService` des Portals das `WebServerAuth`-Plugin zur Verfügung gestellt.

   Von der Standardkonfiguration auf dem Reiter *Options* wurde nur in einem Punkt abgewichen: Wir haben die Option `"Only users with a pre-existing Plone account"` gewählt, um nur solche Kerberos-Benutzer durchzulassen, die auch in der Plone-Instanz bekannt sind.

   Zudem muss das neue Plugin in unserer Beispielkonfiguration nur für zwei Dienste aktiviert werden, nämlich für `Authentication` und `Extraction`.

   `Extraction` ist für das Ermitteln von Login-Daten aus der hereinkommenden HTTP-Anfrage zuständig. Da jeder Zugriff über Apache automatisch den vom neuen Plugin ausgewerteten HTTP-Header enthält und die Verarbeitung dieses Headers schnell und einfach ist, setzen wir das neue Plugin als erstes aktives Extraction-Plugin ein.

   `Authentication` nimmt die im ersten Schritt ermittelten Login-Daten und prüft, ob ein Benutzer mit diesen Login-Daten bekannt ist und angemeldet werden kann. Da in unserem Beispielszenario die Benutzer in ActiveDirectory vorgehalten werden und auch Plone dort die Benutzerdaten sucht, ist das neue Plugin als letztes aktives Authentication-Plugin geführt. Somit wird am bisherigen Verfahren vor dem Einsatz von Kerberos am wenigsten geändert.

4. Der Abmelden-Link des Plone-Portal (ZMI → Plone-Portal → portal_actions → Benutzer → Logout) sollte auf eine spezielle Logout-Seite umleiten, die z.B. den folgenden Inhalt trägt:

   »Sorry, Sie müssen Ihren Web-Browser schließen um sich von diesem Portal abzumelden.«

   Hierzu können Sie auch das `logged_out`-Template entsprechend anpassen.

5. Das Login-Portlet sollte nicht angezeigt werden.

6. Der *Password ändern*-Link (z.B. in ZMI → Plone-Portal → portal_controlpanel) sollte ebenfalls nicht mehr angezeigt werden.

3.12.10 Zusammenfassung

Nach den oben genannten Konfigurationsschritten sollte ein korrekt auf Windows angemeldeter und in Plone bekannter Benutzer bei Besuch der Plone-Instanz sofort und ohne Umweg über die Login-Maske angemeldet sein. Das ist schnell erkennbar daran, dass z.B. kein Login-Link mehr angeboten wird, wohl aber eines auf die eigenen Inhalte und Präferenzen.

Bei Problemen ist besonders das Dokument Using mod_auth_kern and Windows 2000/2003 as KDC hilfreich. Es erklärt in kleinen Schritten, wie die Konfiguration geprüft und Fehler ausgemerzt werden können.

Auf der Plone-Seite kann man die korrekte Weitergabe der Login-Informationen sehr einfach mit einer simplen DTML-Methode testen, die die Werte des `REQUEST` ausgibt:

```
<dtml-var REQUEST>
```

Dort muss im Bereich environ ein Header namens `HTTP_X_REMOTE_USER` sichtbar sein, der den vollen Kerberos-Login-Namen des Benutzers enthält. Ist er es nicht, wurde der Login nicht korrekt von Apache weitergegeben.

Siehe auch:

- Single Sign On with Active Directory
- How to set up a Kerberos Server under Linux
- Step by step guide to Kerberos 5 interoperability
- mod_auth_kerb Dokumentation

3.13 Migrationen

3.13.1 Vorbereitungen

Bevor Sie eine Plone-Site aktualisieren, sollten Sie verschiedenes vorbereiten.

Sammeln Sie die notwendigen Informationen zusammen

1. Lesen Sie *What's new in...* und die *release notes* der für Sie relevanten Plone-Version. Sie finden diese im CMFPlone-Ordner Ihrer Distribution.
2. Überprüfen Sie die Abhängigkeiten
    1. Die Abhängigkeiten werden in den *release notes* angegeben:
        - Welche Python-Version?
        - Welche Zope-Version?
        - Welche Python-Bibliotheken?
    2. Achten Sie darauf, dass alle der von Ihnen verwendeten Zusatzprodukte die neue Plone-Version unterstützen.

        Einen Überblick über alle Artikel einer Plone 2.5- oder 3.x-Site erhalten Sie mit mr.inquisition. Dabei erhalten Sie Informationen

        - zur Art der Artikeltypen
        - zur Anzahl der Artikel je Artikeltyp
        - zum Ort der Artikel
        - zu Artikeln, die mit deinstallierten Produkten erstellt wurden

        Installiert werden kann `mr.inquisition` mit:

```
[buildout]
...
eggs =
    ...
    mr.inquisition

[instance]
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
...
zcml =
    mr.inquisition
```

Nach dem Durchlaufen des Buildout-Skripts und dem Starten der Instanz erhalten Sie z.B. unter `http://localhost:8080/Plone/@@inquisition` einen Überblick über die verfügbaren Ansichten zur Analyse der ZODB.

3. Falls die neue Version von Plone auf einer aktuelleren Zope-Version basiert, sollten Sie diese vor dem Plone-Update installieren.
   - **Achtung:** Zope hat seine eigenen Migrationsrichtlinien, die in den *release notes* derjenigen Zope-Version zu finden sind, auf die Sie migrieren möchten. Im allgemeinen aktualisiert Plone jedoch mit seinen Migrationsskripten auch die Zope-Version.
4. Lesen Sie die folgenden Dateien in Ihrem CMFPlone-Ordner:
   - `README.txt`
   - `INSTALL.txt`
   - `UPGRADE.txt`

   Diese Dateien können wichtige *last minute* Informationen und spezifische Anweisungen enthalten.

Plone-Upgrade

Ein schneller Überblick, wie für eine Plone-Site ein Upgrade durchgeführt werden kann.

1. Erstellen Sie eine neue Zope-Instanz mit *Buildout*.
2. Geben Sie im `[productdistros]`-Abschnitt Ihrer `buildout.cfg`-Datei alle erforderlichen Zusatzprodukte an.
3. Ihre eigenen aktualisierten Produkte können Sie in das `Products`-Verzeichnis Ihres Buildout-Projekts kopieren.
4. Fahren Sie Ihre alte Zope-Instanz herunter.
5. Kopieren Sie die `var/Data.fs`-Datei aus Ihrer alten Instanz in das `var/filestorage`-Verzeichnis Ihres Buildout-Projekts.
6. Konfigurieren Sie Ihre neue Zope-Instanz, s.a. *Buildout-Konfiguration*.
7. Starten Sie Ihre neue Zope-Instanz.
8. Gehen Sie in das Zope Management Interface (ZMI) Ihrer Plone-Site und anschließend zum *Plone Migrations Tool* (`portal_migrations`).
9. Nachdem Sie den *Upgrade*-Reiter gewählt haben, erhalten Sie eine Angabe wie diese:

```
Instance version: 4.3.18
File system version: 5.2
```

10. Klicken Sie die *Upgrade*-Taste.

Aktualisieren von Plone 4.1 zu 4.2

Aktualisieren von `zc.buildout`

Üblicherweise verwendet `bootstrap.py` nun `zc.buildout` in Version 1.5.2 und nicht mehr 1.4.4. Die aktuelle Version von `zc.buildout` referenziert jedoch nicht mehr die globalen Python Site Packages, weshalb eigene Pakete nun in das lokale Build übernommen werden müssen.

In einem bestehenden Buildout-Projekt können Sie `zc.buildout` aktualisieren mit:

```
$ python2.7 bootstrap --version 1.5.2
```

Search-Template

Die Suche wurde in Plone 4.2 grundlegend überarbeitet, sodass frühere Anpassungen der Suche voraussichtlich nicht mehr funktionieren werden.

1. Um nun Ihre Änderungen für Plone 4.2 zu übernehmen, verwenden Sie am besten z3c.jbot.
2. Anschließend kopieren Sie `plone.app.search/plone/app/search/search.pt` in Ihr Produkt, z.B. nach `vs.theme/vs/theme/overrides/plone.app.search.search.pt`.
3. Nun starten Sie Ihre Instanz um zu überprüfen, ob dies fehlerfrei geschieht.
4. Schließlich können Sie Ihre gewünschten Änderungen am Suchformular vornehmen.

Aktualisieren bestehender Kollektionen

Upgrade

Beim Upgrade der Plone-Site bleiben die alten Kollektionen weiterhin erhalten als *Collection (old-style)*; sie werden nicht migriert.

Anpassungen für alte und neue Kollektionen

Die neuen Kollektionen implementieren die `queryCatalog`-Methode in `plone.app.collection.collection`, sodass in vielen Fällen voraussichtlich nur das Interface und die Referenzen auf den `portal_type` geändert werden müssen.

Um diese Änderungen rückwärtskompatibel implementieren zu können, empfiehlt sich, diese Methode nur in Views zu verwenden, die ausschließlich den neuen Kollektionen vorbehalten sind. Dies lässt sich mit Conditionally run ZCML realisieren, also z.B. mit:

```
<browser:page
  zcml:condition="installed plone.app.collection"
  name="mycollectionview"
  for="plone.app.collection.interfaces.ICollection"
  class=".views.MyCollectionView"
  permission="zope2.View"/>
```

Analog lassen sich auch Interfaces nur für neue Kollektionen registrieren, z.B.:

```
<class class="plone.app.collection.collection.Collection"
 zcml:condition="installed plone.app.collection">
 <implements interface=".interfaces.IMyCollectionInterface" />
</class>
```

`getRawQuery`-Methode

Die `getRawQuery`-Methode können Sie selbst verwenden mit:

```
from plone.app.querystring import queryparser
query = queryparser.parseFormquery(collectionobj, collectionobj.getRawQuery())
```

Aktualisieren von Plone 4.2 zu 4.3

Die Abhängigkeiten von Plone 4.3 sind deutlich verringert worden. Dadurch wird der Speicherverbrauch von Plone verringert und Importe beschleunigt. Es kann jedoch sein, dass Plone-Zusatzprodukte entsprechend angepasst werden müssen.

- Importierte `zope.*`-Pakete haben in neueren Versionen weniger Abhängigkeiten
- Zusätzliche Abhängigkeiten können in der `setup.py`-Datei angegeben werden.

  Weitere Informationen hierzu erhalten Sie in *Upgrade von Zusatzprodukten*.

Im Folgenden eine Liste derjenigen Zusatzprodukte, die zwar in Plone 4.2, nicht jedoch in Plone 4.3 enthalten sind:

- elementtree

  stattdessen wird nun lxml verwendet
- Products.kupu
- plone.app.kss

  kann z.B. für das *Inline Editing* zusätzlich installiert werden.
- zope.app.cache
- zope.app.component
- zope.app.container
- zope.app.pagetemplate
- zope.app.publisher
- zope.copypastemove
- zope.dublincore
- zope.hookable

Einen Überblick über die gebräuchlichsten Importe und deren neue Orte erhalten Sie in Updating package dependencies.

Python-3-Migration der ZODB

ZODB selbst ist mit Python 3 kompatibel, eine in Python 2.7 erstellte Datenbank kann jedoch nicht ohne vorherige Migration in Python 3 verwendet werden. Hierfür müssen folgende Schritte ausgeführt werden:

1. Aktualisieren der Site auf Plone 5.2 mit Python 2, s.a. Upgrading Plone 5.1 to 5.2.
2. Stellt sicher, dass der Code aller von euch verwendeten Add-Ons in Python 3 funktioniert, s.a. *Python 3-Migration*.
3. *ZODB packen*.
4. *Backup der ZODB*.
5. Im Buildout mit `py2env` und zodbverify überprüfen, ob die Integrität eurer Datenbank gewährleistet ist:

   1. Installation von `zodbverify`. Hierfür wird zunächst die `devel.cfg` geändert und anschließend das `bin/buildout` aufgerufen:

      ```
      [instance]
      eggs +=
          zodbverify
      ...
      ```

   2. Anschließend kann die Datenbank überprüft werden mit:

      ```
      $ bin/zodbupdate --convert-py3 --file=var/filestorage/Data.fs --encoding utf8
      ...
      Updating magic marker for var/filestorage/Data.fs
      Ignoring index for /Users/pbauer/workspace/projectx/var/filestorage/Data.fs
      Loaded 2 decode rules from AccessControl:decodes
      Loaded 12 decode rules from OFS:decodes
      Loaded 2 decode rules from Products.PythonScripts:decodes
      Loaded 1 decode rules from Products.ZopeVersionControl:decodes
      Committing changes (#1).
      ```

      Zusätzlich können noch weitere Optionen angegeben werden, z.B.:

      `-D` falls defekte *Pickles* entdekckt werden, können diese direkt debugged werden.

      `--encoding-fallback` falls `UnicodeDecodeError` auftreten, ist die Instanz vermutlich nicht einheitlich `utf-8` kodiert. Hier empfiehlt es sich, als Fallback `latin1` zu verwenden, da dies die frühere Standardkodierung von Zope war, also `--encoding-fallback latin1`.

   Falls Integritätsprobleme, auftreten, müssen diese vor der Migration gelöst werden. Weitere typische Probleme sind:

   1. defekter `Data.fs.index`, z.B.:

      ```
      $ ./bin/zodbupdate --convert-py3 --file=var/filestorage/Data.fs --
      ↪encoding=utf8
      Updating magic marker for var/filestorage/Data.fs
      loading index
      Traceback (most recent call last):
        File "/home/erral/downloads/eggs/ZODB-5.5.1-py3.6.egg/ZODB/FileStorage/
      ↪FileStorage.py", line 465, in _restore_index
          info = fsIndex.load(index_name)
        File "/home/erral/downloads/eggs/ZODB-5.5.1-py3.6.egg/ZODB/fsIndex.py",␣
      ↪line 134, in load
          v = unpickler.load()
      UnicodeDecodeError: 'ascii' codec can't decode byte 0x80 in position 249:␣
      ↪ordinal not in range(128)
      ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

Diesen Fehler solltet ihr beheben können indem ihr vor der Migration die Datei `Data.fs.index` löscht.

2. fehlerhafte Suche

   In diesem Fall solltet ihr den Katalog neu erstellen. Hierzu geht ihr in eurer Plone-Site in den `portal_catalog`, wählt den *Advanced*-Reiter und klickt dann auf *Clear and Rebuild*.

6. Kopiert die Datenbank nun in ein Buildout mit `py3env`, startet jedoch **nicht** die Instanz.
7. Migriert die Datenbank mit zodbupdate.
   1. Zunächst wird `zodbupdate` mit Buildout installiert:

```
[buildout]

parts =+
    zodbupdate

[zodbupdate]
recipe = zc.recipe.egg
eggs =
    zodbupdate
    ${buildout:eggs}
```

   2. Anschließend kann die ZODB aktualisiert werden mit:

```
$ bin/zodbupdate -f var/filestorage/Data.fs
```

8. Überprüft die Integrität eurer Datenbank mit `zodbverify`. Wenn Probleme auftreten, behebt diese und wiederholt die Migration.
9. Startet die Instanz und überprüft manuell ob alles wie erwartet funktioniert.

Siehe auch

- Migrating the ZODB

Artikeltypen migrieren

ATContentTypes Migration Framework

Das ATContentTypes-Produkt wird mit einem eigenen Migration Framework ausgeliefert.

ATContentTypes verwendet eine *Registry* für Migrationen, die im ATCT Tool (`portal_atct`) verwaltet werden. Dies ermöglicht, Migrationen in einem einfach zu bedienenden Web-Interface durchzuführen. In vielen Fällen reicht jedoch eine manuell erstellte *External Method* vollkommen aus, da Migrationen meist nur einmal aufgerufen werden müssen.

Beispiel

Zu einem früheren Zeitpunkt wurde ein CMF-Inhaltstyp *PhotoAlbum* erstellt und dieser soll nun in einen einfachen Archetypes *Folder* zurückverwandelt werden. Hierzu wird, sofern noch nicht vorhanden, in unserem Produkt `src/vs.photo/vs/photo/upgrades.py` und folgendem Inhalt angelegt:

```
from Products.CMFCore.utils import getToolByName
from StringIO import StringIO
from Products.ATContentTypes.migration.walker import CatalogWalker
from Products.ATContentTypes.migration.migrator import CMFFolderMigrator

class  PhotoAlbumMigrator(CMFFolderMigrator):
    """Base class to migrate PhotoAlbum to Folder.
    """
    walkerClass = CatalogWalker
    src_meta_type = 'PhotoAlbum'
    src_portal_type = 'PhotoAlbum'
    dst_meta_type = 'ATFolder'
    dst_portal_type = 'Folder'

 def migrate(self):
     """Run the migration"""

     out = StringIO()
     print >> out, "Starting migration"

     portal_url = getToolByName(self, 'portal_url')
     portal = portal_url.getPortalObject()

     migrators = (PhotoAlbumMigrator,)

     for migrator in migrators:
         walker = migrator.walkerClass(portal, migrator)
         walker.go(out=out)
         print >> out, walker.getOutput()

     print >> out, "Migration finished"
     return out.getvalue()
```

Nun muss nur noch im Wurzelverzeichnis der Site eine neue *External Method* erstellt werden mit

ID `migrateTypes`

Module vs.photo.migrate

function name migrate

Um die externe Methode auszuführen, müssen Sie nur noch in den «Test»-Reiter klicken[1].

[1] **Warnung:** Migrationen können normalerweise nur selten rückgängig gemacht werden da sie meist mehrere Transaktionen zugleich umfassen. Daher sollten Sie unbedingt vor der Migration eine Sicherungskopie Ihrer Plone-Site erstellen.

Konzept

In dem oben genannten Beispiel wird in der `migrate()`-Funktion ein *Migrator* auf alle mit einem *Walker* gefundenen Objekte angewendet.

Walker

Sie werden zum Finden der zu migrierenden Inhalte verwendet. Der einfachste *Migrator* ist der *CatalogWalker* in walker.py, der eine Katalogabfrage für alle Inhalte eines bestimmten Typs durchführt.

Migrator

Sie sind einfache Klassen, die die gegebenen Inhaltstypen migrieren. Das Framework enthält Basisklassen, die das Schreiben von *Migrators* stark vereinfachen.

Ein *Migrator* ist meist eine von `CMFFolderMigrator` oder `CMFItemMigrator` abgeleitete Klasse. Diese beiden und weitere Basis-Migrationsklassen sind in migrator.py definiert:

`CMFItemMigrator` migriert einen CMF-Typ einschließlich seiner Meta-Angaben, lokalen Rollen etc.

`CMFFolderMigrator` gewährleistet darüberhinaus, dass auch die Inhalte migriert werden.

Es gibt drei Arten von Migrationen:

1. Jede Methode in einer Klasse, die mit `migrate_` beginnt, wird automatisch aufgerufen.

   Wenn Sie sich z.B. die `BaseMigrator`-Klasse anschauen, sehen Sie eine Reihe solcher Methoden, `migrate_properties`, `migrate_owner`, etc.

   Auch die Reihenfolge, in der die Methoden aufgerufen werden, sind durch Präfixe definiert:

   `beforeChange_` Methoden, wie z.B. `beforeChange_storeDates` oder `beforeChange_storeSubojects`, die vor der Migration angewandt werden.

   `last_migrate_` Methoden, wie `last_migrate_date`, die aufgerufen werden, bevor der *Migrator* die Migration eines Objekts beendet.

2. Die Methode `custom()` wird nach den `migrate_`- aber vor den `last_migrate_`-Methoden aufgerufen. Die Standardimplementierung ist leer und dient nur dazu,von einer eigenen Migrationsmethode überschrieben zu werden. Hier ein Beispiel aus atctmigrator.py:

```
class FileMigrator(CMFItemMigrator):
    walkerClass = CatalogWalker
    ...

    def custom(self):
        ctype = self.old.getContentType()
        file = str(self.old)
        self.new.setFile(file, mimetype = ctype)
```

3. Schließlich noch die einfachste Methode mit der `map class`-Variablen, die eine Zuordnung von Attributen und/oder Methoden erlaubt. Auch hier wieder ein Beispiel aus atctmigrator.py:

```
class LinkMigrator(CMFItemMigrator):
    walkerClass = CatalogWalker
    map = {'remote_url' : 'setRemoteUrl'}
```

contentmigration

Das contentmigration-Produkt wird für Migrationen verschiedener Versionen desselben Typs genutzt.

Häufiger als die Migration von einem Inhaltstyp zu einem anderen ist die Änderung der internen Organisation eines Inhaltstyps. Das Archetypes-Tool bietet einige Hilfsmittel zur Durchführung von solchen Migrationen über den *Update Schema*-Reiter. Diese synchronisieren die ZODB-Schemata Ihrer Inhaltstypen mit dem auf Dateisystem-Ebene definierten Inhaltstyp. Beachten Sie jedoch, dass Sie dabei Daten verlieren können. Wenn Sie z.B. ein Feld umbenannt haben, führt *Update Schema* für bestehende Objekte zu leeren oder Standardwerten für das neue Feld. Um die alten Daten zu erhalten, ist eine Migration nötig.

Für solche Fälle wurde das contentmigration-Produkt geschreiben. Sie können *contentmigration* installieren, indem Sie in `buildout.cfg` folgendes eintragen:

```
[buildout]
...
eggs =
    ...
    Products.contentmigration
```

Beispiel

Anschließend kann es von Ihren eigenen Migrationsmethoden aufgerufen werden, z.B.:

```
from Products.contentmigration.walker import CustomQueryWalker
from Products.contentmigration.migrator import FieldActionMigrator
from Products.Archetypes.public import *

class MyMigrator(FieldActionMigrator):
    src_portal_type = 'MyType'
    src_meta_type = 'MyType'

    fieldActions = ({'fieldName'    : 'someField',
                     'storage'      : AttributeStorage(),
                     'newFieldName' : 'renamedField',
                     'newStorage'   : AnnotationStorage(),
                     'transform'    : lambda obj, val, **kw: val + 10,
                     })
```

Methoden

contentmigration erweitert die ATContentTypes-Migrator um folgende Methoden:

CustomQueryWalker

Der `CustomQueryWalker` spezifiziert eine Katalogabfrage, z.B. kann die Migration auf Teilinhalte Ihrer Site beschränkt werden mit:

```
walker = CustomQueryWalker(portal, migrator,
                           query = {'path' : '/some/path'})
walker.go()
```

Gegebenenfalls können `src_portal_type` und `src_meta_type` aus dem Migrator in die Anfrage eingefügt werden.

Der `CustomQueryWalker` kann von jedem *Migrator* genutzt werden.

BaseInlineMigrator

Der `BaseInlineMigrator` ist dem `BaseMigrator` ähnlich; während jedoch der `BaseMigrator` ein altes Objekt temporär kopiert, ein neues Objekt erstellt und dann die Migrationsmethoden anwendet, werden beim `BaseInlineMigrator` die Migrationsmethoden auf der Stelle angewendet. Dies vereinfacht den Code deutlich, da Attribute, lokale Rollen etc. nicht kopiert werden müssen. Er eignet sich daher für Migrationen, in denen sich die Felder bestimmter Objekte ändern.

Folgende Methoden können verwendet werden: `migrate_`, `beforeChange_` `last_migrate_` und `custom()`, nicht jedoch das `map class`-Attribut.

Achtung: Sie können nur `self.obj` verwenden, nicht `self.old` und `self.new`.

Auch `BaseInlineMigrator` kann von jedem *Walker* verwendet werden.

FieldActionMigrator

Der `FieldActionMigrator` ist eine Erweiterung von `BaseInlineMigrator` um Aktionen, die auf Felder angewendet werden können. Dafür wird eine Liste der zu migrierenden Attribute und der auszuführenden Aktion angegeben. Detaillierte Informationen zu diesen Aktionen erhält man direkt in der field.py-Datei. Beispiele finden sich in testATFieldMigration.py.

Migration von Archetypes zu Dexterity

plone.app.contenttypes kann alle Standard-Plone-Artikeltypen zu Dexterity migrieren. Hierzu gehören auch *Topics*, die früheren Kollektionen. Die Migration schließt auch die meisten Features ein, wie Portlets, Kommentare, Contentrules, lokale Rollen und lokale Workflows.

> **Warnung:** Frühere Versionen der Inhalte bleiben bei der Migration nicht erhalten.

Migration einzelner Artikeltypen

Mit `@@pac_installer` gibt es eine Ansicht, die die Installation von `plone.app.contenttypes` erlaubt ohne dass die alten Archetypes-basierten Artikeltypen ersetzt werden. Anschließend wird auf das Migrationsformular weitergeleitet und die zu migrierenden Artikeltypen können ausgewählt werden. Dies erlaubt ihnen, nur bestimmte Archetypes-Artikeltypen zu migrieren.

Migrieren von *Topics*

Topics unterstützten sog. *Subtopics*, ein Feature, das in Kollektionen nicht mehr exisitert. Daher müssen Kollektionen zunächst *folderish* werden bevor eine Migration von Subtopics durchgeführt werden kann. Hierfür muss die Basisklasse von `plone.dexterity.content.Container` abgeleitet werden und nicht von `plone.dexterity.content.Item`:

```
from plone.app.contenttypes.behaviors.collection import ICollection
from plone.dexterity.content import Container
from zope.interface import implementer

@implementer(ICollection)
class FolderishCollection(Container):
    pass
```

Falls die bestehende Kollektion überschrieben werden soll, kann die `Collection.xml` verwendet werden:

```
<?xml version="1.0"?>
<object name="Collection" meta_type="Dexterity FTI">
    <property name="klass">my.package.content.FolderishCollection</property>
</object>
```

Und falls die Suche der übergeordneten Kollektion vererbt werden soll, sind die Änderungen aus acquire query erforderlich.

Migrieren von Inhalten, die mit LinguaPlone übersetzt wurden

Da LinguaPlone die Dexterity-Artikeltypen nicht unterstützt, muss zunächst von LinguaPlone zu plone.app.multilingual migriert werden. Weitere Hinweise finden Sie in LinguaPlone-Migration.

Migrieren von `collective.contentleadimage`

collective.contentleadimage erweiterte die Standard-Plone-Artikeltypen um ein Bild. Dieses Bild bleibt bei der Migration erhalten sofern der zugehörige Dexterity-Artikeltyp das *Behavior* «Lead Image» hat. Dabei informiert zunächst das Navigationsformular mit dem Kommentar *extended fields: 'leadImage', 'leadImage_caption'* und auch das Migrationsformular zeigt für jeden Dexterity-Typ an, ob er das *Behavior* hat.

Migrieren eigener Artikeltypen

Während der Migration der Standard-Artikeltypen werden die eigenen Artikeltypen nicht migriert. `plone.app.contenttypes` enthält jedoch auch ein Migrationsformular für solche Artikeltypen: `@@custom_migration`. Dabei muss der Dexterity-Artikeltyp, zu dem migriert werden soll existieren und die Klasse des alten Artikeltyps noch existieren. Der alte Artikeltyp muss hingegen nicht in `portal_types` registriert sein – er kann dort auch bereits durch den Dexterity-Typ ersetzt worden sein.

Im View `@@custom_migration` kann für jeden Archetypes- der entsprechende Dexterity-Typ ausgewählt werden. Anschließend können die Felder aufeinander abgebildet werden oder auch Felder ignoriert werden.

Anschließend wird die Konfiguration überprüft indem ein migrierter Artikel aufgerufen wird: Geschieht dies fehlerfrei, ist der Test bestanden. Anschließend werden die Änderungen wieder zurückgerollt.

Upgrade von Zusatzprodukten

Häufig sind mit einer Aktualisierung von Plone auch die verwendetenZusatzprodukte zu aktualisieren. Dies ist nur eine generelle Anleitung, die keine Rücksicht auf produktspezifische Upgrade-Prozeduren nimmt.

1. Haben Sie weitere Produkte in Ihrer alten Plone-Site verwendet, müssen für diese ebenfalls aktuelle Versionen in der neuen Zope-Instanz installiert werden.
2. Starten Sie Ihre neue Zope-Instanz und überprüfen anschließend, ob im *Product Management*-Ordner des Zope Management Interface (ZMI) (`INSTANCE_URL/Control_Panel/Products/manage_main`) alle Produkte korrekt installiert wurden.
3. Navigieren Sie anschließend im ZMI zu Ihrer Site und dort in das *Plone QuickInstaller Tool* (`portal_quickinstaller`). Führen Sie ein Upgrade oder eine Neuinstallation der jeweiligen Produkte durch.

Migration eines Produkts zu Plone 4.0

Für die Migration von Plone3 auf Plone4 sollten die folgenden Änderungen beachtet werden.

Globale Definitionen

Viele globale Definitionen aus `global_defines.pt` werden nicht mehr vollständig im `main_template.pt` eingebunden, da dies die Performance deutlich beeinträchtigt. Einige globale Variablen wie z.B. `context`, `view` und `template` bleiben jedoch erhalten. Nicht erhalten geblieben sind jedoch die Definitionen aus der `_initializeData`-Klasse im `@@plone`-View von Plone3: Products/CMFPlone/browser/ploneview.py

Es empfiehlt sich daher, alle Seiten Ihres Produkts sich in einer Plone4-Site anzuschauen und dann ggf. die globalen Definitionen selbst einzubinden, also z.B.:

```
tal:attributes="action string:$here_url/${template/getId}"
```

ersetzen durch:

```
tal:attributes="action string:${context/@@plone_context_state/object_url}/${template/
↪getId}"
```

Achtung: Sie erhalten keine Fehlermeldung, wenn Sie globale Definitionen in der Überprüfung von `exists` verwenden z.B.::
tal:condition="python:exists(‚portal/mystyle.css‘)"

Diese Bedingung führt zu keinem Fehler, sondern die Überprüfung schlägt auch fehl, wenn `portal` nicht definiert ist. Daher sollten Sie alle Templates nach `exists` durchsuchen und überprüfen, ob die verwendeten globalen Definitionen auch tatsächlich vorhanden sind.

Wegfall des *Action Icons Tool*

Produkte, die Icons für CMF-Aktionen am *Action Icons Tool* regsitrierten, sollten zukünftig die `icon_expr`-Anweisung verwenden um Icons am ‚Actions Tool* oder im *Control Panel Tool* zu registrieren. So wird beispielsweise in Plone4 das Icon für *Document* in `Products/CMFPlone/profiles/default/types/Document.xml` so angegeben:

```
<property name="icon_expr">string:${portal_url}/document_icon.png</property>
```

Wegfall der Zope2-Interfaces

Zope2 vor Version 2.12.0 unterstützte zwei verschiedene Arten von Interfaces, die Zope2- und die Zope3-Implementierung:

Zope2:

```
from Interface import Interface
class MyInterface(Interface):
    pass

class MyClass(object):
    __implements__ = (MyInterface,)
```

Zope3:

```
from zope.interface import Interface
class MyInterface(Interface):
    pass

class MyClass(object):
    implements(MyInterface)
```

In Zope2.12 werden Zope3-Interfaces unterstützt.

Bei einer Zope2-Implementierung von Intervace wird dann folgender Fehler ausgegeben:

```
ImportError: No module named Interface
```

Imports

Diverse *import*-Methoden sind verschoben worden. Früher bereits als *deprecated* gekennzeichnete Methoden wurden entfernt.

Folgende Methoden wurden verschoben:

alt	neu
Products.ATContentTypes.content.folder.ATFolder	plone.app.folder.folder.ATFolder
Products.ATContentTypes.content.folder.ATFolderSchema	plone.app.folder.folder.ATFolderSchema
Products.CMFPlone.browser.navtree.SitemapNavtreeStrategy.iconPo	Products.CMFPlone.browser.navtree.SitemapNavtreeStrategy.item_ico
Products.CMFPlone.browser.plone	Products.CMFPlone.browser.ploneview
Products.CMFPlone.browser.ploneview.cache_decorator	plone.memoize.instance.memoize
Products.CMFPlone.browser.ploneview.IndexIterator	Products.CMFPlone.utils.IndexIterator
Products.CMFPlone.browser.ploneview.Plone.isRightToLeft	@@plone_portal_state/is_rtl
Products.CMFPlone.browser.ploneview.Plone.keyFilteredActions	@@plone_context_state/keyed_actions
Products.CMFPlone.browser.portlets	plone.app.portlets.portlets
Products.CMFPlone.interfaces.OrderedContainer.IOrderedContainer	OFS.interfaces.IOrderedContainer
Products.CMFPlone.utils.BrowserView	Products.Five.BrowserView
Products.CMFPlone.utils.getGlobalTranslationService	Products.PageTemplates.GlobalTranslationService.getGlobalTranslatio
Products.CMFPlone.utils.scale_image	Products.PlonePAS.utils.scale_image
Products.CMFPlone.utils.utranslate	zope.i18n.translate
Products.PageTemplates.GlobalTranslationService.getGlobalTranslationS	zope.i18n.ervice
Products.CMFPlone.utils.ulocalized_time	Products.CMFPlone.i18nl10n.ulocalized_time
zope.app.cache.interfaces.ram.IRAMCache	zope.ramcache.interfaces.ram.IRAMCache
Products.ATReferenceBrowserWidget.ATReferenceBrowserWidget.ReferenceBrowserWidget	plone.app.referencebrowserwidget.ReferenceBrowserWidget

Nicht mehr vorhandene Methoden

- `Products.CMFPlone.CatalogTool.registerIndexableAttribute`

  Stattdessen sollte `plone.indexer` verwendet werden.
- `Products.CMFPlone.PloneTool.setDefaultSkin`
- `Products.CMFPlone.PloneTool.setCurrentSkin`
- `Products.CMFPlone.PortalContent`
- *Favorite*-Artikeltyp
- `use_folder_tabs` aus den `site_properties`
- `keyed_actions`

  Sollte durch die `actions`-Methode aus `@@plone_context_state` ersetzt werden, die nun als einzigen Parameter eine *action category* benötigt.

Validatoren

Auch Validatoren benötigen nun Zope3-Interfaces da ansonsten beim Starten der Instanz folgender Fehler ausgegeben wird:

```
Products.validation.exceptions.FalseValidatorError:
<vs.registration.validators.ProjectIdValidator instance at 0xa92082c>
```

Die Zeile:

```
__implements__= (IValidator,)
```

sollte ersetzt werden durch:

```
from zope.interface import implements
...
try:
   # Plone 4 and higher
    import plone.app.upgrade
    USE_BBB_VALIDATORS = False
except ImportError:
    # BBB Plone 3
    USE_BBB_VALIDATORS = True
...
if USE_BBB_VALIDATORS:
    __implements__ = (ivalidator,)
else:
   implements(IValidator)
```

Aufruf der `translate`-Methode

Folgende Module stehen nicht mehr zur Verfügung:

```
Products.CMFPlone.utils.utranslate
Products.PageTemplates.GlobalTranslationService.getGlobalTranslationService
```

Stattdessen sollte `zope.i18n.translate` verwendet werden.

Und mit `zope.i18n.translate` ändert sich dann auch der Aufruf gegenüber `utranslate`:

- `msgid` ist nun das erste und nicht mehr erst das zweite Argument dieses Aufrufs.
- `domain` ist nun optional.

Add view für Artikeltypen

In Plone 4 kann jeder Artikeltyp im Portal Types Tool eine zusätzliche Eigenschaft für die Ansicht beim Hinzufügen haben. Diese Eigenschaft wird als `TALES`-Ausdruck für eine URL angegeben werden. Ein Link mit dieser URL wird Nutzern dann im *Hinzufügen*-Menü von Plone angezeigt.

Diese Eigenschaft hat den Titel *Add view URL (expression)* und die interne ID `add_view_expr`

This property has the title Add view URL (expression) and the internal id `add_view_expr`.

Auf diese Weise lässt sich z.B. für ein selbst-entwickeltes Hinzufügen-Formular folgender Ausdruck angeben:

```
string:${folder_url}/@@add-my-content
```

Beachten Sie hierbei, dass der View für den `folder`-Artikeltyp registriert wird und nicht für den zu erstellenden Artikeltyp.

`send` statt `secureSend`

Mit der `send`-Methode ändern sich auch weitere Angaben:

Message Type

Nun wird der vollständige MIME type als `msg_type` angegeben und nicht mehr nur der subtype-Parameter, also z.B.:

```
msg_type='text/plain'
```

statt:

```
subtype='plain'
```

Eigene Headers-Angaben

Um eigene Headers-Angaben für eine Nachricht anzugeben, kann z.B. folgendes angegeben werden:

```
from email import message_from_string
from email.Header import Header
my_message = message_from_string(message_body.encode('utf-8'))
my_message.set_charset('utf-8')
my_message['CC']= Header('someone@example.com')
my_message['BCC']= Header('secret@example.com')
my_message['X-Custom'] = Header(u'Some Custom Parameter', 'utf-8')
mailhost.send(my_message, mto, mfrom, subject)
```

Geänderte Syntax des Portlets-Profil

In Plone 3 wird ein Portlet an einen bestimmten Portlet-Manager gebunden mit der Anweisung:

```
for="plone.app.portlets.interfaces.IColumn"
```

In Plone 4 erfolgt dies nun mit:

```
<for interface="plone.app.portlets.interfaces.IColumn" />
```

Somit lassen sich auch mehrere Werte im `for`-Feld angeben:

```
<for interface="plone.app.portlets.interfaces.IColumn" />
<for interface="plone.app.portlets.interfaces.IDashboard" />
```

Zum Weiterlesen

- Upgrading Plone 3.x to 4.0
- Migrating a Product to Plone 4.0
- BACKWARDS_COMPATIBILITY.txt

Migration eines Produkts zu Plone 4.1

Für die Migration auf Plone 4.1 sollten die folgenden Änderungen beachtet werden.

Geänderte Abhängigkeiten von `Plone` zu `Products.CMFPlone`

Aktualisieren der `setup.py`-Datei

In der `setup.py`-Datei sollte bisher folgendes stehen:

```
install_requires=[
    'setuptools',
    'Plone',
],
```

Stattdessen sollte in Plone 4.1 folgendes verwendet werden:

```
install_requires=[
    'setuptools',
    'Products.CMFPlone',
],
```

Auch sollten hier die anderen Pakete aufgelistet werden, von denen das Paket abhängt, z.B. `Products.Archetypes` oder `plone.app.portlets`.

Aktualisieren der Berechtigungen

Sofern `pages` etc. in einer `zcml`-Datei mit ener CMF-Core-Permission geschützt sind, muss die `configure.zcml`-Datei ergänzt werden um:

```
<include package="Products.CMFCore" file="permissions.zcml"
         xmlns:zcml="http://namespaces.zope.org/zcml"
         zcml:condition="have plone-41" />
```

Aktualisieren der Aliase

Einige ältere Import-Aliase funktionieren in Plone 4.1 nicht mehr und müssen ersetzt werden:

Frühere Plone-Versionen	Plone 4.1
`from Products.CMFPlone import Batch`	`from Products.CMFPlone.PloneBatch import Batch`
`from zope.app.interface import queryType`	`from zope.app.content import queryType`
`from Products.Five.formlib import formbase`	`from five.formlib import formbase`

Generic-Setup-Profil für die Versionierung

Ab Plone 4.1 kann die Konfiguration zur Versionierung eigener Artikeltypen im Profil `repositorytool.xml` angegeben werden. Genaue Angaben hierzu finden Sie in

- Enabling versioning on your custom content-types
- History and Versioning

Wenn Sie von Plone 4.0 aktualisieren, erhalten Sie vermutlich folgende Fehlermeldung:

```
ImportError: cannot import name DEFAULT_POLICIES
```

Um die Kompatibilität Ihres Produkts mit Plone 4.1 wiederherzustellen, muss für Ihren bisherigen Code in der `setuphandlers.py`-Datei eine Bedingung angegeben werden, also z.B. statt:

```
from Products.CMFEditions.setuphandlers import DEFAULT_POLICIES
```

sollte folgendes angegeben werden:

```
try:
    from Products.CMFEditions.setuphandlers import DEFAULT_POLICIES
    # we're on plone < 4.1, configure versionable types manually
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    setVersionedTypes(portal)
except ImportError:
    # repositorytool.xml will be used
    pass
```

Für Plone 4.1 wird die Konfiguration zur Versionierung der Artikel im Profil `repositorytool.xml` angegeben:

```
<?xml version="1.0"?>
<repositorytool>
    <policymap>
        <type name="MyType">
            <policy name="at_edit_autoversion"/>
            <policy name="version_on_revert"/>
        </type>
        <type name="AnotherType">
            <policy name="at_edit_autoversion"/>
            <policy name="version_on_revert"/>
        </type>
    </policymap>
</repositorytool>
```

Eigene Upgrade-Skripte schreiben

Falls ein EXTENSION-Profil zum Installieren eines Produkts verwendet wurde, können Upgrade steps des Generic Setup Tools verwendet werden, um Migrationsskripte zu verwalten.

Dabei kann das Generic Setup Tool die Versionsnummer entweder aus der `version.txt`-Datei im Wurzelverzeichnis des Produkts auslesen oder aus einer `metadata.xml`-Datei im Installationsprofil. Entsprechend wurde das `Default`-Profil von `vs.policy` ergänzt um `src/vs.policy/vs/policy/profiles/default/metadata.xml`:

```
<?xml version="1.0"?>
<metadata>
    <description>Policies for www.veit-schiele.de</description>
    <version>1.3</version>
</metadata>
```

Die *Upgrade steps* werden dann in `src/vs.policy/vs/policy/configure.zcml` registriert:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five"
    xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
    i18n_domain="vs.policy">

    ...

    <!-- Upgrade step for the migration -->
    <genericsetup:upgradeStep
        sortkey="1"
        source="1.0"
        destination="1.1"
        title="Upgrade from 1.0 to 1.1"
        description="Fixes the front page title"
        profile="vs.policy:default"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        handler=".upgrades.v1_0_to_v1_1"
        />

    ...

</configure>
```

Hier wird die Aktualisierung von Version 1.0 auf 1.1 definiert, wobei die Funktion `v1_0_to_v1_1()` aufgerufen wird.

Die Funktion ist definiert in `src/vs.policy/vs/policy/upgrades.py`:

```
from Products.CMFCore.utils import getToolByName

def v1_0_to_v1_1(portal_setup):
    portal_url = getToolByName(portal_setup, 'portal_url')
    portal = portal_url.getPortalObject()
    front_page = portal['front-page']
    front_page.setTitle('Welcome to Veit Schiele communication design')
```

Die Aktualisierung von 1.1 auf 1.2 teilt sich in zwei Teile auf, `v1_1_to_v1_2a` und `v1_1_to_v1_2b`. Dabei sorgt `sortkey` für die richtige Reihenfolge bei der Aktualisierung über mehrere Versionen:

```
<genericsetup:upgradeSteps
    sortkey="2"
    source="1.1"
    destination="1.2"
    profile="vs.policy:default"
    >
    <genericsetup:upgradeStep
        title="Upgrade titles"
        description="Fix all other titles"
        handler=".upgrades.v1_1_to_v1_2a"
        />
    <genericsetup:upgradeStep
        title="Upgrade site title"
        description="Fixes the portal title"
        handler=".upgrades.v1_1_to_v1_2b"
        />
</genericsetup:upgradeSteps>
```

Migrationsprofil

Falls eine große Anzahl von Migrationen durchlaufen werden soll, kann sich ein spezielles `Extension`-Profil empfehlen:

```
<genericsetup:registerProfile
      name="1.2_to_1.3"
      title="Migration profile for veit-schiele.de 1.2 to 1.3"
      description=""
      directory="profiles/migrations/v1_2_to_v1_3"
      for="Products.CMFPlone.interfaces.IMigratingPloneSiteRoot"
      provides="Products.GenericSetup.interfaces.EXTENSION"
      />
<genericsetup:upgradeStep
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    sortkey="3"
    source="1.2"
    destination="1.3"
    title="Upgrade from 1.2 to 1.3"
    description="Runs a migration profile"
    profile="vs.policy:default"
    handler=".upgrades.v1_2_to_v1_3"
    />
```

Zunächst wird das `Extension`-Profil in `profiles/migrations/v1_2_to_v1_3` für das Interface `Products.CMFPlone.interfaces.IMigratingPloneSiteRoot` registriert, wodurch das Profil nicht beim Erstellen der Site oder in Plone's Website-Konfiguration zum *Hinzufügen/Entfernen von Produkten* angezeigt wird. Anschließend wird noch ein *upgrade step* registriert, der das Profil angibt.

Schließlich muss noch das Verzeichnis `src/vs.policy/vs/policy/profiles/migrations/v1_2_to_v1_3` und darin die entsprechenden XML-Dateien erzeugt werden.

Python 3-Migration

Requirements

- Alle Third-Party-Add-Ons sollten sowohl Python 2 wie auch Python 3 unterstützen.

  Eine Übersicht, welche Add-Ons in Collective bereits auf Python 3 aktualisiert wurden, findet ihr in Python 3 porting state for Plone add-ons.

- Zur Python-3-Migration verwenden wir six und modernize.

  sie können installiert werden mit:

```
$ python3 -m venv py3env
$ cd py3env
$ ./bin/pip install modernize six
```

precompiler

Zusätzlich verwenden wir plone.recipe.precompiler um Syntaxfehler zu finden. Er kann mit Buildout installiert werden, indem in der `py3.cfg`-Datei folgendes angegeben wird:

```
parts += precompiler
...
[precompiler]
recipe = plone.recipe.precompiler
eggs = ${instance:eggs}
compile-mo-files = true
```

`precompile` wird jedes Mal ausgeführt, wenn ihr Bbuildout ausführt. Wenn ihr nur `precompile` ausführen möchtet, könnt ihr dies mit:

```
$ bin/buildout -c py3.cfg  install precompiler
```

python-modernize

`python-modernize` bereitet Python-2-Code automatisch für die Python-3-Portierung vor. Dabei weist euch `python-modernize` auf Probleme hin, die nicht automatisch gelöst werden können.

Mit `bin/python-modernize -x libmodernize.fixes.fix_import src/my.package` könnt ihr euch anzeigen lassen, welche Änderungen `modernize` an eurem Plone-Add-on `my.package` vornehmen würde.

Bemerkung: Ihr könnt `python-modernize` u.a. mit folgenden Optionen aufrufen:

`-x` schließt bestimmte Fixers aus.

`-l` listet euch alle verfügbaren *Fixers* auf.

Bemerkung: Im Cheat Sheet Writing Python 2-3 compatible code erhaltet ihr einen Überblick, wie sich die Syntax von Python 2 zu Python 3 ändert.

- Wir verwenden die `py3.cfg` aus dem Plone-5.2-Branch von vs_buildut:

```
$ bin/buildout -c py3.cfg
```

Starten der Plone-Instanz

```
$ bin/wsgi.py
```

Häufige Probleme beim Starten sind:

- `Class Advice`
- Relative Imports
- Syntax Error beim Import von `async`

Testen

Neben dem manuellen Testen solltet ihr automatisiert Testen mit:

```
$ bin/test --all -s my.package
```

Alternativ könnt ihr den Testrunner automatisch den Python-Debugger starten lassen mit:

```
$ bin/test -s my.package -D
```

Aktualisieren der Metainformationen

Aktualisiert die `classifiers` in `setup.py` aktualisiert:

```
classifiers=[
    ...
    "Framework :: Plone :: 5.2",
    ...
    "Programming Language :: Python :: 3.6",
    "Programming Language :: Python :: 3.7",
    "Programming Language :: Python :: 3.8",
    ...
],
```

Häufige Probleme

Strings: Text vs. Bytes

Meist wird in Plone Text verwendet und nur in sehr seltenen Fällen Bytes.

Versucht den Code so zu ändern, dass ihr mit Beenden der Python 2-Unterstützung einfach nur das``if``-Statement löschen müsst, z.B.:

```
if six.PY2 and isinstance(value, six.text_type):
    value = value.encode('utf8')
do_something(value)
```

Dabei könnt Ihr Hilfsmethoden verwenden wie `safe_text`, `safe_bytes`, `safe_unicode` und `safe_encode`, z.B.:

```
from Products.CMFPlone.utils import safe_unicode
...
obj = self.context.unrestrictedTraverse(
    safe_unicode(item['_path'].lstrip('/')), None)
```

`python-modernize` ändert ebenfalls nicht `from StringIO import StringIO` obwohl der Import nur in Python-2 funktioniert. Für Python-3 müsst ihr überprüfen, ob es sich um Text- oder Binärdaten handelt und die import-Anweisung entsprechend schreiben:

```
from six import StringIO
```

oder:

```
from six import BytesIO
```

Weitere Informationen findet ihr im The Conservative Python 3 Porting Guide.

Fehlerbehebung

Prozeduren beim Auftreten von Problemen während des Plone-Upgrades.

1. Fährt die Zope-Instanz nicht mit `ZOPE_INSTANCE/bin/zopectl start` hoch, erhalten Sie mit `ZOPE_INSTANCE/bin/zopectl fg` Hinweise, an welcher Stelle Zope hängen bleibt.
2. Überprüfen der Log-Dateien

   Wenn ein Fehler auf der aktualisierten Plone-Site ausgegeben wird, gibt es möglicherweise eine informative Fehlermeldung in den Log-Dateien der Zope-Instanz. Gehen Sie zu `ZOPE_INSTANCE/logs/event.log` und suchen dort nach `error`, `exception` oder `traceback`.

   Detailliertere Informationen zu den Fehlermeldungen finden Sie in

   - Version-specific migration procedures and tips;
   - Plone Error References
3. Testen der Anpassungen

   Wenn Sie Page Templates und Python-Skripte angepasst haben, mögen diese Änderungen mit der neuen Plone-Versionen interferieren. Entfernen Sie temporär diese Anpassungen indem Sie die Layer aus dem von Ihnen verwendeten Skin entfernen.
4. Testen der Zusatzprodukte
   1. Um Bugs oder Kompatibilitätsprobleme in den installierten Zusatzprodukten herauszufinden, deinstallieren Sie alle Produkte, die nicht zusammen mit Plone ausgeliefert werden, in *Site Setup*, *Add/Remove Products*. Anschließend entfernen Sie diese Produkte aus dem Produktverzeichnis Ihrer Zope-Instanz.
   2. Falls das Problem hiermit beseitigt wird, überprüfen Sie für jedes der betreffenden Produkte:
      - ob dessen Version auch mit den aktuellen Versionen von Plone, Zope und Python kompatibel ist?
      - ob dieses Produkt zusätzliche Upgrade-Prozeduren benötigt?
      - ob das Produkt korrekt installiert ist? Installieren Sie es erneut und überprüfen Sie die Ausgabe beim Starten der Zope-Instanz mit `ZOPE_INSTANCE/bin/zopectl fg`.
      - Liegt ein Produkt nicht für die aktuelle Plone-Version vor und hinterlässt beim Deinstallieren »verwaiste« Inhalte, so können Sie sich mit folgendem Skript solche Inhalte auflisten lassen:

```
portal_types = context.portal_types.objectIds()

print "Orphaned items:"
print

for i in context.portal_catalog.uniqueValuesFor('portal_type'):
   if i in portal_types: continue
   print i
   results = context.portal_catalog(portal_type=i)
   for i in results:
       print i.getURL()
   print
return printed
```

      Um das Skript zu verwenden, erstellen Sie im ZMI des Wurzelverzeichnisses Ihrer Plone-Site ein Python-Skript, kopieren den Code in das Formular und sichern es anschließend. Zum Ausführen des Skripts müssen Sie nun nur noch in den «Test»-Reiter klicken.
   3. Wie Sie Upgrade-Skripte für Ihre eigenen Produkte schreiben können, erfahren Sie hier: Eigene Upgrade-Skripte schreiben.

5. Testen Sie in einer neu aufgesetzten Plone-Site.

   Falls das Problem nicht in einer neu aufgesetzten Plone-Site auftritt, bedeutet dies, dass die Ursache zu suchen ist

   - in einer Anpassung
   - einem Zusatzprodukt
   - oder in Inhalten, die nicht sauber migriert wurden.

6. Machen Sie das Problem reproduzierbar

   Bevor Sie andere um Hilfe bitten, sollten Sie das Problem so beschreiben können, dass es für andere in deren Systemumgebung nachvollziehbar ist. Schränken Sie hierzu die Fehlerquellen so weit wie möglich ein um es anderen zu erleichtern, Ihr Problem nachvollziehen zu können.

Entfernen von Local Utilities, Subscribers, Adapters und Portal Tools

Häufig werden Local Persistent Utilities beim Deinstallieren eines Produkts nicht ebenfalls entfernt.

Bemerkung: Mit wildcard.fixpersistentutilities steht nun ein Modul zur Verfügung, mit dem sich Local Utilities, Subscribers, Adapters und Portal Tools auf der Web-Oberfläche entfernen lassen.

Symptome

Nach dem Deinstallieren solcher Produkte kann es z.B. folgende Fehlermeldungen geben:

```
AttributeError: type object 'IQueue' has no attribute '__iro__'
```

oder:

```
AttributeError: type object 'ISalt' has no attribute '__iro__'
```

Entfernen von Local Persistent Utilities

1. Starten der Instanz im Debug-Modus:

   ```
   $ ./bin/instance debug
   ```

2. Anschließend holen wir uns den *site manager* der Site `Plone`. *àpp`* referenziert dabei auf das Zope-Root-Objekt:

   ```
   sm = app.Plone.getSiteManager()
   ```

3. Nun importieren wir das Interface des Utility. Anschließend melden wir es ab und löschen es schließlich. Dies sieht z.B. für Singing & Dancing so aus:

   ```
   from collective.singing.interfaces import ISalt
   from collective.singing.async import IQueue

   util_obj = sm.getUtility(ISalt)
   sm.unregisterUtility(provided=ISalt)
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
del util_obj
sm.utilities.unsubscribe((), ISalt)
del sm.utilities.__dict__['_provided'][ISalt]
del sm.utilities._subscribers[0][ISalt]

util = sm.queryUtility(IQueue, name='collective.dancing.jobs')
sm.unregisterUtility(util, IQueue, name='collective.dancing.jobs')
del util
del sm.utilities._subscribers[0][IQueue]
```

Dabei unterscheidet sich das Vorgehen, da für `ISalt` ein *unnamed utility* und für `IQueue` ein *named utility* registriert sind.

4. Anschließend müssen die Änderungen noch an der ZODB *commited* werden:

```
import transaction
transaction.commit()
app._p_jar.sync()
```

Entfernen von Subscribers, Adapters und Providers

```
sm = app.Plone.getSiteManager()

adapters = sm.utilities._adapters
for x in adapters[0].keys():
    if x.__module__.find("my.package") != -1:
      print "deleting %s" % x
      del adapters[0][x]
sm.utilities._adapters = adapters

subscribers = sm.utilities._subscribers
for x in subscribers[0].keys():
    if x.__module__.find("my.package") != -1:
      print "deleting %s" % x
      del subscribers[0][x]
sm.utilities._subscribers = subscribers

provided = sm.utilities._provided
for x in provided.keys():
    if x.__module__.find("my.package") != -1:
      print "deleting %s" % x
      del provided[x]
sm.utilities._provided = provided

from transaction import commit
commit()
app._p_jar.sync()
```

`Plone` Die ID der Site

`my.package` Das Paket, aus dem Subscriber, Adapter und Provider kommen

Entfernen von Portal Tools

Nach dem Entfernen von lokalen persistenten Komponenten muss ggf. auch noch das Portal Setup Tool bereinigt werden:

```
setup_tool = app.Plone.portal_setup
toolset = setup_tool.getToolsetRegistry()
if 'my.package' in toolset._required.keys():
    del toolset._required['my.package']
    setup_tool._toolset_registry = toolset

from transaction import commit
commit()
app._p_jar.sync()
```

Zum Weiterlesen

- Removing a persistent local utility
- Removing a persistent local utility part II
- How to remove local utility
- gist.github.com/thet/thet/upgrades.py

wildcard.fixpersistentutilities

wildcard.fixpersistentutilities ist ein Python-Package, das das einfache Entfernen von Local Persistent Utilities ermöglicht. Damit lassen sich p4a, Singing & Dancing, LinguaPlone etc. einfach beseitigen.

wildcard.fixpersistentutilities stellt folgende Features bereit:

- Entfernen von *adapters*
- Entfernen von *subscribers*
- Entfernen von *provided interfaces*
- Entfernen von *provided interfaces*

  Dies ist z.B. hilfreich beim Entfernen von collective.flowplayer.

Das Paket sollte nie in produktiven Umgebungen eingesetzt werden. Auch sollten Sie vorher immer eine Sicherungskopie Ihrer ZODB erstellt haben. Um das Paket dann nutzen zu können, fügen Sie einfach der URL Ihrer Plone-Site-Root `@@fix-persistent-utilities` hinzu. Anschließend können Sie alle registrierten Utilities Ihrer Website durchsuchen und ggf. entfernen.

p4a entfernen

Bemerkung: Mit *wildcard.fixpersistentutilities* steht nun ein Modul zur Verfügung, mit dem sich Local Utilities, Subscribers, Adapters und Portal Tools auf der Web-Oberfläche entfernen lassen.

Ist in einer Instanz jemals ein Plone 4 Artists (p4a)-Produkt installiert worden, lässt sich deren Utilities und Interfaces nicht mehr einfach entfernen.

Wir haben nun ein Skript entwickelt, mit dem sich diese Utilities und Interfaces löschen lassen: fixinterfaces.py.

Dieses Skript sollte im Wurzelverzeichnis des Buildout-Projekts abgelegt werden. Anschließend kann die Instanz im Debug-Modus gestartet werden:

```
$ ./bin/instance debug
```

Sofern unsere Site nun die ID `mysite` hat, werden die Utilities und Interfaces gelöscht.

Falls die Site eine andere ID enthält, lasst sich das Skript einfach ändern. Auch für andere Zusatzprodukte kann das Skript leicht modifiziert werden.

GSXML

GSXML nutzt GenericSetup um ATCT-basierte Inhalte als XML-Daten im- und exportieren zu können.

Anforderungen

GSXML benötigt

- lxml

  lxml kann folgendermaßen installiert werden:

  1. Zunächst wird folgende Änderung in der `buildout.cfg`-Datei vorgenommen:

     ```
     [buildout]
     ...
     extends =
         ...
         lxml.cfg
     ```

  2. Die `lxml.cfg`-Datei sieht dann folgendermaßen aus:

     ```
     [lxml]
     parts =
        staticlxml
        pylxml

     [pylxml]
     recipe=zc.recipe.egg
     interpreter=pylxml
     eggs=
         lxml

     [staticlxml]
     ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
recipe = z3c.recipe.staticlxml
egg = lxml
```

3. Nun sollte Buildout problemlos durchlaufen und die Instanz neu gestartet werden können:

```
$ ./bin/buildout
$ ./bin/instance fg
```

Installation

GSXML kann einfach als Egg installiert werden:

```
[buildout]
...
eggs =
    elementtree
    collective.plone.gsxml
    ...
```

Anschließend wird das Egg noch im `[instance]`-Abschnitt angegeben:

```
[instance]
...
zcml=
    collective.plone.gsxml
    ...
```

Nachdem das Buildout-Skript erneut durchlaufen und die Instanz gestartet wurde kann GSXML einfach mit dem Quickinstaller für die Plone-Site installiert werden. Anschließend erscheinen im *Aktionen*-Menü die Einträge *Export* und *Import*:

LinguaPlone-Migration

Für die Migration einer Plone-Site mit LinguaPlone zu plone.app.multilingual sind zunächst einmal beide Produkte zu installieren. Hierfür wird die Buildout-Konfiguration der Plone- 4.3-Site entsprechend geändert, z.B. in der `devel.cfg`:

```
[instance-base]
eggs +=
    ...
    Products.LinguaPlone
    plone.app.multilingual
```

... und in der `versions.cfg`:

```
[versions]
...
Products.LinguaPlone = 4.1.3
plone.app.multilingual =  2.0.0
```

Siehe hierfür auch github.com/veit/vs_buildout/commit.

Anschließend wird das Buildout-Skript aufgerufen und die Instanz neu gestartet. Nun kann in der Plone-4.3-Site `plone.app.multilingual` aktiviert und `LinguaPlone` deaktiviert werden. Anschließend sollten im *language controlpanel* im *Migration*- Reiter die folgenden vier Schritte ausgeführt werden:

1. Reindexieren des Sprachindex (optional)

   Die Migration von LinguaPlone basiert auf einem aktuellen Sprachindex.

2. Zuordnen der Inhalte zum passenden *root language*-Ordner

   In diesem Schritt werden die Inhalte in die entsprechenden *root language*- Ordner verschoben.

3. Übertragen der Referenzen aus dem LinguaPlone-Katalog in den `plone.app.multilingual`-Katalog

4. Aufräumen nach der Migration Dieser Schritt sucht und repariert verlorene Verknüpfungen.

   Dieser Schritt ist nur erforderlich wenn LinguaPlone nicht installiert ist. Daher wird dieser Schritt auch nur dann angezeigt.

Tipps

Tipps zu Migrationen und zukunftsfähigeren Produkten.

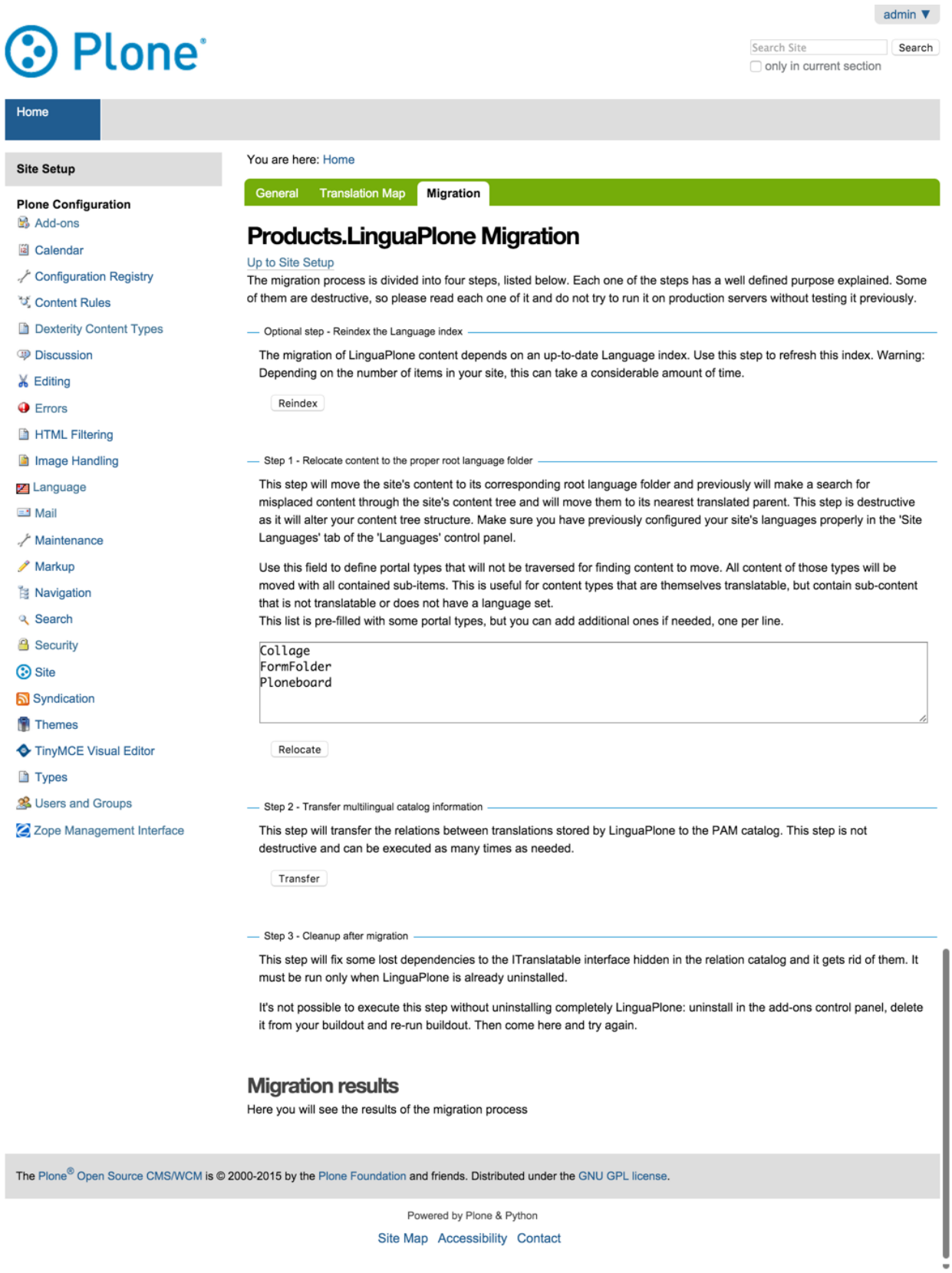
- Starten Sie Ihre Zope-Instanz im Debug-Modus (`ZOPE_INSTANCE/bin/zopectl fg`) und nutzen anschließend Ihr Produkt. Überprüfen Sie, ob auf der Konsole *deprecation warnings* ausgegeben werden.
- Entfernen Sie den `plone_deprecated`-Skin-Layer und überprüfen, ob Ihre Anwendung anschließend noch problemlos funktioniert.
- Auf dem QuickInstaller basierende Installationen sollten stattdessen GenericSetup-Profile verwenden.
  - Ab CMF 2.1.0 steht `addWorkflowFactory` nicht mehr zur Verfügung. Workflows können von da an nur noch mit dem GenericSetup-Tool erstellt werden. Zur Migration installieren Sie einfach Ihr Produkt in einer Plone2.5-Site und exportieren den Workflow anschließend mit dem GenericSetup-Tool. Bevor Sie schließlich die alte Python-basierte Workflow-Definition aus dem *Extensions*-Ordner löschen, achten Sie darauf, dass ggf. die *workflow scripts* erhalten bleiben.
- `manage_`-Methoden sollten durch `events` ersetzt werden.
- Um die *Permissions* aus dem CMFCore-Produkt zu importieren, sollte in der `__init__.py` Datei folgender (rückwärtskompatibler) Code verwendet werden:

```
try: # for Plone 2.5 and above
    from Products.CMFCore import permissions as CMFCorePermissions
except: # for Plone 2.1
    from Products.CMFCore import CMFCorePermissions
```

- Archetypes 1.5, das zusammen mit Plone 3.0 ausgeliefert wird, enthält kein `Transaction`-Modul mehr, statt:

```
from Products.Archetypes import transaction
```

  kann nun einfach folgendes verwendet werden:


admin
Plone
Search Site
Search
only in current section
Home
Site Setup
Plone Configuration
Add-ons
Calendar
Configuration Registry
Content Rules
Dexterity Content Types
Discussion
Editing
Errors
HTML Filtering
Image Handling
Language
Mail
Maintenance
Markup
Navigation
Search
Security
Site
Syndication
Themes
TinyMCE Visual Editor
Types
Users and Groups
Zope Management Interface
You are here: Home
General
Translation Map
Migration
Products.LinguaPlone Migration
Up to Site Setup
The migration process is divided into four steps, listed below. Each one of the steps has a well defined purpose explained. Some of them are destructive, so please read each one of it and do not try to run it on production servers without testing it previously.
Optional step - Reindex the Language index
The migration of LinguaPlone content depends on an up-to-date Language index. Use this step to refresh this index. Warning: Depending on the number of items in your site, this can take a considerable amount of time.
Reindex
Step 1 - Relocate content to the proper root language folder
This step will move the site's content to its corresponding root language folder and previously will make a search for misplaced content through the site's content tree and will move them to its nearest translated parent. This step is destructive as it will alter your content tree structure. Make sure you have previously configured your site's languages properly in the 'Site Languages' tab of the 'Languages' control panel.
Use this field to define portal types that will not be traversed for finding content to move. All content of those types will be moved with all contained sub-items. This is useful for content types that are themselves translatable, but contain sub-content that is not translatable or does not have a language set.
This list is pre-filled with some portal types, but you can add additional ones if needed, one per line.
Collage
FormFolder
Ploneboard
Relocate
Step 2 - Transfer multilingual catalog information
This step will transfer the relations between translations stored by LinguaPlone to the PAM catalog. This step is not destructive and can be executed as many times as needed.
Transfer
Step 3 - Cleanup after migration
This step will fix some lost dependencies to the ITranslatable interface hidden in the relation catalog and it gets rid of them. It must be run only when LinguaPlone is already uninstalled.
It's not possible to execute this step without uninstalling completely LinguaPlone: uninstall in the add-ons control panel, delete it from your buildout and re-run buildout. Then come here and try again.
Migration results
Here you will see the results of the migration process
The Plone® Open Source CMS/WCM is © 2000-2015 by the Plone Foundation and friends. Distributed under the GNU GPL license.
Powered by Plone & Python
Site Map
Accessibility
Contact

```
import transaction
```

- Zope hat die Syntax, um Transaktionen zu importieren geändert. Ab Zope 2.10.x wird die bisherige Syntax:

```
get_transaction().commit(1)
```

durch folgende ersetzt:

```
transaction.commit(1)
```

`transaction` muss selbstverständlich zunächst importiert werden.

- CMF hat seit längerer Zeit `ContentFactoryMetadata` durch `FactoryTypeInformation` ersetzt. Der Aufruf:

```
from Products.CMFCore.TypesTool import ContentFactoryMetadata
```

sollte also ersetzt werden durch:

```
from Products.CMFCore.TypesTool import FactoryTypeInformation
```

- Berechtigungen wie z.B. `cmf.ModifyPortalContent` müssen in `zcml` angegeben werden:

```
<include package="Products.CMFCore" file="permissions.zcml"
         xmlns:zcml="http://namespaces.zope.org/zcml"
         zcml:condition="have plone-41" />
```

Dabei wird `zcml:condition` in Plone 4.1.3 definiert in der `meta.zcml`-Datei:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:meta="http://namespaces.zope.org/meta">

    <meta:provides feature="plone-4" />
    <meta:provides feature="plone-41" />

</configure>
```

Migrieren beliebiger Webinhalte nach Plone

Mit collective.transmogrifier lassen sich sog. Pipelines für den Ex- und Import von Webinhalten konfigurieren. Und zum Crawlen und Parsen einer statischen Website wird funnelweb verwendet.

Am Beispiel der auf dem Documentation-Server Sphinx basierenden Website des Plone-Nutzerhandbuch zeige ich exemplarisch, wie eine solche Migration aussehen kann: https://dev.veit-schiele.de/svn/plone-nutzerhandbuch/trunk/

Voraussetzungen

- Git

  Linux:

```
$ sudo apt-get install git-core
```

  Mac:

```
$ sudo port install git-core
```

Installation

Erstellen des `migration`-Skripts:

```
$ ./bin/buildout -c migration.cfg
```

Aufrufen des Skripts mit:

```
$ ./bin/migration --ploneupload:target=http://admin:secret@localhost:8080/Plone/
↪documentation/manual/plone-nutzerhandbuch
```

Anmerkung 1: In userem Fall wird das Plone-Nutzerhandbuch in eine lokale Plone-Site mit der ID Plone importiert wobei das Plone Help Center die ID documentation und das reference Manual die ID plone-nutzerhandbuch hat. Falls Sie die Dokumentation in eine andere Plone-Site mit anderen HTTP Basic Auth-Credentials importieren möchten, können Sie diese Zeile selbstverständlich entsprechend abändern.

Folgende Schritte werden während der Migration ausgeführt:

1. Aus der Sphinx-Dokumentation werden Titel, Beschreibung und Haupttext von jeder Seite extrahiert.
2. Anschließend werden die Inhalte für das Plone Help Center mit XML-RPC erzeugt.
3. Veröffentlichen der Artikel sofern notwendig und Verstecken des Ordners, der die Bilder enthält in der Navigation.

Anmerkung 2: Das Skript zum Hochladen der Dateien überschreibt momentan noch nicht bereits vorhandene Artikel. Falls Seiten umbenannt oder verschoben wurden, sollte zunächst die gesamte Dokumentation gelöscht werden bevor die Dateien erneut hochgeladen werden.

Konfiguration

`migration.cfg`

Das Skript `bin/migration` wird erstellt mit der in der `migration.cfg` angegebenen Konfiguration:

```
[buildout]
...
migration

[migration]
recipe = funnelweb
crawler-url=file://${buildout:directory}/docs/html
crawler-ignore=
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    cgi-bin
    javascript:
    _static
    _sources
    genindex\.html
    search\.html
    saesrchindex\.js
cache-output =
template1-title = text //div[@class='body']//h1[1]
template1-_permalink = text //div[@class='body']//a[@class='headerlink']
template1-text = html //div[@class='body']
template1-_label = optional //p[contains(@class,'admonition-title')]
template1-description = optional //div[contains(@class,'admonition-description')]/
↪p[@class='last']/text()
template1-_remove_useless_links = optional //div[@id = 'indices-and-tables']
templateauto-condition = python:False
titleguess-condition = python:True
indexguess-condition = python:True
hideguess-condition =  python:item.get("_path","").startswith('_images') and item.get(
↪'_type')=='Folder'
changetype-value=python:{'Folder':'HelpCenterReferenceManualSection','Document':
↪'HelpCenterLeafPage'}.get(item['_type'],item['_type'])
ploneprune-condition=python:item.get('_type') in ['HelpCenterReferenceManualSection',
↪'HelpCenterReferenceManual'] or item['_path'] == ''
```

recipe Verwendet wird funnelweb. Weitere Informationen zu diesem Abschnitt erhalten Sie auf der Homepage.

crawler-url legt die zu migrierende Website fest, in unserem Fall `file://${buildout:directory}/docs/html`. Es könnte jedoch auch eine URL wie z.B. `http://www.plone-nutzerhandbuch.de` angegeben werden.

crawler-ignore Links, denen nicht gefolgt werden soll, können mit regulären Ausdrücken angegeben werden.

In Falle des Documentation Servers Sphinx werden u.a. die Verzeichnisse `_static` und `_sources` ignoriert.

cache-output Da unsere Inhalte aus dem Dateisystem kommen, ist hier kein lokaler Cache nötig.

titleguess-condition, indexguess-condition Bilder erhalten ihren Titel aus dem Backlink-Text

hideguess-condition Bilder werden nicht in der Navigation angezeigt

changetype-value Statt Ordnern werden HelpCenterLeafPage- und statt Seiten HelpCenterLeafPage-Artikeltypen angelegt

ploneprune-condition Die Inhalt aller ordnerähnlichen Artikel wird daraufhin überprüft, ob sie Inhalte enthalten, die noch nicht lokal gespeichert sind.

pipeline.cfg

Die `pipeline.cfg` definiert dann die Reihenfolge, in der die Anweisungen abgearbeitet werden. Ein Beispiel finden Sie wieder für das Plone-Nutzerhandbuch: pipeline.cfg

drop-resources filtert Ressourcen wie css-, Javascript- und Anwendungen aus

drop-unneeded-html filtert nicht benötigten HTML-Code aus

treeserializer muss vor dem localconstructor ausgeführt werden

templatefinder extrahiert Titel, Beschreibung und Haupttext aus den von Sphinx generierten Seiten.

Beachten Sie bitte, dass *Note*-Leerzeichen in XPaths als `&#32;` angegeben werden müssen.

Weitere Infos zu XPath erhalten Sie unter

- http://www.w3schools.com/xpath/default.asp
- http://blog.browsermob.com/2009/04/test-your-selenium-xpath-easily-with-firebug/

mark-container-remote-content-type erstellt einen Hinweis, sodass Verzeichnisse beim Hochladen als HelpCenterReferenceManualSection angelegt werden

mark-page-remote-content-type erstellt einen Hinweis, sodass HTML-Dateien beim Hochladen als HelpCenterLeafPage-Artikel angelegt werden

mark-image-folders-to-navigation-exclusion versteckt den images-Ordner in der Navigation.

mark-remote-folder-to-be-cleaned überprüft, ob ein Ordner Objekte enthält, die noch nicht lokal gespeichert wurden.

mark-remote-root-to-be-cleaned löscht alle Inhalte aus dem ausgewählten Reference Manual

topublish erstellt einen Hinweis, damit nach dem Hochladen der Workflow-Status auf veröffentlicht gesetzt wird.

Dabei wird berücksichtigt, dass Bilder keinen Workflow unterliegen.

ploneuploader erstellt die Artikel in der Plone-Site

schemaupdater aktualisiert die Inhalte der Plone-Site mit den extrahierten Inhalten aus der Sphinx-Dokumentation

set-folder-default-page setzt index.html als Standardseite eines Ordners

publish veröffentlicht die hochgeladene Dokumentation sofern sie noch nicht veröffentlicht ist

excludefromnavigation versteckt Artikel in der Navigation

cleanremotefolder löscht Objekte, die auf der Plone-Site sind, jedoch nicht in der lokalen Kopie

JSON-Im- und Export

JSON (JavaScript Object Notation) ist gut geeignet um die Inhalte aus Plone im- und exportieren zu können. collective.jsonmigrator kann ggf. diese Azfgabe erleichtern.

Alternativ lässt sich auch einfach ein eigener View schreiben zum Export im JSON-Format:

```
import os
import base64

try:
    import json
except ImportError:
    # Python 2.5/Plone 3.3 use simplejson
    import simplejson as json

from Products.Five.browser import BrowserView
from Products.CMFCore.interfaces import IFolderish
from DateTime import DateTime

# Private attributes for the export list
EXPORT_ATTRIBUTES = ["portal_type", "id"]

class ExportFolderAsJSON(BrowserView):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    """
    Exports the current context folder as JSON.
    """

    def convert(self, value):
        """
        Convert value to more JSON friendly format.
        """
        if isinstance(value, DateTime):
            # Zope DateTime
            return value.ISO8601()
        elif hasattr(value, "isBinary") and value.isBinary():
            return None

            # FileField and ImageField payloads are binary
            # as OFS.Image.File object
            data = getattr(value.data, "data", None)
            if not data:
                return None
            return base64.b64encode(data)
        else:
            return value

    def grabData(self, obj):
        """
        Export schema data as dictionary object.
        Binary fields are encoded as BASE64.
        """
        data = {}
        for field in obj.Schema().fields():
            name = field.getName()
            value = field.getRaw(obj)
            print "%s" % (value.__class__)

            data[name] = self.convert(value)
        return data

    def grabAttributes(self, obj):
        data = {}
        for key in EXPORT_ATTRIBUTES:
            data[key] = self.convert(getattr(obj, key, None))
        return data

    def export(self, folder, recursive=False):
        """
        Export content items.
        Possible to do recursively nesting into the children.
        :return: list of dictionaries
        """

    array = []
        for obj in folder.listFolderContents():
            data = self.grabData(obj)
            data.update(self.grabAttributes(obj))

            if recursive:
                if IFolderish.providedBy(obj):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                    data["children"] = self.export(obj, True)
            array.append(data)
        return array

    def __call__(self):
        """
        """
        folder = self.context.aq_inner
        data = self.export(folder)
        pretty = json.dumps(data, sort_keys=True, indent='    ')
        self.request.response.setHeader("Content-type", "application/json")
        return pretty
```

3.14 Ausblick

Siehe auch:

- Martin Aspeli: Pete and Andy try Plone 4
- Martin Aspeli: Is Plone too hard?
- Mock It!: Viewlets are simple again
- Entransit Content Deployment

3.14.1 Deliverance

Deliverance wird zur Gestaltung von HTML verwendet wobei ein konsistenter Stil auf verschiedene Anwendungen und statische Dateien angewendet werden kann.

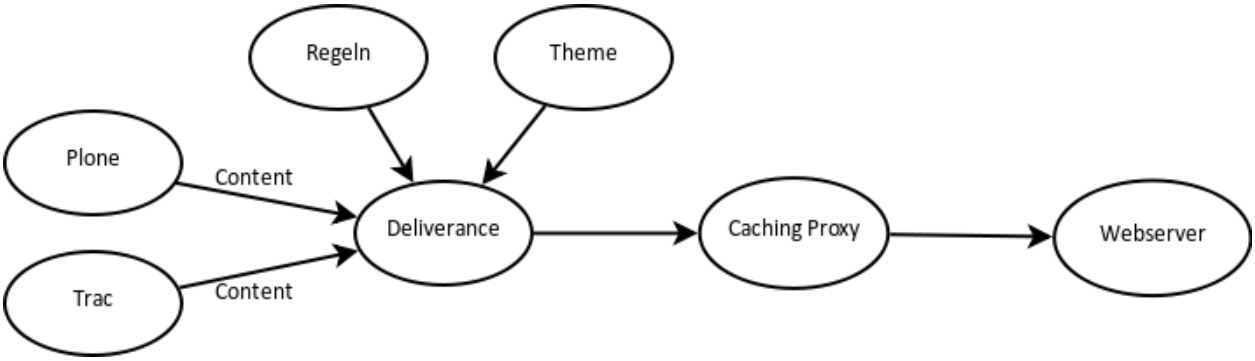
Einführung

Mit Deliverance kann eine Plone-Site ähnlich wie mit *collective.xdv* über XSLT-Regeln gestaltet werden. Daher teilt `Deliverance` mit `collective.xdv` folgende Vorteile:

- Web-Designer müssen kein Vorwissen in Bezug auf Plone und Python mitbringen;
- Standardbibliotheken und -werkzeuge können verwendet werden;
- Auch eine bereits existierende Webgestaltung, z.B. von Open Source Web Design, kann einfach verwendet werden.

`Deliverance` bietet darüberhinaus noch folgende Vorteile:

- Die Transformationsregeln lassen sich an Bedingungen knüpfen und gehen damit über XSLT hinaus.
- `Deliverance` ist **nicht** Plone-spezifisch, sodass das Theme auch für weitere Webanwendungen wie Trac, Mailman, Wordpress etc. genutzt werden kann.
- Mit `Deliverance` lassen sich einfach Mashups verschiedener Webinhalte darstellen.



Installation

`Deliverance` lässt sich einfach mit Buildout installieren. Hierzu erstellen Sie eine `deliverance.cfg`-Datei mit folgendem Inhalt:

```
[buildout]
parts =
    lxml
    server

# Change the number here to change the version of Plone being used
extends =

versions = versions
# Add additional egg download sources here. dist.plone.org contains archives
# of Plone packages.
find-links =

# Add additional eggs here
eggs =

# Reference any eggs you are developing here, one per line
# e.g.: develop = src/my.package
develop =

[versions]
Deliverance = 0.6.0
WebOb = 0.9.8
lxml = 2.2.4

[lxml]
recipe = z3c.recipe.staticlxml
egg = lxml
force = false

[server]
recipe = zc.recipe.egg

eggs =
    lxml
    PasteScript
    Deliverance
interpreter = py
```

Nun sollte Buildout problemlos durchlaufen werden:

```
$ ./bin/buildout -c deliverance.cfg
...
Generated script '/home/veit/deliverance_buildout/bin/paster'.
Generated script '/home/veit/deliverance_buildout/bin/deliverance-proxy'.
Generated interpreter '/home/veit/deliverance_buildout/bin/py'.
```

Server-Konfiguration

Nun definieren wir in `rules.xml` die Konfiguration für den Deliverance-Server:

```
<server-settings>
    <server>localhost:8000</server>
    <execute-pyref>true</execute-pyref>
    <dev-allow>localhost</dev-allow>
    <dev-user username="admin" password="secret" />
</server-settings>
```

<server> gibt den verwendeten Host und Port an.

Wird `localhost` oder `127.0.0.1` angegeben, sind nur lokale Verbindungen zulässig.

Der Standardwert ist `localhost:8080`

<execute-pyref> Hiermit können diejenigen, die Transformationsregeln schreiben dürfen, mit jeder Anfrage beliebigen Code ausgeführen.

Der Standardwert ist `true`.

Für weitere Informationen schauen Sie in die `pyref`-Python-Referenz

<dev-allow> Liste der IP-Adressen, die Zugang erhalten.

<dev-user> Die Angabe von Nutzername und Passwort ist nur sinnvoll, wenn `<dev-allow>` sehr resktrikiv gehandhabt wird.

Weitere Informationen zur Server-Konfiguration erhalten Sie in server-settings.

Proxy-Konfiguration

In derselben Datei werden auch die Pfade für den Proxy-Server angegeben:

```
<proxy path="/static" class="static" editable="1">
    <dest href="{here}/static/" />
</proxy>

<proxy path="/" class="plone">
    <dest href="http://localhost:8080/VirtualHostBase/http/localhost:8000/Plone/
↪VirtualHostRoot/" />
</proxy>
```

<proxy path> Dies liefert bei einer Anfrage an `/static` die Dateien aus `{here}/static` aus, wobei `{here}` dasselbe Verzeichnis ist, in dem auch die `rules.xml`-Datei liegt.

<dest> gibt das Ziel der Anfrage an.

Es kann sowohl als href-URL (`http://...`) als auch als Datei-URL (`file:///...`) angegeben werden.

Starten des Servers

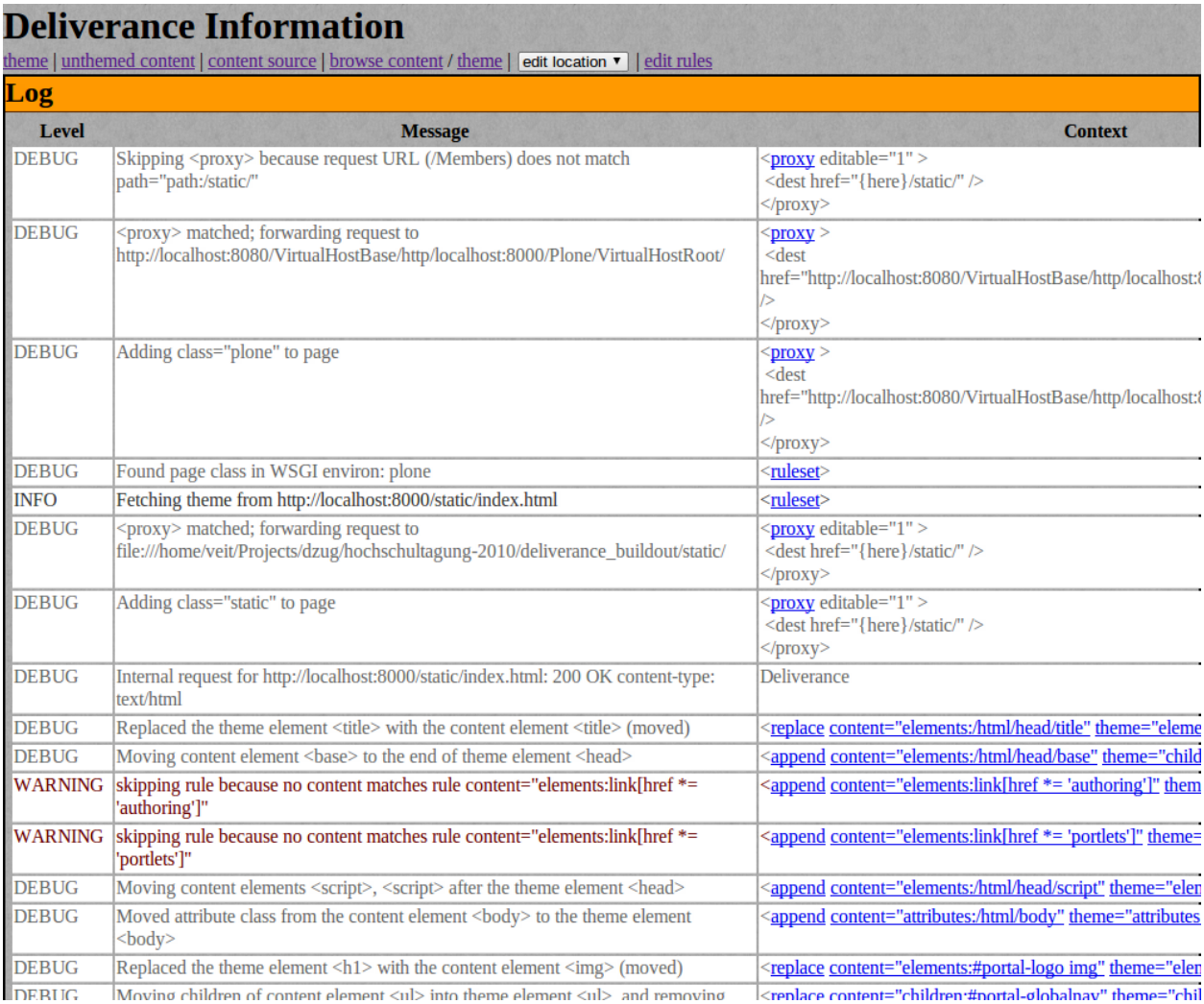
Sie können nun den Server starten mit:

```
$ ./bin/deliverance-proxy rules.xml
```

Soll die Ausgabe nicht im Terminal ausgegeben werden, kann der Deliverance-Proxy gestartet werden mit:

```
$ /usr/bin/nohup ./bin/deliverance-proxy rules.xml &
```

Anschließend können Sie sich unter `http://localhost:8000/.deliverance/login` anmelden und nun jeder URL `?deliv_log` anhängen um das Deliverance-Log am unteren Rand der Seite zu erhalten. Das Log beschreibt detailliert, wie die Seite transformiert wurde einschließlich der Informationen über den Proxy und jede Anfrage. Zudem können Sie sich den Quellcode anschauen, die Regeln bearbeiten etc. Diese Angaben erleichtern das Schreiben der Transformationsregeln ungemein.



Deliverance Information

theme | unthemed content | content source | browse content / theme | edit location ▾ | edit rules

Log

Level	Message	Context
DEBUG	Skipping <proxy> because request URL (/Members) does not match path="path:/static/"	<proxy editable="1" /> <dest href="{here}/static/" /> </proxy>
DEBUG	<proxy> matched; forwarding request to http://localhost:8080/VirtualHostBase/http/localhost:8000/Plone/VirtualHostRoot/	<proxy > <dest href="http://localhost:8080/VirtualHostBase/http/localhost:8 /> </proxy>
DEBUG	Adding class="plone" to page	<proxy > <dest href="http://localhost:8080/VirtualHostBase/http/localhost:8 /> </proxy>
DEBUG	Found page class in WSGI environ: plone	<ruleset>
INFO	Fetching theme from http://localhost:8000/static/index.html	<ruleset>
DEBUG	<proxy> matched; forwarding request to file:///home/veit/Projects/dzug/hochschultagung-2010/deliverance_buildout/static/	<proxy editable="1" /> <dest href="{here}/static/" /> </proxy>
DEBUG	Adding class="static" to page	<proxy editable="1" /> <dest href="{here}/static/" /> </proxy>
DEBUG	Internal request for http://localhost:8000/static/index.html: 200 OK content-type: text/html	Deliverance
DEBUG	Replaced the theme element <title> with the content element <title> (moved)	<replace content="elements:/html/head/title" theme="eleme
DEBUG	Moving content element <base> to the end of theme element <head>	<append content="elements:/html/head/base" theme="child
WARNING	skipping rule because no content matches rule content="elements:link[href *= 'authoring']"	<append content="elements:link[href *= 'authoring']" them
WARNING	skipping rule because no content matches rule content="elements:link[href *= 'portlets']"	<append content="elements:link[href *= 'portlets']" theme=
DEBUG	Moving content elements <script>, <script> after the theme element <head>	<append content="elements:/html/head/script" theme="eler
DEBUG	Moved attribute class from the content element <body> to the theme element <body>	<append content="attributes:/html/body" theme="attributes
DEBUG	Replaced the theme element <h1> with the content element <img> (moved)	<replace content="elements:#portal-logo img" theme="eler
DEBUG	Moving children of content element into theme element , and removing	<replace content="children:#portal-globalnav" theme="chil

Deliverance hinter Apache

Damit das `static`-Verzeichnis nicht über die Deliverance-Port-Nummer ausgeliefert wird, genügt es nicht, eine entsprechende Apache `RewriteRule` zu schreiben. Der Apache muss darüberhinaus konfiguriert werden mit `ProxyPreserveHost On`. Dies sendet den Host-Namen, den der Apache vom Client erhält, an Deliverance weiter. Und sobald der Deliverance-Proxy den Host-Namen erhalten hat, verhält er sich wie gewünscht.

Transformationsregeln

Regeln

```
<rule class="static" />
<rule class="plone" suppress-standard="true">
```

`<rule>` definiert eine Reihe von Transformtionen.

Es werden sowohl page classes als auch request/response matching unterstützt.

Mit *page classes* lässt sich eine `rule class` einem bestimmten `patch`, einer bestimmten `domain` oder einem bestimmten `response-header` zuweisen, z.B:

```
<rule class="news-section">
    <theme href="/static/news.html" />

<match path="regex:^/news" class="news-section" />
```

Weitere Informationen hierzu erhalten Sie im Abschnitt match and page classe.

`suppress-standard="true"` Deliverance kommt üblicherweise mit einer Reihe von Aktionen, die das Kopieren des Titels oder eines Skripts erlauben. Diese sind:

```
<rule>
    <replace content="children:/html/head/title"
             theme="children:/html/head/title" nocontent="ignore" />
    <append content="elements:/html/head/link"
            theme="children:/html/head" nocontent="ignore" />
    <append content="elements:/html/head/script"
            theme="children:/html/head" nocontent="ignore" />
    <append content="elements:/html/head/style"
            theme="children:/html/head" nocontent="ignore" />
</rule>
```

Diese Regeln können unterbunden werden mit dem `suppress-standard="true"`-Attribut.

Theme

```
<theme href="/static/index.html" />
```

`<theme>` definiert das Thema, das Sie verwenden in Form einer URL.

Aktionen

<replace> ersetzt ein Element aus `theme` durch ein Element aus `content`.

Die folgende Aktion ersetzt z.B. den Titel des *Themes* durch denjenigen aus Plone:

```
<replace content='/html/head/title' theme='/html/head/title' />
```

<append> fügt ein Element aus `content` am Ende eines Elements aus `theme` ein.

Die folgende Aktion hängt z.B. die `base`-Url aus Plone an die `head`-Angaben des *Theme*. Dies gewährleistet, dass die Links aus Plone weiterhin funktionieren:

```
<append content='/html/head/base' theme='children:/html/head' />
```

<prepend> fügt ein Element aus `content` am Anfang eines Elements aus `theme` ein.

Die folgende Aktion stellt z.B. das Navigationsportlet aus Plone an den Anfang der rechten Spalte des Theme:

```
<prepend content='dl.portletNavigationTree' theme='children:#rightbar' />
```

<drop> entfernt das Element aus dem `content` oder `theme`.

Die folgende Aktion entfernt z.B. das User-Icon von Plone:

```
<drop content='#user-name img' />
```

Selektoren

CSS3-Selektoren Jede Aktion beruht auf der Auswahl der Elemente des Theme und des Inhalts. Die einfachste Auswahl kann anhand von CSS-Selektoren erfolgen.

XPath Es können auch XPath-Angaben als Selektoren verwendet werden. Diese beginnen immer mit `/`.

Diese beiden Selektoren verweisen immer auf Elemente. Um spezifischere Aktionen ausführen zu können, wurden daher noch die folgenden Selektoren eingeführt:

elements Der Standardselektor.

children erlaubt, Regeln auf Kindelemente der ausgewählten Elemente anzuwenden. Hiermit lassen sich auch Aktionen auf Textinhalte anwenden.

attributes Hiermit lassen sich Aktionen nur auf bestimmte Attribute der ausgewählten Elemente anwenden.

tag Dieser Selektor erlaubt, Aktionen nur auf einen Tag, nicht jedoch auf dessen Kindelemente anzuwenden.

||-Operator

Der `||`-Operator nimmt die Ergebnisse des ersten Selektors, sofern vorhanden. Andernfalls nimmt er die Ergebnisse des zweiten Selektrors. So verwendet z.B. die folgende Aktion alle Elemente der ID `content`; sind in `content` jedoch keine Elemente vorhanden, werden die Kindelemente von `<body>` verwendet:

```
content="#content || children:body"
```

if-content

Alle Aktionen können das Attribut `if-content` erhalten womit die Aktion nur ausgeführt wird wenn die Bedingung erfüllt ist, z.B.:

```
<replace if-content='body.section-news' content='children:dl.portletEvents dt.
↪portletHeader a' theme='children:#rightbar h2' />
```

Dem zu überprüfenden Wert kann auch `not:` vorangestellt werden.

Externe Inhalte einbinden – Mashup

Deliverance erlaubt auch das Einbinden von externen Quellen. Hierzu wird das `href`-Attribut für eine Aktion verwendet, z.B.:

```
<append href="http://twitter.com/plone"
        content="#timeline"
        theme='#rightbar' />
```

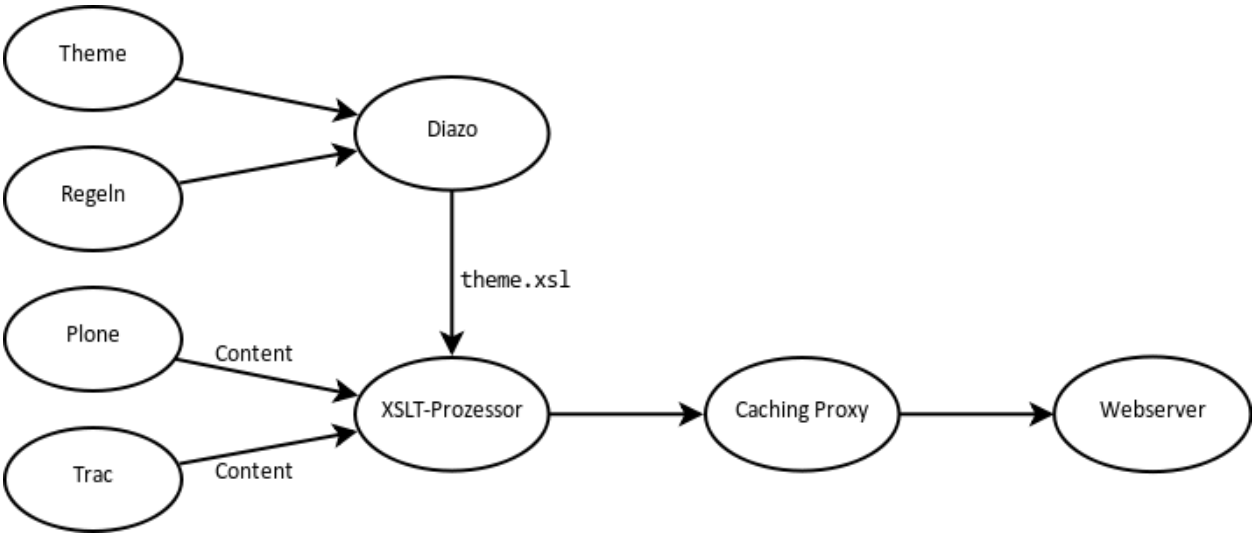
Somit ist Deliverance nicht nur für das Theming beliebiger Webanwendungen geeignet, es kann auch das Mashup verschiedener Inhalte von Webanwendungen übernehmen.

3.14.2 plone.app.theming

Einführung

plone.app.theming integriert den *Diazo*-XSLT-Proxy in Plone. Einige Vorteile von `plone.app.theming` sind:

- Web-Designer müssen kein Vorwissen in bezug auf Plone und Python mitbringen;
- Standardbibliotheken und -werkzeuge können verwendet werden;
- Auch eine bereits existierende Webgestaltung, z.B. von Open Source Web Design, kann einfach verwendet werden.



Im Vergleich zu *Deliverance* weist `plone.app.theming` folgende Unterschiede auf:

- Es wird nur das XSLT-Subset von Deliverance verwendet.
- Die Antwortzeiten von Diazo sind schneller, da nicht erst zur Laufzeit eine XSLT-Datei kompiliert wird.
- Es gibt nur XPath- und CSS3-Selektoren:

Diazo	Deliverance
`replace`	`replace`
`copy`	`replace` + Kindselektortyp
`before`	`prepend`
`after`	`append`
`prepend`	`prepend` + Kindselektortyp

`prepend` + XPATH + `@id`
oder
`prepend` + XPATH + `@class`
	`prepend` + Kindselektortyp

`append`	`append` + Kindselektortyp
`drop`	`drop`

Bedingte Regeln

- Es lassen sich Bedingungen für die Aktionen angeben, z.B.:

```
<drop if-content="not(//*[@class='portlet'])" css:theme="#portlet-wrapper" />
```

- Darüber hinaus kann eine Bedingung auch auf mehrere Aktionen angewendet werden, z.B.:

```
<rules css:if-content="#personal-bar">
    <append css:theme="#header-box" css:content="#user-prefs"/>
    <append css:theme="#header-box" css:content="#logout"/>
</rules>
```

Und auch verschachtelte Bedingungen sind mögich, z.B.:

```
<rules if-content="condition1">
    <rules if-content="condition2">
        <copy if-content="condition3" css:theme="#a" css:content="#b"/>
    </rules>
</rules>
```

oder:

```
<copy if-content="(condition1) and (condition2) and (condition3)" css:theme="#a"␣
↪css:content="#b"/>
```

- Auch mehrere `Themes` lassen sich durch Bedingungen unterscheiden, z.B.:

```
<theme href="theme.html"/>
<theme href="news.html" css:if-content="body.section-news"/>
<theme href="members.html" css:if-content="body.section-members"/>
```

oder:

```
<rules css:if-content="body.section-news">
    <theme href="news.html"/>
    <copy css:content="h2.articleheading" css:theme="h1"/>
</rules>
```

- Es können auch Pfadbedingungen angegeben werden, z.B.:

```
<theme href="news.html" if-path="/news"/>
```

  Endet die Pfadbedingung mit `/`, so wird die Regel nur angewendet, wenn die URL an dieser Stelle endet, also z.B.:

```
<theme href="news.html" if-path="/news/"/>
```

  wird auf `/Plone/news` oder `/Plone/news/` angewendet, nicht jedoch auf `/Plone/news/news1.html`.

- Auch externe Inhalte können eingebunden werden, z.B. in einem Mashup:

```
<append css:content="#timeline"
        href="http://www.heise.de/newsticker/classic/"
        css:theme="#rightbar" />
```

Reihenfolge

Die Regeln werden nicht nacheinander abgearbeitet sondern in folgender festgelegter Reihenfolge:

1. `<before />`
2. `<drop />`
3. `<replace />`
4. `<prepend />`
5. `<after />`

Dies führt meines Erachtens zu einer unnötigen Komplexität bei der Analyse, welche Regeln in welcher Reihenfolge abgearbeitet werden.

- Zudem bringt Diazo keine Debug-Konsole mit, wodurch diese Analyse deutlich erschwert wird.

Installation

`plone.app.theming` lässt sich einfach mit Buildout installieren. Hierzu nehmen Sie folgende Änderungen in der `buildout.cfg`-Datei vor:

```
[buildout]
...
extends =
    http://dist.plone.org/release/4.1b2/versions.cfg
    http://good-py.appspot.com/release/plone.app.theming/1.0b5

find-links =
    http://dist.plone.org/release/4.1b2/
    ...

versions = versions

...
eggs =
    ...
    plone.app.theming
```

Nun sollte Buildout problemlos durchlaufen und die Instanz neu gestartet werden können:

```
$ ./bin/buildout
...
$ ./bin/instance fg
```

Anmerkung: Während der Entwicklung empfiehlt sich, Zope im Debug-Modus laufen zu lassen da dann die Änderungen an `theme` oder `rules` sofort übernommen werden.

Theme-Transformation

Erstellen eines Themes

Wir können nun unser neues *Theme* erstellen indem wir in unserem Buildout-Projekt einen Ordner `static` erstellen und in diesem das Invention-Theme von Open Source Web Design bereitstellen:

```
$ cd plone.app.theming_buildout
$ curl -O http://www.oswd.org/files/designs/3293/Invention.zip
$ unzip Invention.zip
```

Nun erstellen wir in unserem `Invention`-Ordner noch die rules.xml-Datei mit den Angaben für die XSLT-Transformationen.

Aktivieren und Konfigurieren von `plone.app.theming`

Setzen Sie nun eine neue Plone-Site mit dem *Extension*-Profile `Diazo theme support` auf und wählen anschließend in deren *Plone Control Panel* die Konfiguration für das *Diazo theme*-Zusatzprodukt aus:

Basic settings

Enabled ändert die plone.app.theming-Transformation.

Aktivieren Sie diese Option.

Select a theme Wählen Sie ein Theme aus

Advanced settings

Hier können Sie eine eigene Regeldatei für Diazo, den Pfad zu den statischen Dateien oder die URL zu einem entfernten Server angeben. den angeben.

Rules file Der Pfad zu einer XML-Datei, die die Regeln für die Transformation enthält, also z.B.:

```
Invention/rules.xml
```

Es lassen sich auch Python-Pfade angeben, z.B.:

```
python://vs.theme/static/rules.xml
```

Absolute path prefix Verwendet Ihr Theme relative Pfade zu Bildern, CSS-Dateien oder anderen Ressourcen, kann hier ein Präfix eingegeben werden, der gewährleistet, dass diese Ressourcen an jeder Stelle der Plone-Site verfügbar sind.

Read network erlaubt die Verwendung von Regeln und Themes von entfernten Servern.

Unthemed host names Sofern es Namen von Hosts gibt, die nicht gestaltet werden sollen, können diese hier zeilenweise aufgelistet werden. Dies ist zumindest während der Entwicklung des Themes sinnvoll um die Ausgaben der ungestaltetem Website mit der gestalteten zu vergleichen.

Der Standardwert ist `127.0.0.1`

Parameter expressions Hier können Parameter definiert werden, die in den Regeln des Themes verwendet werden können, z.B. mit `$name`. Diese Parameter werden mittels TALES-Ausdrücken definiert, die entweder `string`, `number`, `boolean` oder `None` ausgeben sollten. Verfügbare Variablen sind:

- `context`
- `request`
- `portal`
- `portal_state`
- `context_state`

Je Zeile kann eine Variable definiert werden im Format `name = expression`.

Transformationsregeln

XPath

Schauen wir uns nun unsere einfache Transformationsregeln nochmals genauer an:

```
<replace theme='//*[@id="leftbar"]' content='//*[@id="content"]' />
```

Dabei mag Ihnen die Syntax von XPath zur Adressierung der Knoten des DOM zunächst kompliziert erscheinen. Erfreulicherweise gibt es jedoch Werkzeuge wie z.B. Firebug, die Ihnen diese Arbeit abnehmen:

Die aus Firebug kopierten XPath-Angaben können z.B. so aussehen:

```
//*[@id="content"]
```

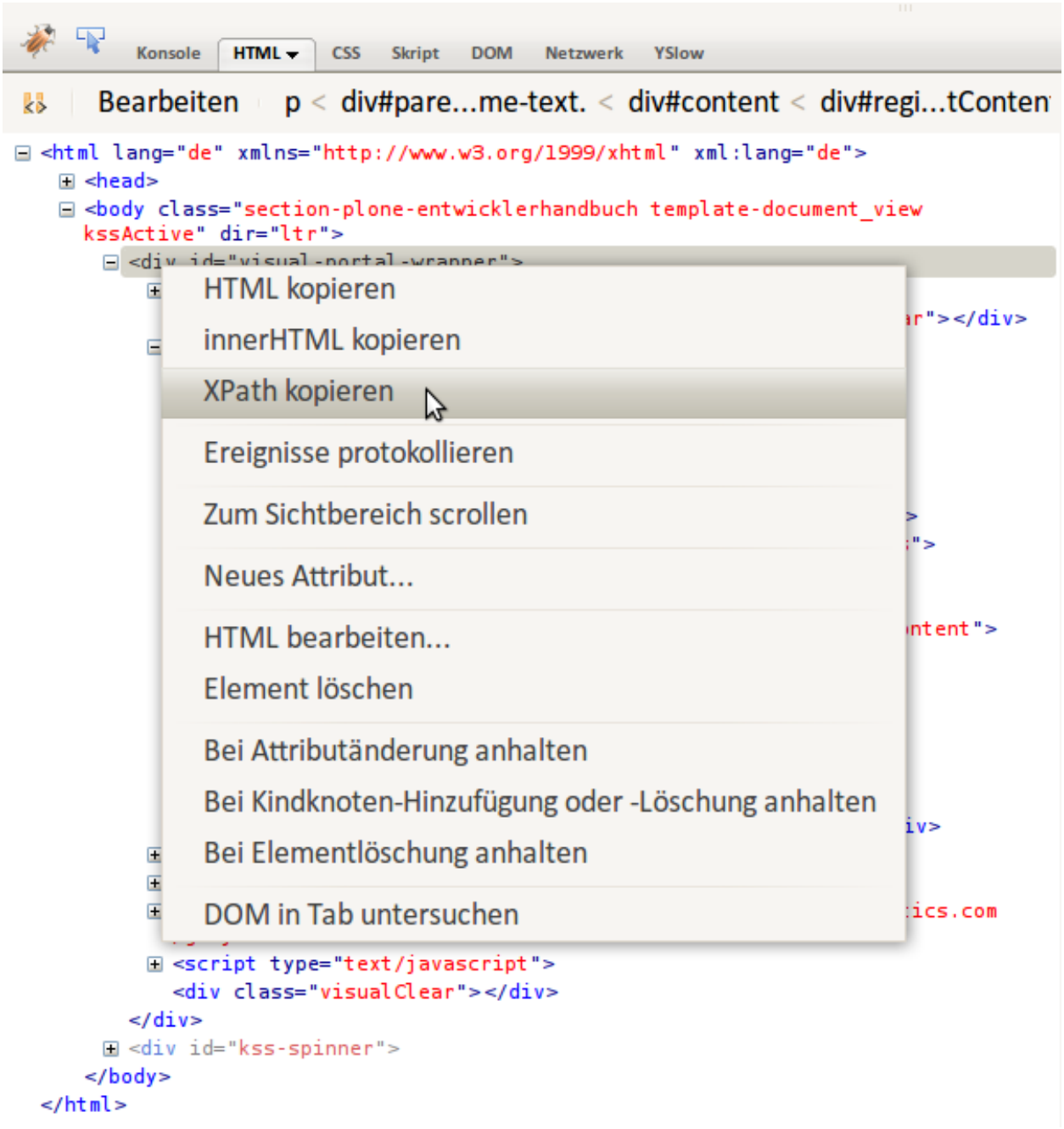
oder so, falls es sich nicht um eine id handelt sondern einen `h2`-Tag innerhalb des zweiten `div`-Tags, der wiedrum von einem `div`-Tag auf oberster Ebene umgeben ist:

```
/html/body/div/div[2]/h2
```

Neben `id` und Anzahl können Knoten auch anhand ihres Namens adressiert werden.

Weitere Informationen erhalten Sie in *XPath*.

Alternativ zu Firebug können Sie auch CSS to XPath verwenden um sich die XPath-Ausdrücke anzeigen zu lassen.


Konsole
HTML
CSS
Skript
DOM
Netzwerk
YSlow
Bearbeiten
p < div#pare...me-text. < div#content < div#regi...tConten
<html lang="de" xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
<body class="section-plone-entwicklerhandbuch template-document_view
kssActive" dir="ltr">

HTML kopieren
innerHTML kopieren
XPath kopieren
Ereignisse protokollieren
Zum Sichtbereich scrollen
Neues Attribut...
HTML bearbeiten...
Element löschen
Bei Attributänderung anhalten
Bei Kindknoten-Hinzufügung oder -Löschung anhalten
Bei Elementlöschung anhalten
DOM in Tab untersuchen
<script type="text/javascript">

</body>
</html>

Regeln

Es gibt lediglich fünf verschiedene Arten von Regeln für die Diazo-Transformationen:

<replace> ersetzt ein Element des Theme durch den Inhalt der Site.

Eine übliche Anwendung ist z.B., die `<title>` und `<base>`-Tags aus Plone zu übernehmen:

```
<replace theme="/html/head/title" content="/html/head/title" />
<replace theme="/html/head/base" content="/html/head/base" />
```

<append>, <prepend> fügt Inhalte der Website am Anfang oder ende des angegebenen Elements ein.

Hiermit können z.B. die von Plone verwalteten CSS- und Javascript-Dateien dem Theme hinzugefügt werden:

```
<append theme='/html/head' content='/html/head/script' />
<append theme="/html/head" content="/html/head/link | /html/head/style" />
```

Eine weitere, häufig verwendete Möglichkeit ist die Verwendung der `id`- und `class`-Elemente des `body`-Tags um verschiedene Bereiche der Website unterschiedlich zu gestalten:

```
<prepend theme="/html/body" content="/html/body/@class" />
```

An dem folgenden Beispiel, in dem die linke und rechte Spalte von Plone in einer Spalte zusammengefasst werden soll, wird deutlich, dass mit der Verwendung von <append> und <prepend> verhindert werden kann, dass eine Regel die Inhalte einer anderen Regel überschreibt:

```
<append content='//*[@id="portal-column-one"]/div' theme='//*[@id="rightbar"]' />
<append content='//*[@id="portal-column-two"]/div' theme='//*[@id="rightbar"]' />
```

<before>, <after> Diese sind äquivalent zu `<append />` und `<prepend />`, aber stellen den betreffenden Inhalt vor oder nach dem betreffenden Element des theme dar.

<copy> kopiert HTML-Knoten aus der Plone-Site innerhalb des Theme-Tags:

```
<copy content='//*[@id="portal-globalnav"]/li' theme='//*[@id="main-nav"]' />
```

Beachten Sie hierbei, dass jedes `<li>`-Element innerhalb von `id="portal-globalnav"` der Plone-Site in den Knoten mit der `id="main-nav"` des Themes kopiert wird.

<drop> entfernt das angegebene Element.

Diese Regel unterscheidet sich insofern von den anderen, da sie aus nur einem `content-value`-Paar besteht.

Hiermit kann die Ausgabe von Inhalten der Plone-Site unterbunden werden, z.B.das Icon des user-name-Knotens:

```
<drop content='//*[@id="user-name"]/img' />
```

Genaugenommen wird jedes `<img>`-Tag innerhalb des Elements mit der ID `user-name` entfernt.

diazotheme-Produkt

Häufig sollen die Resourcen in einem Python-Paket verwaltet werden.

Erstellen und Registrieren des Python-Pakets

Ein solches Python-Paket lässt sich einfach erstellen mit:

```
$ cd src/
$ paster create -t plone vs.diazotheme
 ...
Register Profile (Should this package register a GS Profile) [False]: True
...
```

Hiermit erstellen wir das Python Egg `vs.diazotheme` aus dem Template `plone` mit einem Profil für das *Generic Setup Tool*.

Damit dieses Egg auch der Instanz zur Verfügung steht, ändern Sie Ihre `buildout.cfg`-Datei folgendermaßen ab:

```
[buildout]
...
develop =
    ...
    src/vs.diazotheme
...
[instance]
...
eggs =
    ...
    vs.diazotheme
zcml =
    ...
    vs.diazotheme
```

Rufen Sie anschließend Ihr `./bin/buildout`-Skript erneut auf.

Erstellen Ihres Diazo-Themes

Zunächst wird hierzu in `vs.diazotheme/vs/diazotheme/configure.zcml` das Verzeichnis `Invention` als Theme registriert:

```
<plone:static
    type="theme"
    directory="Invention" />
```

Erstellen Sie hierin ein eigenes Theme oder laden eins von Open Source Web Design herunter, z.B. Invention:

```
$ cd vs.diazotheme/vs/diazotheme/
$ curl -O http://www.oswd.org/files/designs/3293/Invention.zip
$ unzip -v Invention.zip
```

Anschließend können Sie in diesem Theme die Datei `manifest.cfg` erstellen mit Titel, Beschreibung, Verweis auf die `rules.xml`-Datei etc.:

```
[theme]
title = vs.diazotheme
description = A diazo theme for veit schiele communications
rules = /++theme++vs.diazotheme/rules.xml

[theme:parameters]
ajax_load = python: 'ajax_load' in request.form
```

Für die XSLT-Transformationsregeln erstellen Sie anschließend in `Invention` die rules.xml-Datei.

Diazo-Konfiguration

Damit Plone die Regeldatei mit den XSLT-Transformationen auch liest, wird ein Profil `profiles/default/registry.xml` erstellt, das die Werte im Formular *Diazo theme settings* festlegt:

```
<registry>

    <!-- plone.app.theming settings -->
    <record field="enabled" interface="plone.app.theming.interfaces.IThemeSettings">
        <value>True</value>
    </record>
    <record field="rules" interface="plone.app.theming.interfaces.IThemeSettings">
        <value>/++theme++vs.diazotheme/rules.xml</value>
    </record>
    <record field="absolutePrefix" interface="plone.app.theming.interfaces.
↪IThemeSettings">
        <value>/++theme++vs.diazotheme</value>
    </record>
</registry>
```

rules Statt des Verweises auf die `rules.xml`-Datei in der `theme`-Ressource mit `/++theme++vs.diazotheme` könnte hier auch ein Python-Aufruf stehen:

python://vs.diazotheme/Invention/rules.xml

absolutePrefix Auch hier wurde wieder auf die registrierte Theme-Ressource verwiesen:

```
/++resource++vs.diazotheme
```

Die Angabe konvertiert relative URLs zu absoluten unter Verwendung dieses Präfixes.

CSS-Dateien registrieren

Hierzu erstellen wir die Datei `profiles/default/cssregistry.xml` mit folgendem Inhalt:

```
<?xml version="1.0"?>
<object name="portal_css">

    <!-- Set conditions on stylesheets we don't want to pull in -->
    <stylesheet
        expression="not:request/HTTP_X_THEME_ENABLED | nothing"
        id="public.css"
    />

    <!-- Add new stylesheets -->
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <stylesheet title="" authenticated="False" cacheable="True"
        compression="safe" conditionalcomment="" cookable="True" enabled="on"
        expression="request/HTTP_X_THEME_ENABLED | nothing"
        id="++theme++vs.diazotheme/css/style.css" media="" rel="stylesheet"
        rendering="link"
        applyPrefix="True"
    />

</object>
```

not:request/HTTP_X_THEME_ENABLED | nothing sorgt dafür, dass die `public.css`-Datei nicht ausgeliefert wird wenn in der HTML-Anfrage `HTTP_X_THEME_ENABLED` enthalten ist, also die Plone-Site über Diazo ausgeliefert wird.

`request/HTTP_X_THEME_ENABLED | nothing` würde umgekehrt eine Datei nur ausliefern, wenn die Anfrage durch Diazo gestellt wird.

++theme++vs.diazotheme/styles.css registriert unsere `styles.css`-Datei an Plones *Stylesheets Registry*.

applyPrefix In Plone 4 kann eine Stylesheetdatei auch mit relativen URLs geparst werden.

Schließlich erstellen wir noch die Datei `profiles/default/metadata.xml` um mit unserem `vs.diazotheme`-Produkt auch gleichzeitig das benötigte `plone.app.theming` mitzuinstallieren:

```
<metadata>
    <version>1</version>
    <dependencies>
        <dependency>profile-plone.app.theming:default</dependency>
    </dependencies>
</metadata>
```

Wenn Sie nun das Buildout-Skript erneut aufrufen, anschließend die Instanz starten und eine neue Plone-Site mit dem Profil `vs.diazotheme` erstellen, sollte die Plone-Site mit dem neuen Theme erscheinen.

Theme-Debugging

Kompilieren

Wenn Zope im Entwicklungsmodus gestartet wird, z.B. in der Konsole mit `./bin/instance fg` im Vordergrund läuft, wird das Thema bei jeder Anfrage neu kompiliert. Üblicherweise wird jedoch nur beim ersten Aufruf kompiliert und anschließend nur noch bei Änderungen im *Control Panel*. Soll auch in Entwicklungsmodus nicht bei jedem Aufruf neu kompiliert werden, so sollten Sie die Umgebungsvariable `DIAZO_ALWAYS_CACHE_RULES` aktivieren, also z.B. die Instanz starten mit:

```
$ DIAZO_ALWAYS_CACHE_RULES=1 ./bin/instance fg
```

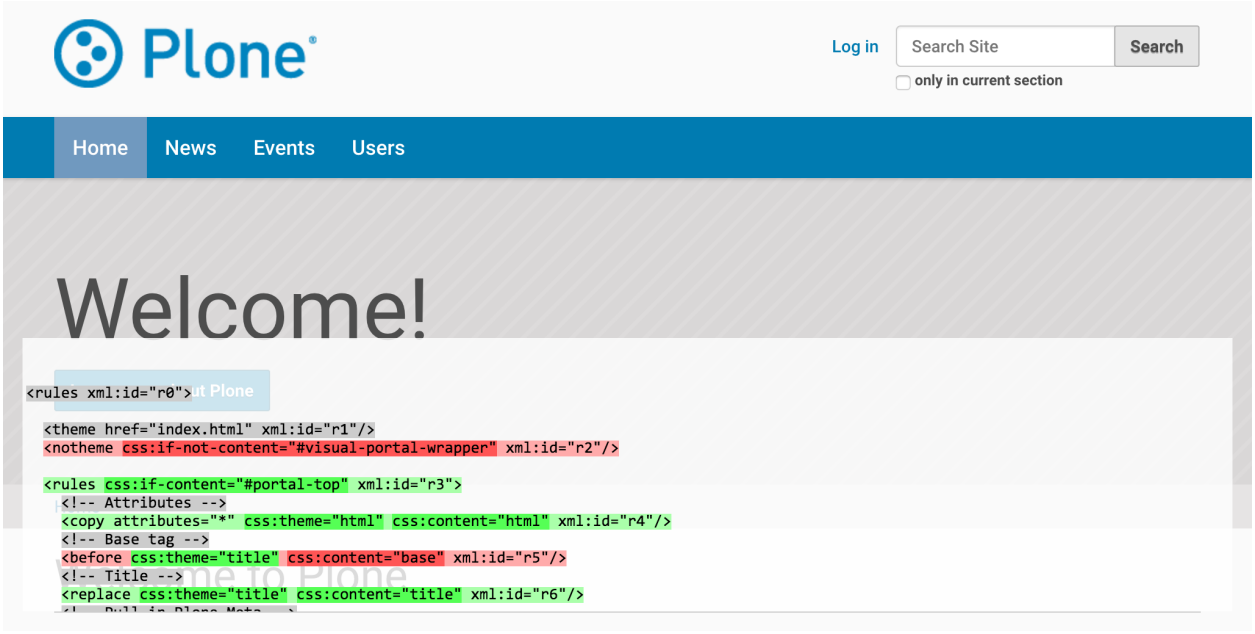
Deaktivieren des Themes

Auch im Entwicklungsmodus ist es möglich, vorübergehend das Thema zu deaktivieren indem Sie den Query String Parameter `diazo.off=1` anhängen, also z.B.:

```
http://localhost:8080/mysite/frontpage?diazo.off=1
```

Overlay

Schließlich können Sie sich in einem Overlay Ihre Diazo-Regeln anzeigen lassen, wobei *grün* bedeutet, dass die Regel angewendet wird, bei *rot* hingegen die Regel nicht angewendet wird.



Um diese Anscicht zu erhalten, ergänzen Sie die URL im Browser um `?diazo.debug=1`, also beispielsweise:

```
http://localhost:8080/mysite/front-page?diazo.debug=1
```

Der Parameter wird in Nicht-Entwicklungsmodus ignoriert.

`xsl:message`

Der Debugger untersucht jede Regel und Zustand isoliert. Wird eine Regel jedoch durch eine andere Regel außer Kraft gesetzt, zeigt er dies nicht an. Wird also z.B. wenn eine `drop`-Regel ein Element entfernt, dem eine *àppend*`-Regel etwas hinzufügen möchte, also z.B.:

```
<rules css:if-content="#visual-portal-wrapper xml:id="r0">
    <drop css:theme="content" xml:id="r1"/>
    <append css:theme="content" xml:id="r2">
        <xsl:message>Some content</xsl:message>
    </append>
    ...
</rules>
```

Dabei fügt `<xsl: message>` nichts in das Dokument selbst ein, sondern nur in das `error_log`, das ebenfalls im Overlay angezeigt wird.

Migration von collective.xdv

plone.app.theming ist eine Weiterentwicklung von collective.xdv so wie Diazo eine Weiterentwicklung von XDV ist.

Migrating der XDV-Regeln zu Diazo-Regeln

Die Syntax der Diazo-Regeln ist denen von XDV sehr ähnlich. Zunächst einmal haben sich die Namespaces geändert. Während diese in XDV noch angegeben wurden mit:

```
<rules
    xmlns="http://namespaces.plone.org/xdv"
    xmlns:css="http://namespaces.plone.org/xdv+css"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    ...
</rules>
```

lauten diese für Diazo nun:

```
<rules
    xmlns="http://namespaces.plone.org/diazo"
    xmlns:css="http://namespaces.plone.org/diazo/css"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    ...
</rules>
```

Zusätzlich haben sich einige Regeln vereinfacht, z.B.:

- `<copy />` sollte nur zum Kopieren von Attributen verwendet werden, zum Ersetzen bestehender Attribute sollte `<replace />` mit `theme-children` verwendet werden
- `<prepend />` wurde ersetzt durch `<before />``mit ``theme-children`.
- `<append />` wurde ersetzt durch `<after />` mit `theme-children`.

In der Diazo-Dokumentation finden Sie weitere Hinweise über verfügbare Rwgeln.

Änderungen in der Plone-Integration

Zum Aktualisieren einer Website, die mit collective.xdv gestaltet wurde, sind die folgenden Schritte nötig:

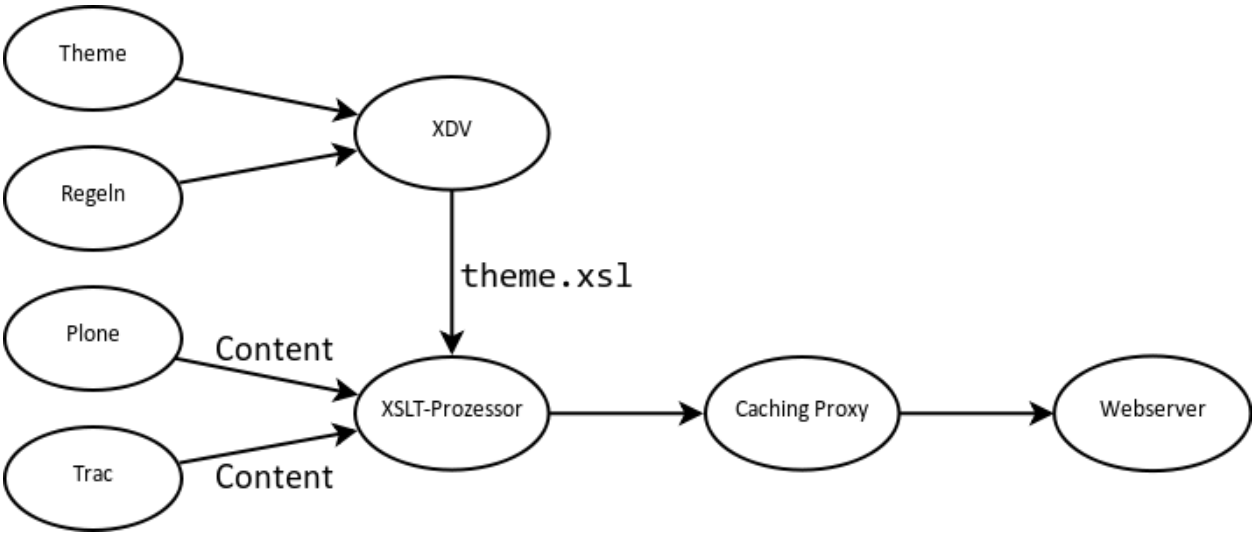
1. Deinstallieren Sie das XDV-Theme-Paket im *Quickinstaller Tool*.
2. Stoppen Sie die Instanz und entfernen collective.xdv aus Ihrem Buildout (entweder in `buildout.cfg`, einer ähnlichen Konfigurationsdatei oder in der `setup.py`-Datei unter `install_requires`).
3. Installieren Sie `plone.app.theming` und ändern Ihre Regeln wie oben beschrieben.

3.14.3 collective.xdv

Einführung

`collective.xdv` ist ein neuer Ansatz um eine Plone-Website zu gestalten Einige Vorteile von `collective.xdv` sind:

- Web-Designer müssen kein Vorwissen in bezug auf Plone und Python mitbringen;
- Standardbibliotheken und -werkzeuge können verwendet werden;
- Auch eine bereits existierende Webgestaltung, z.B. von Open Source Web Design, kann einfach verwendet werden.



Im Vergleich zu *Deliverance* weist `collective.xdv` folgende Unterschiede auf:

- Es wird nur das XSLT-Subset von Deliverance verwendet.
- Die Antwortzeiten von XDV sind schneller, da nicht erst zur Laufzeit eine XSLT-Datei kompiliert wird.
- Es gibt nur XPath- und CSS3-Selektoren:

XDV	Deliverance
`replace`	`replace`
`copy`	`replace` + Kindselektortyp
`before`	`prepend`
`after`	`append`
`prepend`	`prepend` + Kindselektortyp

`prepend` + XPATH + `@id`
oder
`prepend` + XPATH + `@class`
	`prepend` + Kindselektortyp

`append`	`append` + Kindselektortyp
`drop`	`drop`

Bedingte Regeln

- Es lassen sich Bedingungen für die Aktionen angeben, z.B.:

```
<drop if-content="not(//*[@class='portlet'])" css:theme="#portlet-wrapper" />
```

- Darüber hinaus kann eine Bedingung auch auf mehrere Aktionen angewendet werden, z.B.:

```
<rules css:if-content="#personal-bar">
    <append css:theme="#header-box" css:content="#user-prefs"/>
    <append css:theme="#header-box" css:content="#logout"/>
</rules>
```

  Und auch verschachtelte Bedingungen sind mögich, z.B.:

```
<rules if-content="condition1">
    <rules if-content="condition2">
        <copy if-content="condition3" css:theme="#a" css:content="#b"/>
    </rules>
</rules>
```

  oder:

```
<copy if-content="(condition1) and (condition2) and (condition3)" css:theme="#a"␣
↪css:content="#b"/>
```

- Auch mehrere `Themes` lassen sich durch Bedingungen unterscheiden, z.B.:

```
<theme href="theme.html"/>
<theme href="news.html" css:if-content="body.section-news"/>
<theme href="members.html" css:if-content="body.section-members"/>
```

  oder:

```
<rules css:if-content="body.section-news">
    <theme href="news.html"/>
    <copy css:content="h2.articleheading" css:theme="h1"/>
</rules>
```

- Es können auch Pfadbedingungen angegeben werden, z.B.:

```
<theme href="news.html" if-path="/news"/>
```

  Endet die Pfadbedingung mit `/`, so wird die Regel nur angewendet, wenn die URL an dieser Stelle endet, also z.B.:

```
<theme href="news.html" if-path="/news/"/>
```

  wird auf `/Plone/news` oder `/Plone/news/` angewendet, nicht jedoch auf `/Plone/news/news1.html`.

- Auch externe Inhalte können eingebunden werden, z.B. in einem Mashup:

```
<append css:content="#timeline"
        href="/http://twitter.com/plone"
        css:theme="#rightbar" />
```

Reihenfolge

Die Regeln werden nicht nacheinander abgearbeitet sondern in folgender festgelegter Reihenfolge:

1. `<before />`
2. `<drop />`
3. `<replace />`
4. `<prepend />`
5. `<after />`

Dies führt meines Erachtens zu einer unnötigen Komplexität bei der Analyse, welche Regeln in welcher Reihenfolge abgearbeitet werden. Zudem bringt XDV keine Debug-Konsole mit, wodurch diese Analyse deutlich erschwert wird.

Installation

`collective.xdv` lässt sich einfach mit Buildout installieren. Hierzu nehmen Sie folgende Änderungen in der `buildout.cfg`-Datei vor:

```
[buildout]
...
extends =
    http://dist.plone.org/release/3.3.5/versions.cfg
    lxml.cfg
    http://good-py.appspot.com/release/collective.xdv/1.0?plone=3.3.5

versions = versions

...
eggs =
    ...
    collective.xdv [Zope2.10]

[versions]
zope.i18n = 3.7.2
```

Bemerkung: Beachten Sie bitte, dass in der URL durch `1.0?plone=3.3.5` die Version von collective.xdv auf die Version 1.0 und Plone auf die Version 3.3.5 festgeschrieben wird.

Bemerkung: `[Zope2.10]` wird für Plone 3.3 oder früher benötigt um mit `ZPublisherEventsBackport` die *publication events* aus Zope 2.12 auch Zope 2.10 zur Verfügung zu stellen. Verwenden Sie Zope 2.12 und Plone 4, genügt Ihnen einfach:

```
eggs =
    ...
    collective.xdv
```

Die `lxml.cfg`-Datei sieht dann folgendermaßen aus:

```
[lxml]
parts =
   staticlxml
   pylxml

[pylxml]
recipe=zc.recipe.egg
interpreter=pylxml
eggs=
    lxml

[staticlxml]
recipe = z3c.recipe.staticlxml
egg = lxml
```

Nun sollte Buildout problemlos durchlaufen und die Instanz neu gestartet werden können:

```
$ ./bin/buildout
...
We have the distribution that satisfies 'collective.xdv[zope2.10]==1.0rc11'.
Getting required 'collective.directoryresourcepatch==1.0'
We have the distribution that satisfies 'collective.directoryresourcepatch==1.0'.
Getting required 'ZPublisherEventsBackport==1.1'
We have the distribution that satisfies 'ZPublisherEventsBackport==1.1'.
Getting required 'five.globalrequest==1.0'
We have the distribution that satisfies 'five.globalrequest==1.0'.
Getting required 'repoze.xmliter==0.1'
We have the distribution that satisfies 'repoze.xmliter==0.1'.
Getting required 'plone.transformchain==1.0b1'
We have the distribution that satisfies 'plone.transformchain==1.0b1'.
Getting required 'plone.subrequest==1.3'
We have the distribution that satisfies 'plone.subrequest==1.3'.
Getting required 'plone.app.registry==1.0b2'
We have the distribution that satisfies 'plone.app.registry==1.0b2'.
Getting required 'lxml==2.2.4'
We have the distribution that satisfies 'lxml==2.2.4'.
Getting required 'xdv==0.4b2'
We have the distribution that satisfies 'xdv==0.4b2'.
...

$ ./bin/instance fg
```

Anmerkung: Während der Entwicklung empfiehlt sich, Zope im Debug-Modus laufen zu lassen da dann die Änderungen an `theme` oder `rules` sofort übernommen werden.

Theme-Transformation

Erstellen eines Themes

Wir können nun unser neues *Theme* erstellen indem wir in unserem Buildout-Projekt einen Ordner `static` erstellen und in diesem das Invention-Theme von Open Source Web Design bereitstellen:

```
$ cd collective-xdv_buildout
$ curl -O http://www.oswd.org/files/designs/3293/Invention.zip
$ unzip Invention.zip
```

Nun erstellen wir in unserem `Invention`-Ordner noch die `rules.xml`-Datei mit den Angaben für die XSLT-Transformationen.

Aktivieren und Konfigurieren von `collective.xdv`

Setzen Sie nun eine neue Plone-Site mit dem *Extension*-Profile `XDV theme support` auf und wählen anschließend in deren *Plone Control Panel* die Konfiguration für das *XDV theme*-Zusatzprodukt aus:

`Enabled` ändert die XDV-Theme-Transformation.

Aktivieren Sie diese Option.

`Domains` Auf welche Domäne soll die Transformation angewendet werden?

Dabei ist zu berücksichtigen, dass auf `127.0.0.1` aus Sicherheitsgründen nie eine Theme-Transformation angewendet werden kann, sodass Sie immer wieder zu Ihrer Site zurückkehren können auch wenn die Transformationsregeln unbrauchbar werden sollten. Dieses Verhalten kann auch dazu genutzt werden, die CSS3-Selektoren und XPath-Anweisungen für `content` auszulesen.

Der Standardwert ist `localhost:8080`

`Theme template` Der Pfad zu einer statischen HTML-Datei. Eine relative Pfadangabe ist immer relativ zum Buildout-Verzeichnis, also z.B.:

```
Invention/index.html
```

Neben Pfadangaben im Dateisystem lassen sich hier auch Python-Pfade angeben, z.B.:

```
python://vs.xdvtheme/static/index.html
```

`Rules file` Der Pfad zu einer XML-Datei, die die Regeln für die Transformation enthält, also z.B.:

```
Invention/rules.xml
```

Auch hier lassen sich wieder Python-Pfade angeben, z.B.:

```
python://vs.xdvtheme/static/rules.xml
```

Häufig sind an dieser Stelle keine weiteren Angaben zu machen. Daher speichern wir die Einträge ab und sollten anschließend unsere einfachen XSLT-Transformationen betrachten können:

Im folgenden noch kurz die weiteren Optionen für die *XDV theme*-Konfiguration:

`Alternate themes` Hier können alternative `themes`- und `rules`-Dateien angegeben werden. Die Angabe erfolgt in der Form `path|theme|rules`.

`path` Regulärer Ausdruck

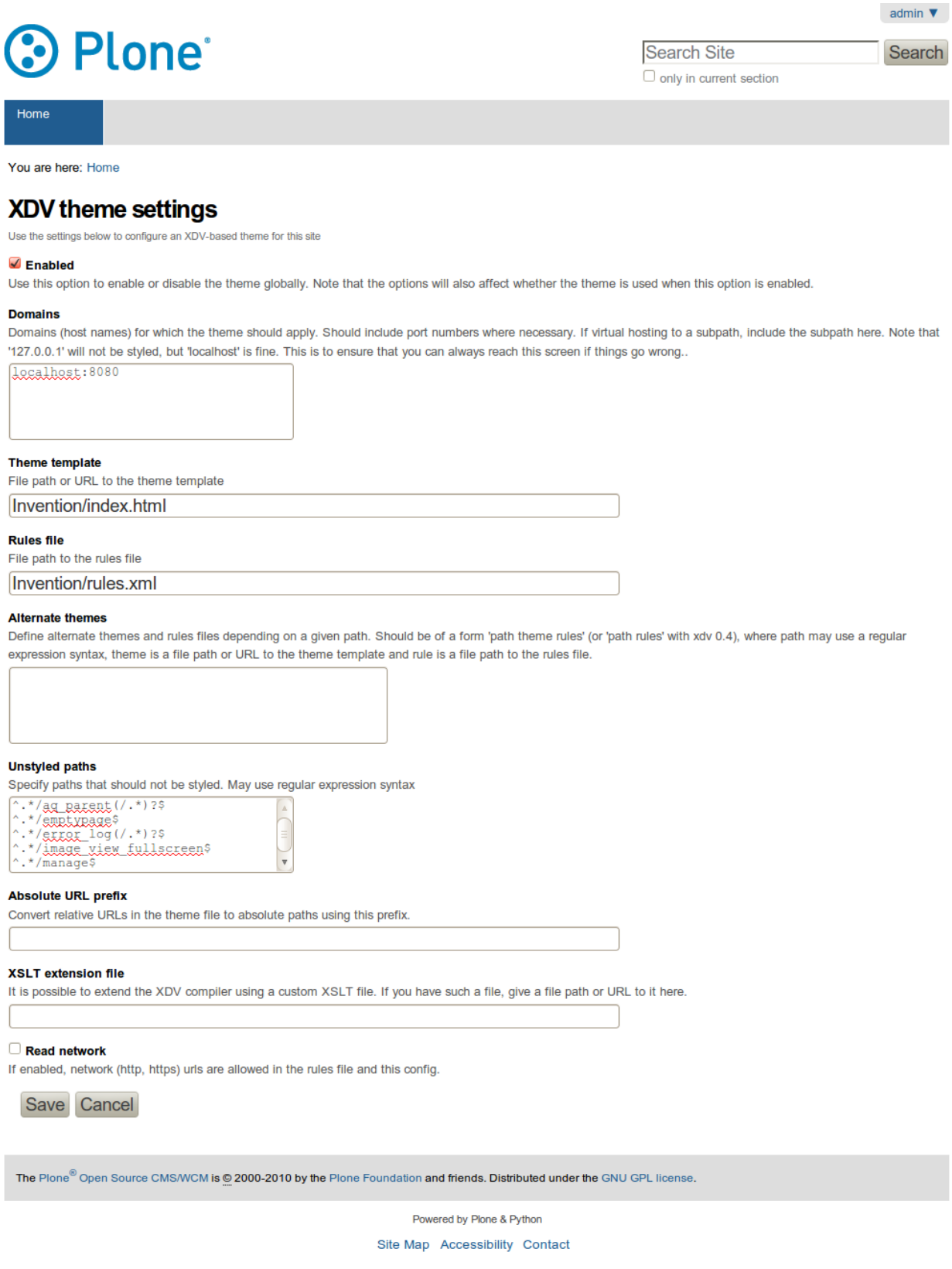
`theme` Dateipfad oder URL

`path` Pfad zu der Datei, die die XDV-Transformationsregeln bereitstellt.

`Unstyled paths` Pfade, die nicht gestaltet werden sollen wobei die Pfadangaben aus regulären Ausdrücken bestehen sollen, z.B.:

```
^.*/manage$
```

`Absolute URL prefix` konvertiert relative URLs in der `theme`-Datei zu absouten Pfaden unter Verwendung dieses Präfixes.


admin ▼
Plone
Search Site
Search
only in current section
Home
You are here: Home
XDV theme settings
Use the settings below to configure an XDV-based theme for this site
Enabled
Use this option to enable or disable the theme globally. Note that the options will also affect whether the theme is used when this option is enabled.
Domains
Domains (host names) for which the theme should apply. Should include port numbers where necessary. If virtual hosting to a subpath, include the subpath here. Note that '127.0.0.1' will not be styled, but 'localhost' is fine. This is to ensure that you can always reach this screen if things go wrong..
localhost:8080
Theme template
File path or URL to the theme template
Invention/index.html
Rules file
File path to the rules file
Invention/rules.xml
Alternate themes
Define alternate themes and rules files depending on a given path. Should be of a form 'path theme rules' (or 'path rules' with xdv 0.4), where path may use a regular expression syntax, theme is a file path or URL to the theme template and rule is a file path to the rules file.
Unstyled paths
Specify paths that should not be styled. May use regular expression syntax
^.*/ag_parent(/.*)?$
^.*/emptypage$
^.*/error_log(/.*)?$
^.*/image_view_fullscreen$
^.*/manage$
Absolute URL prefix
Convert relative URLs in the theme file to absolute paths using this prefix.
XSLT extension file
It is possible to extend the XDV compiler using a custom XSLT file. If you have such a file, give a file path or URL to it here.
Read network
If enabled, network (http, https) urls are allowed in the rules file and this config.
Save
Cancel
The Plone® Open Source CMS/WCM is © 2000-2010 by the Plone Foundation and friends. Distributed under the GNU GPL license.
Powered by Plone & Python
Site Map
Accessibility
Contact


Plone
have your punchline here
Startseite
Nachrichten
Termine
Benutzer
inventions
for a wireless world
Willkommen bei Plone
erstellt von admin — zuletzt verändert: 26.09.2010 11:19 — Historie
Herzlichen Glückwunsch! Sie haben das professionelle Open-Source Content-Management-System Plone erfolgreich installiert.
Als Präsentation anzeigen...
Wenn Sie diese Seite anstelle des von Ihnen erwarteten Inhalts sehen, hat der Betreiber dieser Website gerade erst Plone installiert. Bitte benachrichtigen Sie NICHT das Plone Team, sondern den Betreiber dieser Website.
So starten Sie!
Bevor Sie sich mit Ihrer neuen Plone-Website vertraut machen, stellen Sie bitte sicher, dass
1. Sie als Administrator angemeldet sind. (Sie müssten im Menü oben rechts den Eintrag 'Konfiguration' finden.)
2. Sie den E-Mail-Dienst konfiguriert haben. (Plone benötigt einen SMTP-Server zur Benutzerregistrierung und um Benutzern die Möglichkeit zu geben, ein vergessenes Passwort neu zu setzen.)
3. Sie wissen, welche Sicherheitseinstellungen für Ihre Website gültig sind. (Das betrifft zum Beispiel die Selbstregistrierung und das Setzen von Passwörtern)
Machen Sie sich mit Plone vertraut!
Anschließend empfehlen wir Ihnen Folgendes:
• Lesen Sie, welche neuen Funktionen Plone enthält (in Englisch),
• Lesen Sie die Dokumentation (in Englisch), insbesondere die grundlegenden Kapitel und die Empfehlungen zur Server-Konfiguration.
• Lernen Sie die grundlegenden Leistungsmerkmale von Plone kennen.
• Lesen Sie das deutschsprachige Plone-Benutzerhandbuch.
• Nutzen Sie das deutschsprachige Plone-Entwicklerhandbuch, wenn Sie Erweiterungen für Plone programmieren möchten.
• Entdecken Sie die verfügbaren Erweiterungen für Plone.
• Lesen oder abonnieren Sie die englischsprachigen oder die deutschsprachigen Mailinglisten.
Individualisieren Sie Plone!
Plone kann sehr individuell konfiguriert werden. Hier einige Beispiele:
• Wählen Sie unter den installierten Designs ein neues aus, oder installieren Sie eins der verfügbaren Designs von plone.org. (Bitte stellen Sie sicher, dass das Design mit der Plone-Version, die Sie installiert haben, kompatibel ist)
• Bestimmen Sie die Arbeitsabläufe in Ihrer Website. (Standardmäßig ist ein Arbeitsablauf für öffentliche Websites eingestellt, wenn Sie Plone als geschlossenes Intranet betreiben wollen, können Sie den Arbeitsablauf entsprechend ändern.)
• Standardmäßig bearbeiten Sie die Inhalte mit einem visuellem Texteditor. (Wenn Sie eine textbasierte Syntax oder Wiki-Markup bevorzugen, können Sie dies in den Einstellungen für Textauszeichnung auswählen)
• Weitere Optionen stehen Ihnen in der Website-Konfiguration zur Verfügung.
Sagen Sie uns, wie Sie Plone nutzen!
Haben Sie mit Plone etwas Interessantes vor? Möchten Sie eine große Website betreiben,oder haben Sie einen außergewöhnlichen Anwendungsfall? Bieten Sie als Unternehmen Lösungen auf Basis von Plone an?
• Tragen Sie Ihr Unternehmen in die Liste der Plone Dienstleister ein.
• Tragen Sie Ihre Website in die Liste der Plone Websites ein. (Entdecken Sie, welche Websites bereits mit Plone betrieben werden!)
• Beschreiben Sie in einer Fallstudie Ihr Projekt und Ihren Kunden.
Lernen Sie mehr über die Software-Architektur!
Plone ist eine Anwendung für den Zope Applikationsserver und wurde in der objektorientierten Programmiersprache Python entwickelt. Mehr über diese Technologien erfahren Sie:
• auf der Website der Plone Community
• auf der Website für den Zope Applikationsserver
• auf der Website der Python-Community.
Spenden Sie an die Plone Foundation!
Zahllose engagierte Personen und Unternehmen haben Plone möglich gemacht. Die Plone Foundation:
• ...schützt und unterstützt Plone,
• ...ist eine gemeinnützige Organisation nach dem US-Gesetz 501(c)(3) und
• ...kann Spendenquittungen ausstellen.
Helfen Sie mit!
Danke, dass Sie Plone einsetzen. Wir hoffen, dass Sie begeistert sein werden!
Ihr Plone-Team
latest news
12/08/2006
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Utid anisl nec leo congue fringilla.
10/08/2006
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Utid anisl nec leo congue fringilla.
28/07/2006
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Utid anisl nec leo congue fringilla.
info@yourcompany.com
Valid XHTML | CSS
Close

XSLT extension file XDV kann erweitert werden um eine eigene XSLT-Datei. Diese kann hier als Pfadangabe oder URL angegeben werden.

Read network Ist diese Option aktiviert, können URLs für die Angabe der `rules`- und `config`-Dateien angegeben werden.

Transformationsregeln

XPath

Schauen wir uns nun unsere einfache Transformationsregeln nochmals genauer an:

```
<replace theme='//*[@id="leftbar"]' content='//*[@id="content"]' />
```

Dabei mag Ihnen die Syntax von XPath zur Adressierung der Knoten des DOM zunächst kompliziert erscheinen. Erfreulicherweise gibt es jedoch Werkzeuge wie z.B. Firebug, die Ihnen diese Arbeit abnehmen:

Die aus Firebug kopierten XPath-Angaben können z.B. so aussehen:

```
//*[@id="content"]
```

oder so, falls es sich nicht um eine id handelt sondern einen `h2`-Tag innerhalb des zweiten `div`-Tags, der wiedrum von einem `div`-Tag auf oberster Ebene umgeben ist:

```
/html/body/div/div[2]/h2
```

Neben `id` und Anzahl können Knoten auch anhand ihres Namens adressiert werden.

Weitere Informationen erhalten Sie in *XPath*.

Alternativ zu Firebug können Sie auch CSS to XPath verwenden um sich die XPath-Ausdrücke anzeigen zu lassen.

Regeln

Es gibt lediglich fünf verschiedene Arten von Regeln für die XDV-Transformationen:

<replace> ersetzt ein Element des Theme durch den Inhalt der Site.

Eine übliche Anwendung ist z.B., die `<title>` und `<base>`-Tags aus Plone zu übernehmen:

```
<replace theme="/html/head/title" content="/html/head/title" />
<replace theme="/html/head/base" content="/html/head/base" />
```

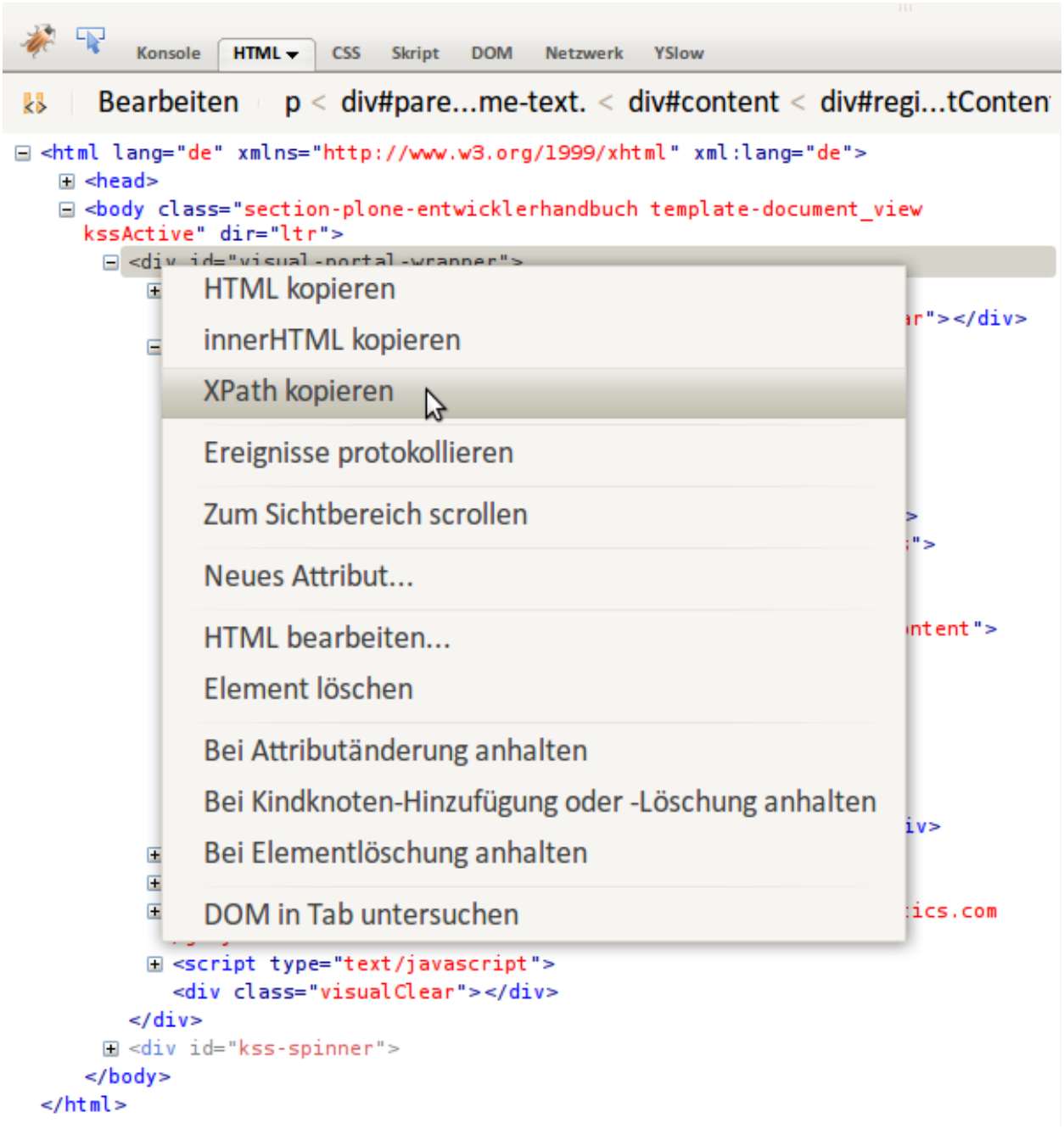
<append>, <prepend> fügt Inhalte der Website am Anfang oder ende des angegebenen Elements ein.

Hiermit können z.B. die von Plone verwalteten CSS- und Javascript-Dateien dem Theme hinzugefügt werden:

```
<append theme='/html/head' content='/html/head/script' />
<append theme="/html/head" content="/html/head/link | /html/head/style" />
```

Eine weitere, häufig verwendete Möglichkeit ist die Verwendung der `id`- und `class`-Elemente des `body`-Tags um verschiedene Bereiche der Website unterschiedlich zu gestalten:

```
<prepend theme="/html/body" content="/html/body/@class" />
```


Konsole
HTML
CSS
Skript
DOM
Netzwerk
YSlow
Bearbeiten
p < div#pare...me-text. < div#content < div#regi...tConten
<html lang="de" xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
<body class="section-plone-entwicklerhandbuch template-document_view
kssActive" dir="ltr">

HTML kopieren
innerHTML kopieren
XPath kopieren
Ereignisse protokollieren
Zum Sichtbereich scrollen
Neues Attribut...
HTML bearbeiten...
Element löschen
Bei Attributänderung anhalten
Bei Kindknoten-Hinzufügung oder -Löschung anhalten
Bei Elementlöschung anhalten
DOM in Tab untersuchen
<script type="text/javascript">

</body>
</html>

An dem folgenden Beispiel, in dem die linke und rechte Spalte von Plone in einer Spalte zusammengefasst werden soll, wird deutlich, dass mit der Verwendung von <append> und <prepend> verhindert werden kann, dass eine Regel die Inhalte einer anderen Regel überschreibt:

```
<append content='//*[@id="portal-column-one"]/div' theme='//*[@id="rightbar"]' />
<append content='//*[@id="portal-column-two"]/div' theme='//*[@id="rightbar"]' />
```

<before>, <after> Diese sind äquivalent zu `<append />` und `<prepend />`, aber stellen den betreffenden Inhalt vor oder nach dem betreffenden Element des theme dar.

<copy> kopiert HTML-Knoten aus der Plone-Site innerhalb des Theme-Tags:

```
<copy content='//*[@id="portal-globalnav"]/li' theme='//*[@id="main-nav"]' />
```

Beachten Sie hierbei, dass jedes `<li>`-Element innerhalb von `id="portal-globalnav"` der Plone-Site in den Knoten mit der `id="main-nav"` des Themes kopiert wird.

<drop> entfernt das angegebene Element.

Diese Regel unterscheidet sich insofern von den anderen, da sie aus nur einem `content-value`-Paar besteht.

Hiermit kann die Ausgabe von Inhalten der Plone-Site unterbunden werden, z.B.das Icon des user-name-Knotens:

```
<drop content='//*[@id="user-name"]/img' />
```

Genaugenommen wird jedes `<img>`-Tag innerhalb des Elements mit der ID `user-name` entfernt.

xdvtheme-Produkt

Häufig sollen die Resourcen in einem Python-Paket verwaltet werden.

Erstellen und Registrieren des Python-Pakets

Ein solches Python-Paket lässt sich einfach erstellen mit:

```
$ cd src/
$ paster create -t plone vs.xdvtheme
 ...
Register Profile (Should this package register a GS Profile) [False]: True
...
```

Hiermit erstellen wir das Python Egg `vs.xdvtheme` aus dem Template `plone` mit einem Profil für das *Generic Setup Tool*.

Damit dieses Egg auch der Instanz zur Verfügung steht, ändern Sie Ihre `buildout.cfg`-Datei folgendermaßen ab:

```
[buildout]
...
eggs =
    ...
    vs.xdvtheme

develop =
    ...
    src/vs.xdvtheme
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
...
[instance]
...
zcml =
vs.xdvtheme
```

Rufen Sie anschließend Ihr `./bin/buildout`-Skript erneut auf.

Erstellen Ihres xdv-Theme

Zunächst wird hierzu in `vs.xdvtheme/vs/xdvtheme/configure.zcml` das Verzeichnis `Invention` als Ressource registriert:

```
<browser:resourceDirectory
    name="vs.xdvtheme" directory="Invention"  />
```

Erstellen Sie hierin ein eigenes Theme oder laden eins von Open Source Web Design herunter, z.B. Invention:

```
$ cd vs.xdvtheme/vs/xdvtheme/
$ curl -O http://www.oswd.org/files/designs/3293/Invention.zip
$ unzip -v Invention.zip
```

Für die XSLT-Transformationsregeln erstellen Sie anschließend in `Invention` die rules.xml-Datei.

XDV-Konfiguration

Damit Plone die Regeldatei mit den XSLT-Transformationen auch liest, wird ein Profil `profiles/default/registry.xml` erstellt, das die Werte im Formular *XDV theme settings* festlegt:

```
<registry>

    <!-- collective.xdv settings -->
    <record field="enabled" interface="collective.xdv.interfaces.ITransformSettings">
        <field type="plone.registry.field.Bool">
            <title>Enabled</title>
        </field>
        <value>True</value>
    </record>
    <record interface="collective.xdv.interfaces.ITransformSettings" field="domains">
        <value>
            <element>localhost:8080</element>
        </value>
    </record>
    <record interface="collective.xdv.interfaces.ITransformSettings" field="rules">
        <value>python://vs.xdvtheme/Invention/rules.xml</value>
    </record>
    <record interface="collective.xdv.interfaces.ITransformSettings" field="theme">
        <value>python://vs.xdvtheme/Invention/index.html</value>
    </record>
    <record interface="collective.xdv.interfaces.ITransformSettings" field="absolute_
↪prefix">
        <value>/++resource++vs.xdvtheme</value>
    </record>
</registry>
```

python://vs.xdvtheme/Invention/rules.xml Die Ressourcen können nicht nur als Dateipfad angegeben werden sondern auch durch einen Python-Aufruf

/++resource++vs.xdvtheme Die Angabe im Feld `absolute_prefix` konvertiert relative URLs zu absoluten unter Verwendung dieses Präfixes.

CSS-Dateien registrieren

Hierzu erstellen wir die Datei `profiles/default/cssregistry.xml` mit folgendem Inhalt:

```
<?xml version="1.0"?>
<object name="portal_css">

 <!-- Set conditions on stylesheets we don't want to pull in -->
 <stylesheet
     expression="not:request/HTTP_X_XDV | nothing"
     id="public.css"
     />

 <!-- Add new stylesheets -->

 <stylesheet title="" authenticated="False" cacheable="True"
    compression="safe" conditionalcomment="" cookable="True" enabled="on"
    expression="request/HTTP_X_XDV | nothing"
    id="++resource++vs.xdvtheme/css/style.css" media="" rel="stylesheet"
    rendering="link"
    applyPrefix="True"
    />

</object>
```

not:request/HTTP_X_XDV | nothing sorgt dafür, dass die `public.css`-Datei nicht ausgeliefert wird wenn in der HTML-Anfrage `HTTP_X_XDV` enthalten ist, also die Plone-Site über xdv ausgeliefert wird.

`request/HTTP_X_XDV | nothing` würde umgekehrt eine Datei nur ausliefern, wenn die Anfrage durch xdv gestellt wird.

++resource++vs.xdvtheme/styles.css registriert unsere `styles.css`-Datei an Plones *Stylesheets Registry*.

applyPrefix In Plone 4 kann eine Stylesheetdatei auch mit relativen URLs geparst werden.

Schließlich erstellen wir noch die Datei `profiles/default/metadata.xml` um mit unserem `vs.xdvtheme`-Produkt auch gleichzeitig das benötigte `collective.xdv` mitzuinstallieren:

```
<metadata>
    <version>1</version>
    <dependencies>
        <dependency>profile-collective.xdv:default</dependency>
    </dependencies>
</metadata>
```

Wenn Sie nun das Buildout-Skript erneut aufrufen, anschließend die Instanz starten und eine neue Plone-Site mit dem Profil `vs.xdvtheme` erstellen, sollte die Plone-Site mit dem neuen Theme erscheinen.

Tipps & Tricks

Zum Entwickeln Ihres Themes Sollen Sie die Zope-Instanz im `debug`-Modus starten, da Änderungen am Template oder den XSLT-Regeln dann sofort sichtbar werden. Und wenn die *Stylesheets Registry* die css-Dateien ebenfalls im *Debug-Modus* ausliefert, werden auch die Änderungen in diesen Dateien sofort sichtbar.

3.14.4 Diazo

Diazo ist eine Weiterentwicklung von XDV und teilt mit Deliverance die folgenden Vorteile:

- Diazo ist im Gegensatz zu plone.app.theming nicht Plone-spezifisch, sodass das Theme auch für weitere Webanwendungen wie Trac, Mailman, Wordpress etc. genutzt werden kann.
- Mit Diazo lassen sich auch einfach Mashups verschiedener Webinhalte darstellen.

Diazo hat gegenüber Deliverance die folgenden Vorteile:

- Die Regeln sind einfacher
- Die Entwicklung wird von der Plone-Community getragen.

Diazo lässt sich auf zweierlei Arten aufsetzen:

- als einfacher XSLT-Proxy
- zusammen mit WSGI-Middleware-Filtern; dann sollte beim Installieren des Diazo-Eggs zusätzlich `[wsgi]` angegeben werden.

Installation

1. Erstellen eines Buildout-Verzeichnisses:

```
$ mkdir diazo
```

2. Herunterladen der `bootstrap.py`-Datei:

```
$ cd diazo
$ curl -O http://svn.zope.org/*checkout*/zc.buildout/trunk/bootstrap/bootstrap.py
```

3. Erstellen der `buildout.cfg`-Datei:

```
[buildout]
# Adjust the version number as required. See
# http://good-py.appspot.com/release/diazo for a full list

extends = http://good-py.appspot.com/release/diazo/1.0rc4
versions = versions

parts =
    lxml
    diazo

[diazo]
recipe = zc.recipe.egg
eggs =
    diazo [wsgi]
    PasteScript
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[lxml]
recipe = z3c.recipe.staticlxml
egg = lxml
```

4. Bootstrapping des Buildout-Projekts:

```
$ python2.6 bootstrap.py
```

5. Erstellen des Buildout-Projekts:

```
$ bin/buildout
```

Dies sollte die drei Skripte `./bin/paster`, `./bin/diazocompiler` und `./bin/diazorun` erstellen.

Konfiguration

Die Konfigurationsdatei `diazo/proxy.ini` für den Proxy-Server nutzt Paste Deploy um eine WSGI-Anwendung zu erstellen:

```
[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 8000

[composite:main]
use = egg:Paste#urlmap
/static = static
/ = default

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

[pipeline:default]
pipeline = theme
           content

[filter:theme]
use = egg:diazo
rules = %(here)s/rules.xml
prefix = /static
debug = true

[app:content]
use = egg:Paste#proxy
address = http://127.0.0.1:8080/VirtualHostBase/http/127.0.0.1:8000/Plone
```

`[server:main]` Server, der mit `./bin/paster serve proxy.ini` aufgerufen werden kann.

`[composite:main]` definiert das grundlegende URL-Mapping.

`paster` liefert alles aus `/static` mit `[app:static]` aus und alles andere mit `[app:default]`.

`[app:static]` liefert das Theme unter `/static` aus dem `static`-Verzeichnis aus.

`[pipeline:default]` liefert die durch Diazo transformierten Inhalte aus `theme` und `content` als `default`.

[filter:theme] Hier wird der Pfad auf die `rules.xml`-Datei und der Präfix für alle relativen Pfade (z.B. auf CSS-Dateien) angegeben.

debug = true Hiermit wird das Theme bei jeder Anfrage neu erstellt, sodass die Entwicklung deutlich leichter fällt. Für den produktiven Betrieb sollte jedoch `debug = false` gesetzt werden um die Performance zu verbessern.

[app:content] liefert die Inhalte aus `http://127.0.0.1:8000/Plone`

Regeln

Schließlich sind noch die Transformationsregeln in `diazo/rules.xml`-Datei anzugeben. In dem hier abgebildeten Beispiel werden jedoch nur einige grundlegende Transformationen ausgeführt:

```
<rules
    xmlns="http://namespaces.plone.org/diazo"
    xmlns:css="http://namespaces.plone.org/diazo/css"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <theme href="theme/theme.html" />

  <prepend theme="//head" content="//head/base"
           nocontent="ignore" />
  <prepend theme="//head" content="//head/link"
           nocontent="ignore" />
  <prepend theme="//head" content="//head/style"
           nocontent="ignore" />
  <append theme="//head" content="//head/script"
          nocontent="ignore" />
  <append theme="//head" content="//head/meta"
          nocontent="ignore" />

  <replace css:theme="title"
           css:content="title"
           nocontent="ignore" />
  <copy css:theme="div.container"
        css:content="body > *"
        nocontent="ignore" />

</rules>
```

Theme

Wesentlicher Bestandteil eines Themes ist eine HTML-Datei, `theme/theme.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dummy title</title>
    <link rel="stylesheet"
          href="./screen.css"
          type="text/css"
          media="screen, projection" />
    <link rel="stylesheet"
          href="./print.css"
          type="text/css"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
          media="print" />
    <!--[if IE]>
    {% compress css %}
    <link rel="stylesheet"
          href="./ie.css"
          type="text/css"
          media="screen, projection" />
    {% endcompress %}
    <![endif]-->
  </head>
  <body>
    <div class="container">
      <h1>Dummy Headline</h1>
      <p>Sample content</p>
    </div>
  </body>
</html>
```

Schließlich sollten Sie den Diazo-Server starten können mit:

```
$ ./bin/paster serve --reload proxy.ini
```

Diazo-Regeln

<theme /> spezifiziert eine `theme`-Datei, z.B.:

```
<theme href="theme/theme.html" />
```

href erforderlicher Verweis auf eine HTML-Datei als relative oder absolute URL

if spezifiziert eine mögliche Bedingung für diesen Verweis. Dies ermöglicht auch die Verwendung mehrerer Themes

if-path spezifiziert einen URL-Pfad, der mit dem aktuellen Request erfüllt sein muss damit die Regel angewendet wird.

<notheme /> unterdrückt das Theming, z.B.:

```
<theme href="theme/theme.html" />
<notheme css:if-content="body.template-file_view" />
```

Es können auch mehrere `<notheme />`-Elemente angegeben werden.

Es stehen die Attribute `if`, `if-path`, `if-content` und `css:if-content` zur Verfügung.

<replace> ersetzt ein Element des Theme durch den Inhalt der Site.

Eine übliche Anwendung ist z.B., die `<title>` und `<base>`-Tags aus Plone zu übernehmen:

```
<replace theme="/html/head/title" content="/html/head/title" />
<replace theme="/html/head/base" content="/html/head/base" />
```

<append>, <prepend> fügt Inhalte der Website am Anfang oder Ende des angegebenen Elements ein.

Hiermit können z.B. die von Plone verwalteten CSS- und Javascript-Dateien dem Theme hinzugefügt werden:

```
<append theme='/html/head' content='/html/head/script' />
<append theme="/html/head" content="/html/head/link | /html/head/style" />
```

Eine weitere, häufig verwendete Möglichkeit ist die Verwendung der `id`- und `class`-Elemente des `body`-Tags um verschiedene Bereiche der Website unterschiedlich zu gestalten:

```
<prepend theme="/html/body" content="/html/body/@class" />
```

An dem folgenden Beispiel, in dem die linke und rechte Spalte von Plone in einer Spalte zusammengefasst werden soll, wird deutlich, dass mit der Verwendung von <append> und <prepend> verhindert werden kann, dass eine Regel die Inhalte einer anderen Regel überschreibt:

```
<append content='//*[@id="portal-column-one"]/div' theme='//*[@id="rightbar"]' />
<append content='//*[@id="portal-column-two"]/div' theme='//*[@id="rightbar"]' />
```

<before>, <after> Diese sind äquivalent zu `<append />` und `<prepend />`, aber stellen den betreffenden Inhalt vor oder nach dem betreffenden Element des theme dar.

<copy> kopiert HTML-Knoten aus der Plone-Site innerhalb des Theme-Tags:

```
<copy content='//*[@id="portal-globalnav"]/li' theme='//*[@id="main-nav"]' />
```

Beachten Sie hierbei, dass jedes `<li>`-Element innerhalb von `id="portal-globalnav"` der Plone-Site in den Knoten mit der `id="main-nav"` des Themes kopiert wird.

<drop> entfernt das angegebene Element.

Diese Regel unterscheidet sich insofern von den anderen, da sie aus nur einem `content-value`-Paar besteht.

Hiermit kann die Ausgabe von Inhalten der Plone-Site unterbunden werden, z.B.das Icon des user-name-Knotens:

```
<drop content='//*[@id="user-name"]/img' />
```

Genaugenommen wird jedes `<img>`-Tag innerhalb des Elements mit der ID `user-name` entfernt.

<strip> entfernt das angegebene Element aus `theme` oder `content` wobei die Kindelemente erhalten bleiben.

Beispiele:

```
<strip css:theme="#content" />
```

Das Element mit der ID `content` wird entfernt nicht jedoch dessen Kindelemente:

```
<strip css:content="#main-area .wrapper" />
<replace css:theme="#content-area" css:content="#main-area" />
```

Zunächst wird das Element in `theme` mit der ID `content-area` durch das Element in `content` mit der ID `main-area` ersetzt und dann werden alle Kindelemente von `#main-area` mit der Klasse `wrapper` abgezogen.

<merge> fügt die Werte von Elementen aus `theme` und `content` desselben Attributs zusammen, also z.B.:

```
<merge attributes="class" css:theme="body" css:content="body" />
```

Reihenfolge

Die Regeln werden nicht nacheinander abgearbeitet sondern in folgender festgelegter Reihenfolge:

1. `<before />`-Regeln, die `theme` verwenden, nicht jedoch `theme-children`.
2. `<drop />`
3. `<replace />`-Regeln, die `theme` verwenden, nicht jedoch `theme-children`.

   Soll `<drop />` auf demselben Knoten verwendet werden, muss `method="raw"` verwendet werden, also z.B.:

   ```
   <drop css:content="#viewlet-social-like"/>
   <replace method="raw"
       css:theme-children="#viewlet-social-like"
       css:content="#viewlet-social-like"
       />
   ```

4. `<strip />`
5. Regeln mit `attributes`, also z.B.:

   ```
   <merge attributes="class" css:theme="body" css:content="body" />
   ```

6. `theme-children` von `<before />`-, `<replace />`- und `<after />`-Regeln sofern keine `<replace />`-Regel für `theme` für denselben Knoten bereits früher ausgeführt wurde.
7. `<after />`-Regeln, die `theme` verwenden, nicht jedoch `theme-children`.

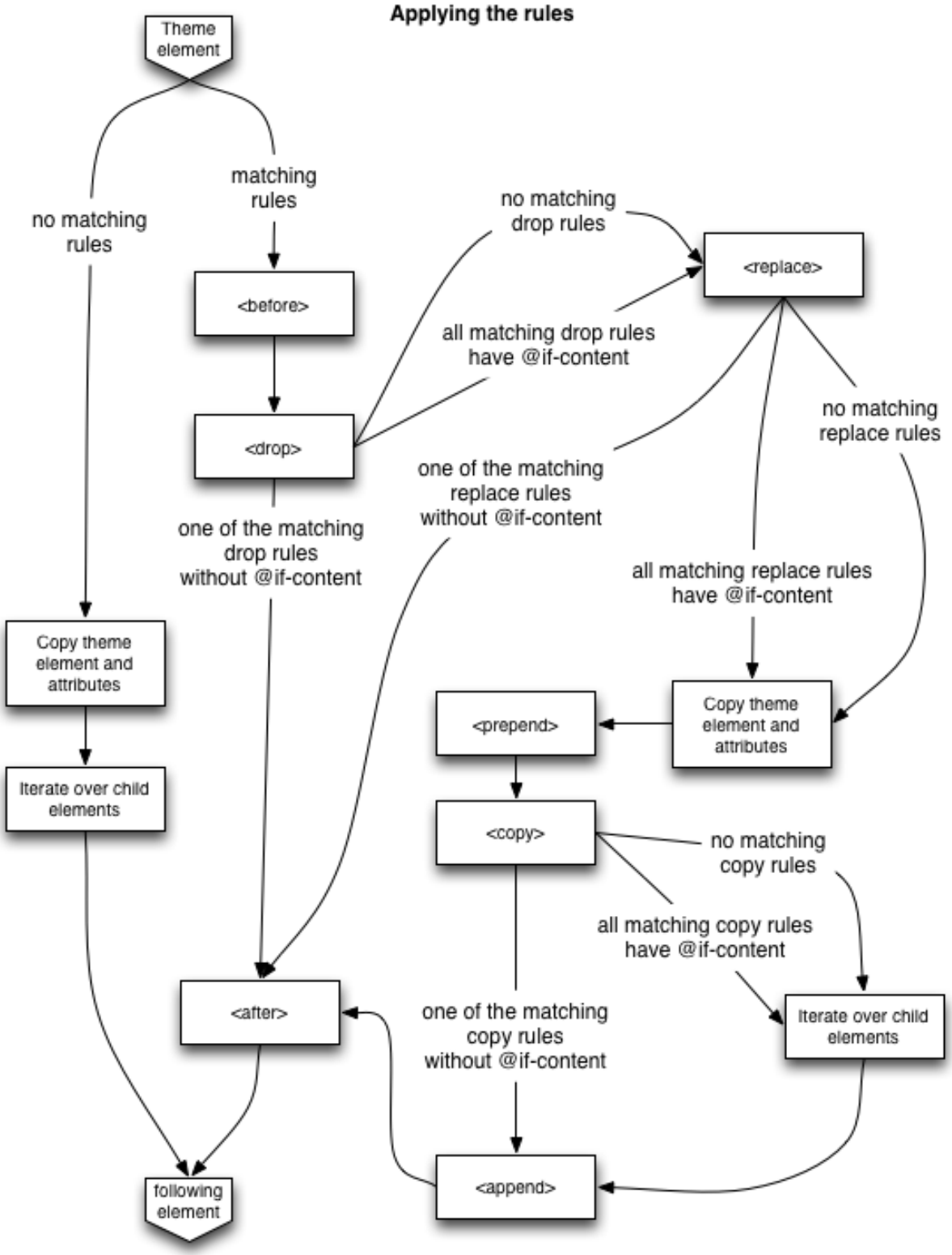
Die folgende Grafik veranschaulicht die Reihenfolge, in der die Regeln angewendet werden:

Diese festgelegte Reihenfolge führt meines Erachtens zu einer unnötigen Komplexität bei der Analyse, welche Regeln in welcher Reihenfolge abgearbeitet werden. Zudem bringt Diazo zum aktuellen Zeitpunkt noch keinen Debugger mit, wodurch die Analyse zusätzlich erschwert wird.

Keine Übereinstimmung

- Falls keine Regel auf ein bestimmtes Element des `theme` zutrifft, so wird dieses Element ignoriert.
- Eine `<replace>`-Regel, die auf ein Element in `theme`, jedoch nicht auf ein Element in `content` zutrifft, wird im `theme` entfernt.
- Eine `<copy>`-Regel, die auf ein bestimmtes Element des `theme` zutrifft, entfernt ebenfalls dieses Element wenn es kein entsprechendes Element in `content` gibt.
- Sollen die Elemente nicht entfernt sondern ein Platzhalter erhalten bleiben, kann eine bedingte Regel mit `if-content` angegeben werden, z.B.:

  ```
  <replace css:theme="#header" content="#header-element" if-content="" />
  ```


Applying the rules
Theme element
no matching rules
matching rules
<before>
<drop>
one of the matching drop rules without @if-content
no matching drop rules
all matching drop rules have @if-content
<replace>
no matching replace rules
one of the matching replace rules without @if-content
all matching replace rules have @if-content
Copy theme element and attributes
Iterate over child elements
<prepend>
Copy theme element and attributes
<copy>
no matching copy rules
all matching copy rules have @if-content
Iterate over child elements
<after>
one of the matching copy rules without @if-content
<append>
following element

Fortgeschrittene Diazo-Regeln

Bedingte Regeln

... basierend auf Knoten im Inhalt

`if-content` oder `css:if-content` spezifiziert ein Element im `content`

`if-not-content` oder `css:if-not-content` kehrt die Bedingung für ein Element im `content` um, z.B.:

```
<drop css:theme="#portlet-wrapper" css:if-not-content=".portlet"/>
```

`if-not-path` spezifiziert einen URL-Pfad, der mit dem aktuellen Request **nicht** erfüllt sein darf damit die Regel angewendet wird, z.B.:

```
<drop css:theme="#news-box" if-not-path="/news"/>
```

... basierend auf Pfadangaben im Inhalt

`if-path` oder `css:if-path` spezifiziert einen Pfad in `content`

Soll der Pfad z.B. beginnen mit `somewhere`, sieht die Regel folgendermaßen aus:

```
<copy
    if-path="/somewhere"
    css:theme="#content"
    css:content="body > *"
    />
```

Angabe eines exakter Pfad mit:

```
if-path="/somewhere/"
```

Angabe des Pfadende mit:

```
if-path="somewhere/"
```

Angabe eines Pfadbestandteils mit:

```
if-path="somewhere"
```

`if-not-path` oder `css:if-not-path` kehrt die Bedingung für einen Pfad im `content` um

... basierend auf XPath-Ausrücken

`if="$mode=''"` spezifiziert einen Knoten, der vorhanden sein muss, damit eine Regel oder ein Theme angewendet werden:

```
<drop css:theme=".warning" if="$mode = 'anon-personalbar'" />
```

`if-not="$mode=''"` spezifiziert einen Knoten, der **nicht** vorhanden sein darf, damit eine Regel oder ein Theme ausgeführt werden

Gruppierung und Verschachtelung von Bedingungen

Gruppierung von Bedingungen:

```
<rules xmlns="http://namespaces.plone.org/diazo"
       xmlns:css="http://namespaces.plone.org/diazo/css"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <rules css:if-content="#personal-bar">
        <after css:theme-children="#header-box" css:content="#user-prefs"/>
        <after css:theme-children="#header-box" css:content="#logout"/>
    </rules>
    ...
</rules>
```

Verschachtelung von Bedingungen:

```
<rules if="condition1">
    <rules if="condition2">
        <copy if="condition3" css:theme="#a" css:content="#b"/>
    </rules>
</rules>
```

entspricht:

```
<copy if="(condition1) and (condition2) and (condition3)" css:theme="#a" css:content="
↪#b"/>
```

Mehrere bedingte Themes

```
<theme href="theme.html"/>
<theme href="news.html" css:if-content="body.section-news"/>
<theme href="members.html" css:if-content="body.section-members"/>
```

Ausgabe ändern

Mit Inline-XSLT-Anweisungen lassen sich z.B. die Leerzeichen zwischen ELementen entfernen und automatische Einrückungen vornehmen:

```
<xsl:strip-space elements="*" />
<xsl:output indent="yes"/>
```

Inline XSLT-Anweisungen werden direkt innerhalb des `<rules>`-Tag angegeben und ohne Bedingungen ausgeführt.

Ändern des Themes

Inline-Markup

```
<after theme-children="/html/head">
    <style type="text/css">
        /* From the rules */
        body > h1 { color: red; }
    </style>
</after>
```

XSLT-Anweisungen

```
<replace css:theme="#details">
    <dl id="details">
        <xsl:for-each css:select="table#details > tr">
            <dt><xsl:copy-of select="td[1]/text()"/></dt>
            <dd><xsl:copy-of select="td[2]/node()"/></dd>
        </xsl:for-each>
    </dl>
</replace>
```

Ändern des Inhalts

Inline-Markup

Mit `<replace>` lässt sich auch der Inhalt modifizieren, so kann z.B. das `input`-Element mit der Klasse `searchButton` ersetzt werden durch ein `button`-Element vom Typ `submit`:

```
<replace css:content="div#portal-searchbox input.searchButton">
    <button type="submit">
        <img src="images/search.png" alt="Search" />
    </button>
</replace>
```

Entfernen leerer Tags

Ein Absatz ohne Inhalte lässt sich z.B. so entfernen:

```
<drop content="p[not(*) and (not(normalize-space()) or text() = ' ')]"/>
```

Einfügen eines Tags

Tags lassen sich z.B. am Beginn oder Ende eines Inhaltsbereichs einfügen:

```
<replace css:theme="#account a.dropdown-toggle"
         css:content="#portal-personaltools li#anon-personalbar a" />
<before css:theme-children="#account a.dropdown-toggle"
        method="raw">
    <i class="icon-user"></i>
</before>
```

Etwas aufwändiger wird es, wenn Tags innerhalb von Inhaltselementen eingefügt werden sollen:

```
<replace css:content-children="#content" css:theme-children="#content"/>
<before css:theme-children="#content">
    <div id="wrapper">
      <xsl:apply-templates css:select="#title" mode="raw"/>
      <xsl:apply-templates css:select="#description" mode="raw"/>
    </div>
</before>
<drop css:content="#title"/>
<drop css:content="#description"/>
```

Attribute ändern

Auch die Attribute eines Tags lassen sich ändern. So kann z.B. eine css- Klasse hinzugefügt werden mit:

```
<xsl:template match="ul[@id='portal-globalnav']/li/@class[contains(., 'selected')]">
    <xsl:attribute name="class"><xsl:value-of select="." /> current-menu-item</
↪xsl:attribute>
</xsl:template>
```

Auch Bilder in `content` lassen sich hiermit in einer bestimmten Größe anzeigen mit:

```
<replace css:theme="#content" css:content="#content" />
<xsl:template match="img/@src[not(contains(., '@@'))]">
    <xsl:attribute name="src"><xsl:value-of select="." />/@@/images/image/thumb</
↪xsl:attribute>
</xsl:template>
```

Dies ändert z.B.:

```
<img src="smiley.gif" class="myimage" />
```

in:

```
<img src="smiley.gif/@@/images/image/thumb" class="myimage" />
```

Text einfügen

Mit `xsl:copy` lassen sich Texte im Inhalt ergänzen, z.B.:

```
<replace css:theme="#content"
         css:content="#content" />
<xsl:template match="h2/text()">
     <xsl:copy /> - Extra text
</xsl:template>
```

Einbinden weiterer `rules`-Dateien

Mit dem `XInclude`-Protokoll lassen sich andere `rules`-Dateien einschließen, z.B.:

```
<rules
    ...
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="base.xml" />
</rules>
```

Einbinden externer Inhalte

```
<replace  css:theme-children="#navigation ul.dropdown-menu li a"
          css:content-children=".navTreeLevel2  > li > div"
          href="/sitemap" />
```

Um entfernte Inhalte einbinden zu können, muss Diazo folgendermaßen konfiguriert werden:

```
[filter:theme]
use = egg:collective.diazo.readheaders
#You can use any other Diazo middleware options here, too!
read_network = True
```

XSLT-Anweisungen

Da der von Diazo verwendete libxml2-HTMLParser Namespace-Präfixe herauskürzt, kann z.B. der FaceBook Like-Button `<fb:like></fb:like>` nicht integriert werden mit `//*[local-name()="like"]`. Stattdessen kann z.B. folgende XSL-Transformation verwendet werden:

```
<xsl:template match="activity|add-profile-tab|bookmark|comments|friendpile|like|like-
↪box|live-stream|login-button|pronoun|recommendations|serverFbml|profile-pic|user-
↪status">
  <xsl:element name="fb:{local-name()}" xmlns:fb="http://www.facebook.com/2008/fbml">
    <xsl:apply-templates select="@*|node()"/>
  </xsl:element>
</xsl:template>
```

Doctype

Üblicherweise gibt Diazo den HTML-Seiten den Doctype `XHTML 1.0 Transitional`. Um `Strict` anzugeben, sollte folgende XSLT angegeben werden:

```
<xsl:output
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
```

Es ist nicht möglich, den HTML5-Doctype mit XSLT zu setzen. Stattdessen sollte dann `<!DOCTYPE html>` gesetzt werden.

Siehe auch:

Advanced usage Englische Diazo-Dokumentation

diazo/lib/diazo/tests Die Tests von Diazo enthalten viele gebräuchliche Regeln

Diazo Snippets Library Snippets vor allem zur Nutzung von Bootstrap und Foundation

Tipps & Tricks

Pop-ups

Damit die Pop-ups für Kontakt- und Login-Formulare etc. funktionieren, sollte im Template ein `div`-Tag mit der ID `content` vorhanden sein. Dies ist notwendig da `popupforms.js` ansonsten eine Fehlermeldung ausgibt:

```
var common_content_filter = '#content>*:not(div.configlet),dl.portalMessage.error,dl.
↪portalMessage.info';
```

Deployment

Für das Deployment wird ein Proxy-Web-Server benötigt, der die XSL-Transformationen ausführen kann.

Nginx

Um das Diazo-Theme über Nginx auszuliefern, muss Nginx Moment in einer speziellen Version des html-xslt-Projekts kompiliert werden. Hierzu geben Sie folgendes an:

```
$ ./configure --with-http_xslt_module
```

Falls libxml2 und libxslt nicht an den erwarteten Stellen installiert wurden, müssen mit `--with-libxml2=<path>` und `--with-libxslt=<path>` die passenden Pfade angegeben werden.

Anschließend wird die Site entpsrechend konfiguriert:

```
location / {
    xslt_stylesheet /etc/nginx/theme.xsl
        path='$uri'
        ;
    xslt_html_parser on;
    xslt_types text/html;
    rewrite ^(.*)$ /VirtualHostBase/http/localhost/Plone/VirtualHostRoot$1 break;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    proxy_pass http://127.0.0.1:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Diazo "true";
    proxy_set_header Accept-Encoding "";
}
```

Varnish

Um Edge Side Includes (ESI) in Varnish zu ermöglichen, fügen wir in der Varnish-Konfigurationsdatei folgendes hinzu:

```
sub vcl_fetch {
    if (obj.http.Content-Type ~ "text/html") {
        esi;
    }
}
```

Apache

Apache erfordert `mod_transform` mit *html parsing*-Unterstützung. Dann kann die Konfiguration folgendermaßen aussehen:

```
NameVirtualHost *
LoadModule transform_module /usr/lib/apache2/modules/mod_transform.so
<VirtualHost *>

    FilterDeclare THEME
    FilterProvider THEME XSLT resp=Content-Type $text/html

    TransformOptions +ApacheFS +HTML +HideParseErrors
    TransformSet /theme.xsl
    TransformCache /theme.xsl /etc/apache2/theme.xsl

    <LocationMatch "/">
        FilterChain THEME
    </LocationMatch>

</VirtualHost>
```

Bemerkung: Apache ist zum aktuellen Zeitpunkt leider nicht in der Lage, Error-Responses durch die WSGI-Pipeline zu schleusen. Daher lassen sich mit Apache zum aktuellen Zeitpunkt z.B. nicht *404 Not Found*-Seiten gestalten.

Diazo Performance-Monitoring

Das Ausführen von XSLT für ein Diazo-Theme ist gewöhnlich deutlich schneller als die Auslieferung einer Plone-Seite selbst. Dabei sollte eine Anfrage weniger als eine Sekunde benötigen. 10–50ms je Anfrage an Diazo, je nach Komplexität des Themes sind dabei vollkommen normal. Wenn Sie jedoch feststellen, dass das Rendering von XSLT viel mehr Zeit verbraucht, haben Sie folgende Möglichkeiten um den Fehler genauer analysieren zu können:

1. Einfache binäre Suche

   Entfernen Sie Schritt für Schritt eine weitere Hälfte Ihres Themes und wiederholen anschließend Ihre Messung. So können Sie ggf. den zeitaufwendigen XSLT-Anweisungen auf die Spur kommen.

   Dabei werden Sie entdecken, dass Regeln wie z.B. die folgende sehr aufwändig sind:

   ```
   <before css:content="head link" css:theme="head link" />
   ```

   Diese Diazo-Regel trifft auf mehrere Knoten in einem Dokument zu. Diese Regel fügt eine Kopie aller `link`-Tags aus `content` vor jeden `link`-Tag im Theme. Stattdessen wird jedoch vermutlich das Folgende benötigt:

   ```
   <before css:content="head link" theme="/head/link[1]" />
   ```

2. Kompilieren Sie Ihr Theme mit dem Diazo-Server

   Anschließend können Sie folgende Messungen durchführen:

   ```
   $ bin/diazocompiler -r rules.xml -o compiled.xsl
   $ xsltproc --timing --repeat --html --noout compiled.xsl mypage.html
   Parsing stylesheet compiled.xsl took 1 ms
   Parsing document mypage.html took 26 ms
   Applying stylesheet 20 times took 197 ms
   ```

   Dabei erhalten wir mit `xsltproc --profile` jedoch nur Statistiken über einzelne Templates. Dies führt leider nicht immer zum gewünschten Ergebnis.

3.14.5 XMPP

Das Extensible Messaging and Presence Protocol (XMPP) ist ein offener Standard für ein Kommunikationsprotokoll für Nachrichten, der auf der Extensible Markup Language (XML) basiert.

Das Protokoll wurde ursprünglich Jabber genannt und wurde von der Jabber Open-Source-Community 1999 als Protokoll für *Instant Messaging (IM)* entwickelt. Das Protokoll war jedoch von Anfang an auf Erweiterungen ausgelegt, sodass heute auch andere Anwendungen wie *Voice over Internet Protocol (VOIP)* und File transfer signaling* darüber realisiert werden können.

Siehe auch:

- XMPP Standards Foundation

Installation und Konfiguration

ejabberd

ejabberd ist ein XMPP-Applikationsserver, der vorwiegend in Erlang geschrieben ist.

Voraussetzungen

ejabberd setzt neben GNU Make und GCC mindestens auch Expat und Erlang voraus:

```
# apt-get install libexpat1-dev erlang
```

Installation

Dies erfolgt im Buildout-Abschnitt `[ejabberd]`:

```
[ejabberd]
recipe = rod.recipe.ejabberd
erlang-path = /usr/bin
url = http://www.process-one.net/downloads/ejabberd/2.1.8/ejabberd-2.1.8.tar.gz
```

Konfiguration

Auch die Konfiguration erfolgt mit Buildout:

```
[ejabberd.cfg]
recipe = collective.recipe.template
input = templates/ejabberd.cfg.in
output = ${buildout:directory}/etc/ejabberd.cfg
xmppdomain = localhost
pubsub_max_items_node = 1000
admin_userid = admin
collaboration_allowed_subnet = 0,0,0,0
collaboration_port = 5347
component_password = secret
```

Das Template finden Sie hier: templates/ejabberd.cfg.in.

nginx

nginx

Voraussetzungen

Die `pcre`-Bibliothek wird benötigt für reguläre Ausdrücke in der `location`-Direktive und für das `ngx_http_rewrite_module`:

```
# apt-get install pcre-devel
```

Installation

Herunterladen und Installation erfolgen mit Buildout:

```
[nginx]
recipe = zc.recipe.cmmi
url = http://nginx.org/download/nginx-1.0.8.tar.gz
md5sum = 1049e5fc6e80339f6ba8668fadfb75f9
```

Konfiguration

Die Konfiguration erfolgt ebenfalls in Buildout:

```
[nginx-conf]
recipe = gocept.nginx
configuration =
  worker_processes 1;
  daemon off;
  events {
    worker_connections 1024;
  }
  http {
    proxy_read_timeout 400;
    server {
        listen       8080;
        server_name  localhost;

        location ~ ^/http-bind {
            proxy_pass http://localhost:5280;
        }

        location / {
           proxy_pass http://localhost:8081/VirtualHostBase/http/localhost:8080/Plone/
↪VirtualHostRoot/;
       }

   }
 }
```

Starten der Anwendung

Nach der Installation und Konfiguration können die Diente gestartet werden:

```
$ ./bin/ejabberdctl restart
$ ./bin/nginx  start
$ ./bin/instance start
```

supervisord

Mit supervisord lässt sich das Starten und Stoppen der Dienste automatisieren. Zur Installation von supervisord wird folgendes in der Buildout-Konfiguration eingetragen:

```
[supervisor]
recipe = zc.recipe.egg
eggs = supervisor

[supervisor-conf]
recipe = collective.recipe.template
input = ${buildout:directory}/templates/supervisord.conf.in
output = ${buildout:directory}/etc/supervisord.conf
```

Die Vorlage für die Konfigurationsdatei sieht dann folgendermaßen aus: supervisord.conf.in.

Anschließend lassen sich die Dienste starten mit:

```
$ ./bin/supervisord
```

Plone einrichten

1. Überprüfen Sie, ob `ejabberd`, `nginx` und der ZEO-Cluster gestartet sind.
2. Erstellen Sie eine neue Plone-Site mit collective.xmpp.chat.
3. Gehen Sie zu den Konfigurationseinträgen unter `http://localhost:8082/Plone/portal_registry` und editieren die `collective.xmpp.*`-Einträge.
4. Starten Sie anschließend die Instanz neu.
5. Melden Sie sich als Administrator an der Site an.
6. Erstellen Sie die gewünschten Nutzer.
7. Erstellen Sie die benötigten PubSub-Knoten durch Aufrufen des `@@setup-xmpp`-View in Ihrer Plone-Site.

Strophe.js

Strophe.js ist eine Javascript-Bibliothek die bidirektionales Streaming über HTTP-Verbindungen (BOSH) erlaubt.

Stanza Handlers

```
function chatMessageReceived {
    alert(Reviewved a message stanza);
    return true;
}
connection.addHandler(chatMessageReceived, null, 'message', 'chat');
```

Stanzas erstellen

```
var message = $msg({
            to: "someone@jabber.plone.org",
            type: "chat"
        }).c("body").t("Welcome to the plone chat");
```

ergibt z.B. folgende Stanza:

```
<message to="someone@jabber.plone.org"
         type="chat">
    <body>Welcome to the plone chat</body>
</message>
```

Plugins

Hier nur einige der wesentlichen Plugins:

muc Multi User Chat

roster Roster Management

Pubsub Publish-Subscribe protocol

Siehe auch:

- XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)
- XEP-0206: XMPP Over BOSH

Backbone.js

Backbone.js erlaubt strukturiertes Javascript mit

- Models
- Collections
- Views

Models

Erstellen eines Model

```
ChatBox = Backbone.Modeol.extend({
    initialize: function () {
        this.set ({
            'jid' : Strophe.getNodeFromJid(this.get('jid')),
            'box_id' : this.get('id'),
            'fullname' : this.get('fullname'),
        });
    }
});
```

Instantiieren eines Model

```
var box = new ChatBox({'id': hex_sha1(jid), 'jid': jid, 'fullname': name});
```

Views

Die offensichtlichsten Views für `collective.xmpp.chat` sind definiert in collective.xmpp.chat.browser.javascripts.converse.js:

- `ControlBoxView`
- `ChatRoomView`
- `ChatBoxView`

Erstellen eines eigenen Views

```
ChatBoxView = Backbone.View.extend({
    tagName: 'div',
    className: 'chatbox',
    events: {'keypress textareea.chat-textarea': keyPressed'};
    template: _.template(
        '<div class="chat-title"> {{fullname}} </div>' +
        '<div class="chat-content"></div>' +
        '<form class="sendXMPPMessage" method="post">' +
            '<textarea type="text" class="chat-area" />' +
        '</form>'),
    render: function() {
        $(this.el).html(this.template(this.model.toJSON()));
            return this;
    },
    keyPressed: function (ev) {
        ...
    }
});
```

3.14.6 Repoze

Repoze vereinigt verschiedene Technologien um WSGI und Zope zu verbinden.

WSGI Python-Standard (PEP 333), der die Kommunikation zwischen Web-Servern und Web-Anwendungen spezifiziert.

Server akzeptieren Anfragen eines Browser/Client und reichen die Daten an die Anwendungen weiter.

Sie antworten auf Anfragen, wobei sie die von Anwendungen zurückgelieferten Daten verwenden.

Anwendungen geben Antworten zurück.

Middleware Anwendung, die die *nächste* Anwendung aufruft, wobei die funktionale Anordnung eine sog. Pipeline bildet.

Repoze ermöglicht Zope in einer WSGI-Umgebung zu nutzen oder umgekehrt anderen WSGI-Anwendungen Zope-Technologien als Middleware bereitzustellen.

Dabei besteht Repoze einerseits aus einer Reimplementierung von Zope-Funktionalitäten als Python-Bibliotheken und WSGI-Middleware, andererseits aus bestehender WSGI-Middleware (Paste).

Beispiel

```
[buildout]
extends = http://dist.plone.org/release/4.1-latest/versions.cfg
parts =
   instance
   paster
   wsgiconfig

[instance]
recipe = plone.recipe.zope2instance
eggs =
    Plone
    PIL
user = admin:admin

[paster]
recipe = zc.recipe.egg
eggs =
    ${instance:eggs}
    Paste
    PasteScript
    repoze.tm2
    repoze.retry
script = paster

[wsgiconfig]
recipe = collective.recipe.template
input = inline:
    [app:zope]
    use = egg:Zope2#main
    zope_conf = ${buildout:directory}/parts/instance/etc/zope.conf

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        zope

    [server:main]
    use = egg:paste#http
    host = localhost
    port = 8000
output = ${buildout:directory}/zope2.ini
```

Die Buildout-Konfiguration nutzt umfangreich Ian Bicking's Paste, speziell PasteDeploy, das eine deklarative Syntax zum Konfigurieren von WSGI-Pipelines bereitstellt.

Zope2 kann nun einfach gestartet werden mit:

```
$ ./bin/paster serve zope2.ini
```

Anmerkung 1: infrae.wsgi scheint eine sauberere und besser dokumentierte Lösung zu sein im Vergleich zu `repoze.zope2` und dem neuen Zope2 WSGI publisher. Um `infrae.wsgi` zu verwenden, tragen Sie einfach im `[wsgiconfig]`-Abschnitt stattdessen folgendes ein:

```
use = egg:infrae.wsgi#zope2
```

Anmerkung 2: Soll ein anderer Web-Server wie z.B. gunicorn verwendet werden, so kann einfach im Abschnitt `[server:main]` stattdessen folgendes eingetragen werden:

```
use = egg:gunicorn#main
```

Siehe auch:

- [Zope-dev] Zope 2 WSGI investigation

Verzeichnisstruktur

`bin/` Verzeichnis, das alle Skripte dieses Projekts enthält, einschließlich der Skripte von Paste, Repoze, Zope, ZODB und ZEO.

Die Repoze-Skripte sind:

`addzope2user` fügt einen Nutzer mit *Manage*-Rechten in Zopes `acl_users`-Ordner hinzu; äquivalent zu `zopectl adduser`.

`debugzope2` startet einen Python-Interpreter wobei das Zope-Objekt an `app` gebunden wird.

`runzope2script` startet ein Python-Skript wobei das Zope-Objekt an `app` gebunden wird; äquivalent zu `zopectl run`.

`etc/` Verzeichnis, das die Konfigurationsdateien enthält, z.B.

`zope2.ini` Paste-Konfigurationsdatei,

`zope.conf` und `site.zcml` Zope-Konfigurationsdatei

`apache2.conf` mod_wsgi-Konfiguration für den Apache-Webserver.

`import/` Verzeichnis, um die `.zexp`-Dateien importieren zu können.

`Products/` Verzeichnis, das klassische Zope2-Produkte enthalten kann.

`var/` Verzeichnis mit den Daten der Zope-Instanz.

Konfiguration

Die Konfigurationsdatei des Repoze-Projekts `etc/zope2.ini` gliedert sich in folgende Abschnitte:

`[Default]` Dieser Abschnitt enthält globale Angaben. Zunächst ist nur folgendes angegeben:

```
debug = True
```

Hiermit wird das Repoze-Projekt üblicherweise im Debug-Modus gestartet, wobei hiervon nicht nur der Zope-Server, sondern auch weitere Paste-Middleware beeinflusst werden kann.

`[app:zope2]` In diesem Abschnitt wird die `zope2`-WSGI-Anwendung definiert.

`paste.app_factory` Paste-spezifische Angabe für den Aufruf einer WSGI-Anwendung.

Die weiteren Angaben dieses Abschnitt sind Konfigurationen von `obob`, einem Object Publishing Framework für Repoze.

`zope.conf` gibt den Ort der Zope-Konfigurationsdatei an, wobei `%(here)s` eine Paste-Konvention für das Verzeichnis ist, in der die Paste-Konfigurationsdateien liegen.

`[pipeline]` Abschnitt, der jeweils eine WSGI-Pipeline definiert. Eine Pipeline kann aus keiner oder mehr Middleware und einer Anwendung bestehen:

```
[pipeline:main]
pipeline = egg:Paste#cgitb
           egg:Paste#httpexceptions
#           egg:Paste#translogger
           egg:repoze.retry#retry
           egg:repoze.tm#tm
           egg:repoze.vhm#vhm_xheaders
           errorlog
           zope2
```

In dieser Konfiguration steht `zope2` am Ende und verweist auf die im Abschnitt `[app:zope2]` definierte Zope-WSGI-Anwendung.

`egg:Paste#cgitb` Exception handler, der die Ausgabe des tracebacks coloriert. Es können auch andere exception handler, wie z.B. evalerror, verwendet werden:

```
egg:Paste#evalerror
```

`egg:Paste#httpexceptions` gibt für bestimmte Python-Exceptions entsprechende HTTP-Exceptions aus, z.B. `404 Not Found`, `302 Redirect`, `401 Unauthorized`.

`egg:Paste#translogger` Wird der `translogger` eingeschaltet, wird das Access-Log in der Konsole ausgegeben.

`egg:repoze.retry#retry` Implementierung einer *retry policy*, wobei konfiguriert werden kann, bei welchen Fehler eine erneute Anfrage erfolgt und wie oft eine solche Anfrage wiederholt wird.

`egg:repoze.tm#tm` Implementierung der *ZODB transaction management policy*. `repoze.tm` kann auch verwendet werden um Transaktionen z.B. von relationalen Datenbanken oder von Dateisystem-Operationen zu steuern.

`egg:repoze.vhm#vhm_xheaders` Zope's Virtual Host Monster entsprechende WSGI-Middleware. Die Schreibweise unterscheidet sich zwar, die Wirkung ist jedoch dieselbe (s.a. README.txt).

`errorlog` Ersatz für Zope2's `error_log`.

[server:main] Dieser Abschnitt definiert, welcher HTTP-Server in welcher Konfiguration verwendet wird. Statt des ZServers könnte hier auch Paste, WSGIUtils oder cherrypy angegeben werden. Einen Überblick über verschiedene Server-Konfigurationen finden Sie hier: In the Fitting Room: Trying on WSGI Servers.

Deliverance

Deliverance erlaubt Gestaltungen auf Inhalte nach bestimmten Regeln anzuwenden.

1. Deliverance kann einfach in einem Repoze-Projekt installiert werden mit:

```
$ sudo easy_install Deliverance
```

2. Anschließend wird in `etc/zope2.ini` im Abschnitt `pipeline:main` deliverance hinzugefügt:

```
[pipeline:main]
pipeline = egg:Paste#cgitb
           egg:Paste#httpexceptions
#           egg:Paste#translogger
           egg:repoze.retry#retry
           egg:repoze.tm#tm
           egg:repoze.vhm#vhm_xheaders
           errorlog
           deliverance
           zope2
```

3. Dann wird ein neuer Abschnitt eingefügt:

```
[filter:deliverance]
paste.filter_app_factory = deliverance.wsgimiddleware:make_filter
theme_uri = http://www.veit-schiele.de/frontpage
rule_uri = file:///%(here)s/rules.xml
```

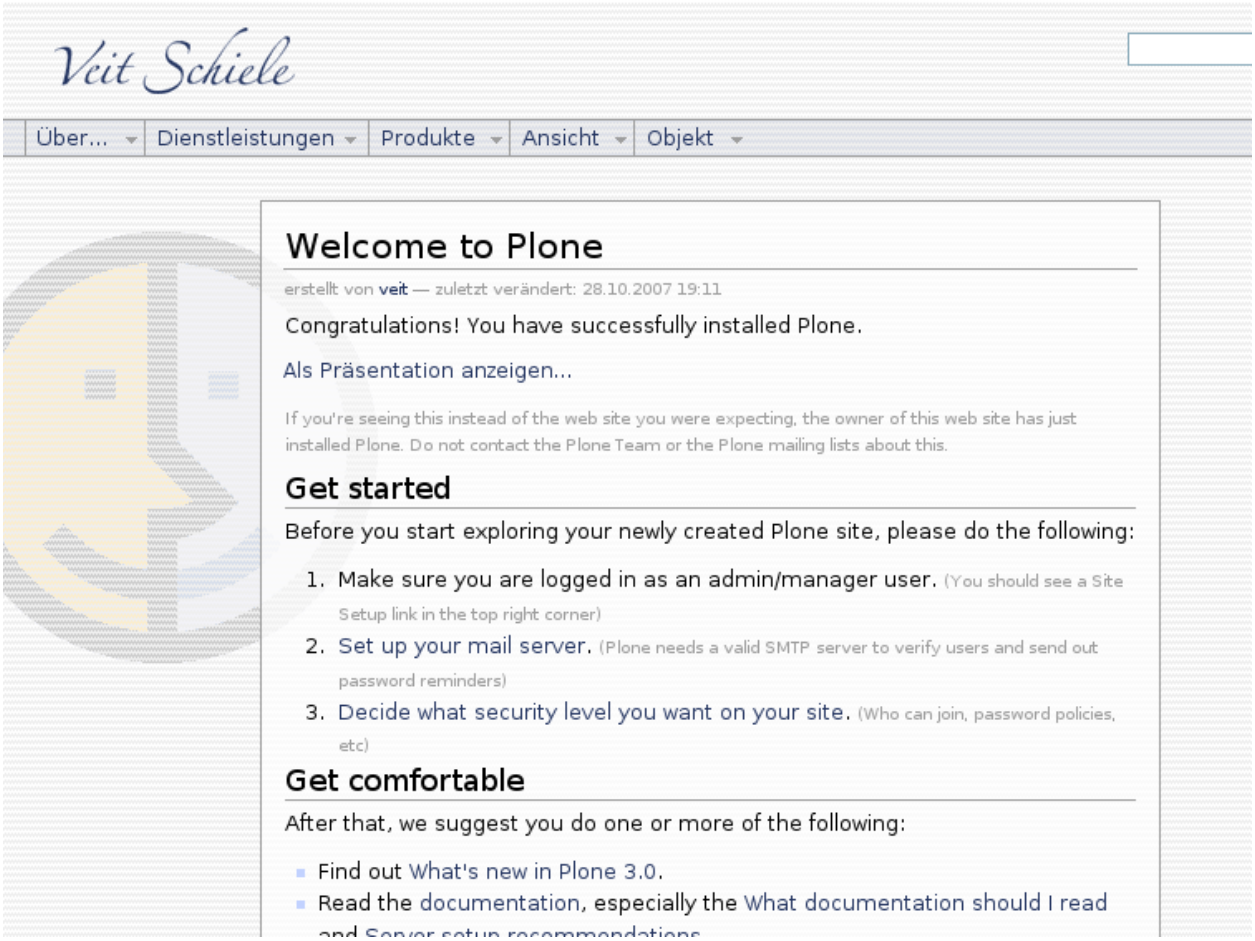
4. Schließlich wird die Datei `etc/rules.xml` mit folgendem Inhalt erstellt:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns:xi="http://www.w3.org/2001/XInclude" xmlns="http://www.plone.org/
↪deliverance">
  <prepend theme="//head" content="//head/link" nocontent="ignore" />
  <prepend theme="//head" content="//head/style" nocontent="ignore" />
  <append theme="//head" content="//head/script" nocontent="ignore" />
  <append theme="//head" content="//head/meta" nocontent="ignore" />
  <append-or-replace theme="//head"
                     content="//head/title"
                     nocontent="ignore" />
  <replace theme="//body//div[@id='content']"
           content="//body//div[@id='content']"
           nocontents="ignore" />
</rules>
```

5. Nach dem Neustart der Zope-Instanz mit:

```
$ ./bin/paster serve etc/zope2.ini
```

sollte die Plone-Site nun das Motiv meiner Website übernommen haben:


Veit Schiele
Über...
Dienstleistungen
Produkte
Ansicht
Objekt
Welcome to Plone
erstellt von veit — zuletzt verändert: 28.10.2007 19:11
Congratulations! You have successfully installed Plone.
Als Präsentation anzeigen...
If you're seeing this instead of the web site you were expecting, the owner of this web site has just installed Plone. Do not contact the Plone Team or the Plone mailing lists about this.
Get started
Before you start exploring your newly created Plone site, please do the following:
1. Make sure you are logged in as an admin/manager user. (You should see a Site Setup link in the top right corner)
2. Set up your mail server. (Plone needs a valid SMTP server to verify users and send out password reminders)
3. Decide what security level you want on your site. (Who can join, password policies, etc)
Get comfortable
After that, we suggest you do one or more of the following:
Find out What's new in Plone 3.0.
Read the documentation, especially the What documentation should I read

Tipps & Tricks

- Ab `repoze.zope2 0.2.6` kann man sich das *error log* mit `/__error_log__` anzeigen lassen (das Zope `error_log`-Objekt funktioniert nicht in Repoze-Projekten).

3.14.7 Solr

Apache Solr ist eine OpenSource Suchmaschine, die auf Sites wie Twitter Apple- and iTunes-Stores, Wikipedia und vielen anderen eingesetzt wird.

Apache Solr erlaubt nicht nur, die Inhalte verschiedener Systeme zu durchsuchen, es bietet auch weitere umfangreiche Suchfunktionen:

Facettierte Suche erlaubt die zunehmende Verfeinerung der Suche

Räumliche Suche (*Geospatial search*) anhand von Geodaten

Autovervollständigung (*suggestions*) Anhand der von Ihnen gemachten Eingaben werden Ihnen die am häufigsten gesuchten Phrasen vorgeschlagen

Rechtschreibkorrektur Falls Sie sich vertippt haben sollten, schlägt Solr Ihnen eine korrekte Schreibweise vor.

Indizierung von binären Dateien Hierzu gehören z.B. auch PDFs und MS-Office-Dokumente.

Darüberhinaus kann ein Cluster für Solr erstellt werden, wodurch sich die Last deutlich verteilen lässt.

Solr: Installation und Konfiguration

Solr lässt sich einfach mit Buildout installieren. Hierzu kann das Paket collective.recipe.solrinstance verwendet werden.

Hierzu erstellen wir die Datei `solr.cfg`, die z.B. so aussehen kann:

```
[buildout]
parts =
    solr-download
    solr-instance

[versions]
collective.recipe.solrinstance = 3.5
gocept.download = 0.9.5

[solr-download]
recipe = hexagonit.recipe.download
download-directory = parts/solr-download
strip-top-level-dir = true
url = http://ftp-stud.hs-esslingen.de/pub/Mirrors/ftp.apache.org/dist//lucene/solr/3.
↪4.0/apache-solr-3.4.0.tgz
ignore-existing = true
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
solr-instance]
recipe = collective.recipe.solrinstance
solr-location = ${solr-download:location}
host = 83.223.91.163
port = 8983
basepath = search
```

solr-location Pfad zur Installation von Solr. In unserem Fall ist dies

```
${solr-download:location}
```

host Name oder IP-Adresse des Solr-Servers.

Der Standardwert ist `localhost`:

port Der Port, an dem Solr auf Anfragen lauscht.

Der Standardwert ist `8983`.

basepath Pfad zum Solr-Service auf dem Server. Hieruas wird die endgülitge URL für den Solr-Service generiert:

```
$host:$port/$basepath
```

Der Standardwert ist `solr`.

Der Solr-Server kann nun einfach gestartet werden mit:

```
$ ./bin/solr-instance fg
```

oder:

```
$ ./bin/solr-instance start
```

Logging

```
logdir = ${buildout:directory}/var/solr
logging-template = ${buildout:directory}/templates/logging.properties.tmpl
```

logdir

```
${buildout:directory}/var/solr
```

logging-template

```
${buildout:directory}/templates/logging.properties.tmpl
```

Das `logging.properties.tmpl` kann dann z.B. so aussehen:

```
# Default global logging level:
.level= INFO
```

Konfiguration der Suche

```
max-num-results = 500
section-name = SOLR
unique-key = id
default-search-field = text
filter =
    text solr.StopFilterFactory ignoreCase="true" words="stopwords.txt"
    text solr.WordDelimiterFilterFactory generateWordParts="1" generateNumberParts="1
↪" catenateWords="0" catencatenateAll="0"
    text solr.LowerCaseFilterFactory
    text solr.RemoveDuplicatesTokenFilterFactory

index =
    name:text             type:text    stored:true
    name:title            type:text    stored:true
    name:created          type:date    stored:true required:true
    name:modified         type:date    stored:true
    name:filesize         type:integer stored:true
    name:mimetype         type:string  stored:true
    name:id               type:string  stored:true required:true
    name:relpath          type:string  stored:true
    name:fullpath         type:string  stored:true
    name:renderurl        type:string  stored:true
    name:tag              type:string  stored:true
```

max-num-results Die maximale Anzahl von Ergebnissen, die der Solr-Server ausliefern soll.

Der Standardwert ist `500`.

section-name Name des Abschnitts für die Produktkonfiguration, die für die `zope.conf` generiert wird.

Der Standardwert ist `solr`.

unique-key beschreibt ein Feld als eindeutig für alle Dokumente. Weitere Informationen hierzu erhalten Sie unter SchemaXml.

Der Standardwert ist `uid`.

default-search-field konfiguriert das Standardsuchfeld sofern kein Feld explizit angegeben wurde.

filter konfiguriert zusätzliche Filter für den Standard-Feldtyp. Jeder Filter wird in einer neuen Zeile aus einem Index und Parametern definiert. Dabei kann einer der verfügbaren Indextypen angegeben werden und als Parameter Schlüssel-Wert-Paare. Einen Überblick über die verfügbaren Filter erhalten Sie in TokenFilterFactories.

Weitere Konfigurationsmöglichkeiten von `collective.recipe.solrinstance` erhalten Sie in Supported options.

SolrIndex

SolrIndex ist ein ZCatalog Multi-Index, der Solr verwendet. Er ersetzt den Standard Volltext-Index von Plone und ermöglicht damit u.a.

- die unterschiedliche Gewichtung von Feldern
- die Verwendung von Stopwords
- die Einbeziehung von Synonymen

SolrIndex lässt sich so erweitern um

Facettierte Suche Dies erlaubt die zunehmende Verfeinerung der Suche

Autovervollständigung (suggestions) Anhand der von Ihnen gemachten Eingaben werden Ihnen die am häufigsten gesuchten Phrasen vorgeschlagen

Rechtschreibkorrektur Falls Sie sich vertippt haben sollten, schlägt Solr Ihnen eine korrekte Schreibweise vor.

Installation

SolrIndex kann einfach mit Buildout installiert werden:

```
[instance]
...
eggs =
    ...
    ...alm.solrindex
```

Anschließend sollte SolrIndex ein SolrIndex auf der Plone-Site hinzugefügt werden. Ein solcher SolrIndex kann dann mehrere ZCatalog-Indexe enthalten.

bg.solr

bg.solr stellt Views für Plone bereit, mit denen sich in Solr suchen lässt und die Ergebnisse in Plone angezeigt werden.

Falls Solr nicht über *Deliverance* oder *Diazo* angezeigt werden soll, können die Ansichten für die Solr-Suche auch einfach mit bg.solr in Plone integriert werden.

Installation

Die Installation kann in Buildout einfach erfolgen mit:

```
[buildout]
...
extensions =
   mr.developer
sources = sources
auto-checkout =
    bg.solr

[sources]
bg.solr = git https://github.com/zopyx/bg.solr

[instance]
...
eggs =
   ...
   bg.solr
```

bg.crawler

bg.crawler ermöglicht die Indexierung von Dateien und Dateibäumen im Dateisystem durch Solr.

bg.crawler erlaubt auf der Komandozeile (Command line interface, CLI), einzelne Dateien oder Dateibäume von Solr indizieren zu lassen.

Voraussetzungen

- Python 2.6 oder 2.7
- curl

Installation

In einer `virtualenv`-Umgebung lässt sich `bg.crawler` einfach installieren mit:

```
$ easy_install bg.crawler
```

Optionen

Innerhalb dieser virtuellen Umgebung lässt sich `bg.crawler` einfach aufrufen mit:

```
$ ./bin/solr-crawler --help
```

Folgende Parameter stehen Ihnen zur Verfügung:

`--solr-url` definiert die URL des Solr-Servers.

`--render-base-url` Basis-URL, die den ersten Teil von Solrs `renderurl` bildet.

`--max-depth` begrenzt die Tiefe der Ordnerhierarchie bis zu der Dateien indiziert werden sollen.

`--commit-after` Die Anzahl der Dokumente, die mit einem *commit* an Solr übergeben werden.

`--tag` Die importierten Dokumente werden mit einer bestimmten Zeichenkette getagt.

So lassen sich unterschiedliche Datenquellen auch bei einer späteren Suchanfrage in Solr noch durch unterschiedliche Tags unterscheiden.

`--clear-all` leert den Solr-Index vollständig bevor die Daten neu importiert werden.

`--clear-tag` entfernt alle Dokumente aus dem Solr-Index, die einen bestimmten Tag enthalten, bevor die Daten neu importiert werden.

`--verbose` ermöglicht ein besseres Logging.

`--no-type-check` Falls diese Option gewählt wird, wird nicht nach bestimmten Dateitypen gefiltert.

Weitere Informationen zu `bg.crawler` erhalten sie unter bg.crawler documentation.

3.15 Anhang

3.15.1 Praxisbeispiele

archetypes.schemaextender

archetypes.schemaextender erlaubt das Erweitern, Ändern und Löschen von Feldern eines Archetypes-Schema.

archetypes.schemaextender erlaubt, Archetypes- Schemas dynamisch mit Adaptern zu erweitern. Dies kann verwendet werden um neue Felder hinzuzufügen, Felder oder Fieldsets neu anzuordnen etc.

So sind dann auch drei verschiedene Adapter verfügbar:

ISchemaExtender erlaubt das Hinzufügen neuer Felder zu einem Schema.

IOrderableSchemaExtender erlaubt neue Felder hinzuzufügen und Felder neu anzuordnen, ist jedoch deutlich kostspieliger als `ISchemaExtender`.

IBrowserLayerAwareExtender verwendet `plone.browserlayer` sodass der Extender nur verfügbar ist sofern ein Layer registriert wurde. Damit lässt sich die Schemaerweiterung von Plone- Artikeltypen auf eine Site begrenzen.

ISchemaModifier erlaubt auf niedrigschwelligem Niveau die Manipulation von Schemas.

Beispiel

1. Zunächst wird `archetypes.schemaextender` als Abhängigkeit unseres Produkts in `vs/registration/configure.setup.py` registriert:

```
install_requires=[
    'setuptools',
    ...
    'archetypes.schemaextender',
],
```

2. Anschließend wird in `vs/registration/interfaces.py` das entsprechende Interface definiert:

```
from plone.theme.interfaces import IDefaultPloneLayer

class IVSRegistrationExtenderLayer(IDefaultPloneLayer):
    """A Layer Specific to VSRegistrationExtender"""
```

3. Dieser Layer wird nun registriert in `vs/registration/profiles/default/browserlayer.xml` mit:

```
<layers>
    <layer name="vs.registration"
           interface="vs.registration.interfaces.IVSRegistrationExtenderLayer" />
</layers>
```

4. Dann wird ein neues Paket hinzugefügt:

```
$ mkdir extender
$ touch extender/__init__.py
```

5. Anschließend wird es in die Konfiguration eingeschlossen indem in `vs/registration/configure.zcml` folgende Zeile hinzugefügt wird:

```
<include package=".extender" />
```

6. Nun wird der Extender selbst registriert in `vs/registration/extender/configure.zcml`:

```
<include package="archetypes.schemaextender" />
<adapter factory=".teaser.TeaserExtender"
         provides="archetypes.schemaextender.interfaces.ISchemaExtender"
         for="Products.ATContentTypes.interfaces.IATEvent" />
```

7. Nun wird die Klasse `TeaserExtender` in `vs/registration/extender/teaser.py` erstellt:

```
class TeaserExtender(object):
    """ teaser fields """

    implements(ISchemaExtender, IBrowserLayerAwareExtender)
    # bind this extender to the browser layer
    layer = VSRegistrationExtenderLayer
    fields = [TeaserField('teaserImage',
                          default=False,
                          storage = AnnotationStorage(migrate=True),
                          swallowResizeExceptions=zconf.
↪swallowImageResizeExceptions.enable,
                          pil_quality=zconf.pil_config.quality,
                          pil_resize_algo=zconf.pil_config.resize_algo,
                          max_size=config.TEASER_MAX_DIMENSION,
                          sizes=config.TEASER_SIZES,
                          widget=atapi.ImageWidget(
                              label=u"Teaser image",
                              label_msgid='label_teaser_image',
                              i18n_domain='plone',
                              ),
                           schemata='Teaser',
                           ),
            ]

    def __init__(self, context):
        self.context = context

    def getFields(self):
        return self.fields
```

Die vollständige Datei können Sie sich hier anschauen: `teaser.py`.

Feldreihenfolge ändern

1. Der hierfür notwendige Adapter wird in `vs/registration/extender/configure.zcml` konfiguriert:

```
<adapter
    factory=".extender.VSRegistrationExtender"
    provides="archetypes.schemaextender.interfaces.IOrderableSchemaExtender" />
```

2. Nun wird die Klasse `TeaserExtender` in `vs/registration/extender/teaser.py` erweitert:

```
from archetypes.schemaextender.interfaces import IOrderableSchemaExtender,␣
↪IBrowserLayerAwareExtender
...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
class TeaserExtender(object):
    implements(IOrderableSchemaExtender, IBrowserLayerAwareExtender)
    ...
    def fiddle(object, schema):
    def getOrder(self, order):
        do = order['default']

        # place teaser at the very top
        do.remove('teaser')
        do.insert(0, 'teaser')
        return order
```

Ändern eines bestehenden Schemas

1. Ein bestehendes Schema lässt sich ändern, indem zunächst ein Adapter konfiguriert wird in `vs/registration/extender/configure.zcml`:

```
<adapter
    factory=".teaser.TeaserExtender"
    provides="archetypes.schemaextender.interfaces.ISchemaModifier" />
```

2. Nun wird die Klasse `TeaserExtender` in `vs/registration/extender/teaser.py` erweitert:

```
from archetypes.schemaextender.interfaces import IOrderableSchemaExtender,␣
↪IBrowserLayerAwareExtender, ISchemaModifier
...
def fiddle(object, schema):
    schema['image'].widget.visible = {'edit':'invisible','view':'invisible'}
    return schema
```

Siehe auch:

- Plone Documentation Team: archetypes.schemaextender
- archetypes/schemaextender/usage.txt

plone.indexer

plone.indexer vereinfacht die Erstellung und Verwaltung eigener Indizees in Plone.

plone.indexer erlaubt das Erstellen von Adaptern zum Indizieren des ZCatalog.

1. Zunächst wird ein Adapter-Paket erstellt:

```
$ mkdir vs.theme/vs/theme/adapters
$ touch vs.theme/vs/theme/adapters/__init__.py
```

2. Anschließend wird es in die Konfiguration eingeschlossen in `vs.theme/vs/theme/configure.zcml`:

```
<include package=".adapters" />
```

3. Ein einzelner Adapter wird dann konfiguriert in `vs.theme/vs/theme/adapters/configure.zcml`:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:five="http://namespaces.zope.org/five"
    xmlns:cmf="http://namespaces.zope.org/cmf"
    xmlns:i18n="http://namespaces.zope.org/i18n"
    i18n_domain="vs.theme">
  <adapter name="hasTeaserImage" factory=".indexer.hasTeaserImageDocument" />
</configure>
```

4. Die Klasse `hasTeaserImageDocument` in `vs.theme/vs/theme/adapters/indexer.py` sieht dann so aus:

```
from plone.indexer import indexer
from Products.ATContentTypes.interfaces import ITextContent

def _hasTeaserImage(obj, fieldname):
    """ generic wrapper """
    field = obj.getField(fieldname)
    if field is None:
        return False
    img = field.get(obj)
    img_data = str(img.data)
    return len(img_data) > 0

@indexer(ITextContent)
    def hasTeaserImageDocument(obj):
    """ Returns True/False if an teaser image exists or not """
    return _hasTeaserImage(obj, 'teaserImage')
```

z3c.jbot

Mit z3c.jbot lassen sich Views und alle Objekte des *Skins Tool*, wie Page Templates, CSS- und Javascript-Dateie einfach überschreiben oder erweitern.

Überschreiben eines bestehenden Views

#.1 Zunächst tragen wir in unser Theme-Produkt die Abhängigkeit von `z3c.jbot` in `vs.theme/setup.py` ein:

```
...
install_requires=[
    'setuptools',
    'z3c.jbot',
    ...
],
...
```

1. Anschließend wird folgende Direktive in `vs.theme/vs/theme/configure.zcml` angegeben:

```
<configure
    ...
    xmlns:browser="http://namespaces.zope.org/browser">
...
<include package="z3c.jbot" file="meta.zcml" />
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
...
<browser:jbot
 directory="overrides"
 layer=".interfaces.IThemeSpecific"
 />
```

2. Der Layer wird nun konfiguriert in `vs.theme/vs/theme/profiles/default/browserlayer.xml`:

```
<?xml version="1.0"?>
<layers>
    <layer name="vs.theme"
           interface="vs.theme.interfaces.IThemeSpecific" />
</layers>
```

3. Der Layer erwartet das Interface `IThemeSpecific` in der Datei `vs.theme/vs/theme/interfaces.py`:

```
from plone.theme.interfaces import IDefaultPloneLayer
from plone.portlets.interfaces import IPortletManager

class IThemeSpecific(IDefaultPloneLayer):
    """Marker interface that defines a Zope 3 browser layer.
    """
```

4. Schließlich wird der Ordner `overrides` angelegt und darn z.B. eine Kopie von `atct_topic_view.pt` als `vs.theme/vs/theme/overrides/Products.ATContentTypes.skins.ATContentTypes.atct_topic_view.pt`.

   Dieses PageTemplate lässt sich dann z.B. um ein Teaser-Element erweitern:

```
<td>
    <img tal:condition="obj/hasTeaserImage"
         tal:attributes="src string:${obj/getURL}/@@teaserImage?scale=teaser"
         class="teaserImage"
         />
    <div class="teaserText"
         tal:condition="obj/teaserText"
         tal:content="structure obj/teaserText" />
</td>
```

Erweitern bestehender Templates

Mit z3c.jbot lassen sich auch einfach neue Templates aus bestehenden erstellen.

1. Hierzu wird nun in `vs.theme/vs/theme/overrides/Products.ATContentTypes.skins.ATContentTypes.atct_topic_view.pt` eine Bedingung für die Tabellenzeile mit dem Teaser eingefügt:

```
<table class="listing"
       summary="Content listing"
       i18n:attributes="summary summary_content_listing;">
    <thead>
        <tr tal:condition="options/with_teaser  | request/with_teaser | nothing" >
        ...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        <td tal:condition="options/with_teaser | request/with_teaser | nothing">
            <tal:if condition="obj/hasTeaserImage">
                <a tal:attributes="href obj/getURL">
                    <img tal:condition="obj/hasTeaserImage"
                         tal:attributes="src string:${obj/getURL}/@@teaserImage?
↪scale=teaser"
                         class="teaserImage"
                    />
                </a>
            </tal:if>
            <tal:if condition="not: obj/hasTeaserImage">
                <div class="title" tal:content="obj/Title" />
                <div class="description" tal:content="obj/Description" />
                <a tal:attributes="href obj/getURL"
                   i18n:translate="label_more">
                    more
                </a>
            </tal:if>
        </td>
```

2. Nun wird `with_teaser` definiert in `vs.theme/vs/theme/browser/topic.py`:

```
from Products.Five.browser import BrowserView

class TopicTeaserView(BrowserView):
    """ Topic table view with teaser """

    def __call__(self):
        view = self.context.restrictedTraverse('atct_topic_view')
        return view(with_teaser=True)
```

3. `hasTeaserImage` wird aus dem Index abgefragt. Sehen Sie hierzu plone.indexer.

4. Anschließend wird in `vs.theme/vs/theme/browser/configure.zcml` die neue Ansicht registriert:

```
<browser:page
  name="atct_topic_teaser_view"
  for="Products.ATContentTypes.interfaces.topic.IATTopic"
  permission="zope2.View"
  class=".topic.TopicTeaserView"
  />
```

5. Schließlich wird dieser View in derselben `zcml`-Datei noch für das *Hinzufügen*-Menü konfiguriert:

```
<include package="plone.app.contentmenu" />
...
<browser:menuItem
  for="Products.ATContentTypes.interfaces.topic.IATTopic"
    menu="plone_displayviews"
    title="Collection with teaser"
    action="atct_topic_teaser_view"
    description="Collection table view with teaser"
    />
```

Skalierung von Bildern

Dieie Angabe der Bildgröße erfolgt hierbei nicht in der Schemadefinition sondern im Page Template:

```
<img tal:define="scale context/@@images"
 tal:replace="structure python: scale.scale('image',
              width=260, height=160, direction='down').tag()" />
```

Die Berechnung der Bildgröße erfolgt dann beim Aufrufen der Seite.

Folgende Parameter sind verfügbar:

up skaliert die kürzere Seite auf die gewünschte Größe und beschneidet die andere Seite falls nötig.

down skaliert die längere Seite auf die gewünschte Größe und beschneidet die andere Seite falls nötig.

thumbnail skaliert das Bild auf die gewünschte Größe ohne es zu beschneiden.

Die Proportionen des Bildes können sich hierdurch verändern.

Diese Option erfordert die Angabe sowohl der Höhe als auch der Breite.

quality Qualität des resultierenden Bildes.

Deco Grid System

Spaltenbreite

Die Breite der Klassen lässt sich dann einfach berechnen mit6.25*n - 2.25)%. Dies führt dann z.B. zu folgenden Klassen:

width-full Die volle Breite beträgt 97.75%, da links und rechts jeweils eine Margin von 1,125% erhalten bleiben.

width-1:2 Die Breite beträgt 47.75%.

width-3:4 Die Breite beträgt 72.75%.

Positionierung

Die Positionierung der Klassen wird vom rechten Rand aus vorgenommen mit `margin-left: -100 + (6.25*n) + 1.125`. Dies führt z.B. zu folgenden Klassen:

position-0 `{margin-left: -73.875%;}`

position-1:4 `{margin-left: -73.875%;}`

position-1:2 `{margin-left: -48.875%;}`

position-3:4 `{margin-left: -23.875%;}`

Verschieben von Spalten

Sollen nun beispielsweise die beiden Portlet-Spalten links vom Inhalt positioniert werden, so sind hierfür nur geringe Änderungen notwendig:

Zunächst wird der bestehende sunburstview überschrieben in `vs.theme/vs/theme/browser/configure.zcml`:

```
<browser:page
    for="*"
    name="sunburstview"
    class=".sunburstview.SunburstView"
    layer=".interfaces.IThemeSpecific"
    permission="zope.Public"
    allowed_interface="plonetheme.sunburst.browser.interfaces.ISunburstView"
    />
```

Anschließend kopieren wir sunburstview.py in unser browser-Package und ändern darin die Berechnung der css-Klasse für denjenigen div-Tag, der den Inhaltsbereich enthält:

Sofern beide Portlet-Spalten angezeigt werden, soll der Inhaltsbereich 2 von 4 Spalten breit sein und ab der Hälfte der Seite beginnen, also:

```
elif sl and sr:
    return "cell width-1:2 position-1:2"
```

Sofern nur die rechte Portlet-Spalte, nicht jedoch die linke angezeigt werden soll, bleibt die rechte Spalte an ihrer Position stehen und auch in diesem Fall wird der Inhalt 2 Spalten breit sein und ab der Hälfte der Seite beginnen:

```
elif (sr and not sl) and (not portal_state.is_rtl()):
    return "cell width-1:2 position-1:2"
```

Schließlich müssen wir noch die main_template.pt-Datei anpassen um die rechte Portlet-Spalte immer an zweiter Stelle im Raster-layout anzuzeigen. Hierfür wird für den div-Tag mit der ID portal-column-two die position-Klasse geändert in:

```
<div id="portal-column-two"
    class="cell width-1:4 position-1:4"
    ...
    tal:attributes="class python:isRTL and 'cell width-1:4 position-0' or 'cell width-
↪1:4 position-1:4'">
```

3.15.2 Dummy-Inhalte

Das Erstellen von Dummy-Inhalten, seien es nun Seiten, Bilder oder Dateien, ist sehr nützlich um mit geringem Aufwand einen besseren Eindruck vom *Look & Feel* der neuen Website zu erhalten. Hierzu haben wir die `setup.py`-Datei erweitert, sodass Texte mit dem loremipsum-Modul generiert werden. Das loremipsum-Modul kann so konfiguriert werden, dass es einzelne Sätze oder mehrere Absätze generieren kann:

```
import loremipsum

def gen_paragraphs(num=3):
    return u'/'.join([p[2] for p in loremipsum.Generator().generate_paragraphs(num)])

def gen_sentence():
```

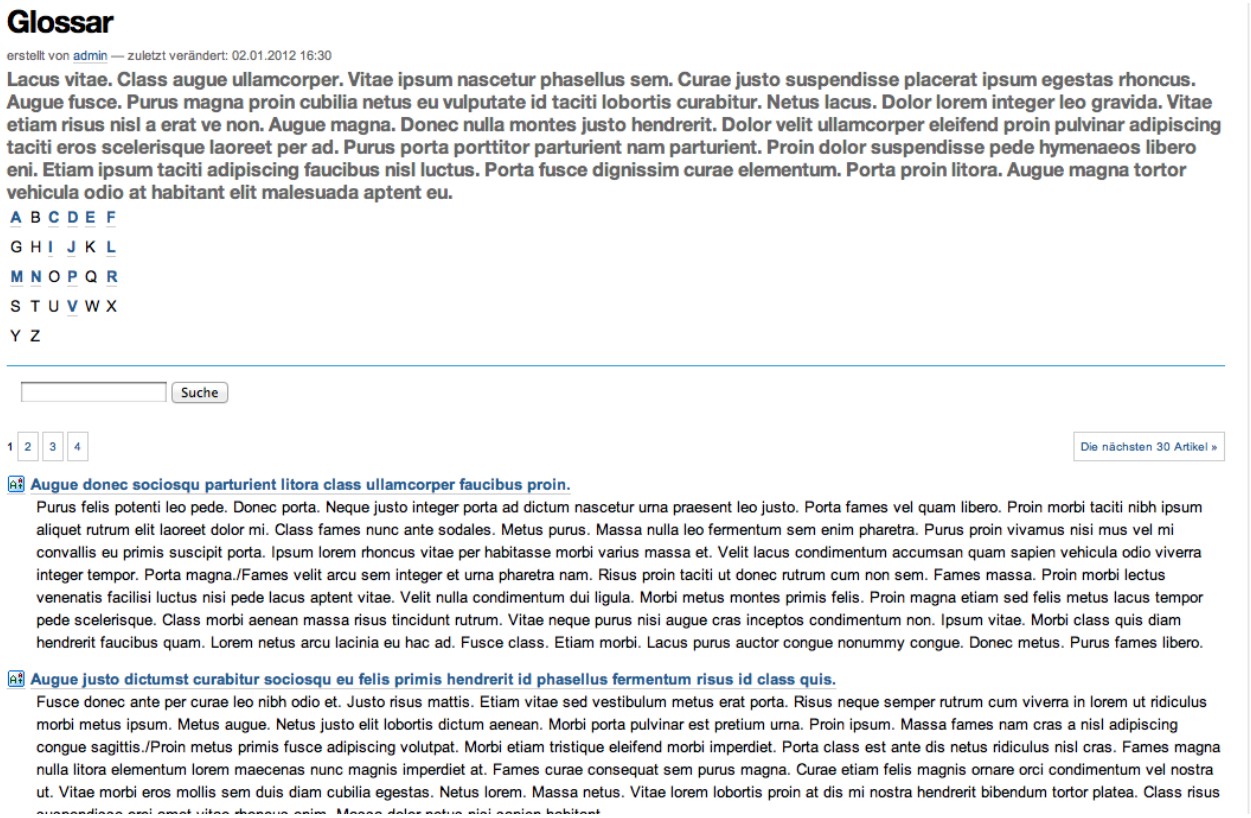
(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    return loremipsum.Generator().generate_sentence()[-1]

def gen_sentences(length=80):
    return u'/'.join([s[2] for s in loremipsum.Generator().generate_
↪sentences(length)])
```

Das folgende Glossar kann dann einfach so erzeugt werden:

```
def installGlossary(self, site):
    service = site.restrictedTraverse('deutschland/de/service')
    glossary = invokeFactory(service, 'PloneGlossary', 'Glossar')
    for i in range(100):
        term = invokeFactory(glossary, 'PloneGlossaryDefinition')
        term.setDefinition(gen_paragraphs(2))
        term.reindexObject()
```



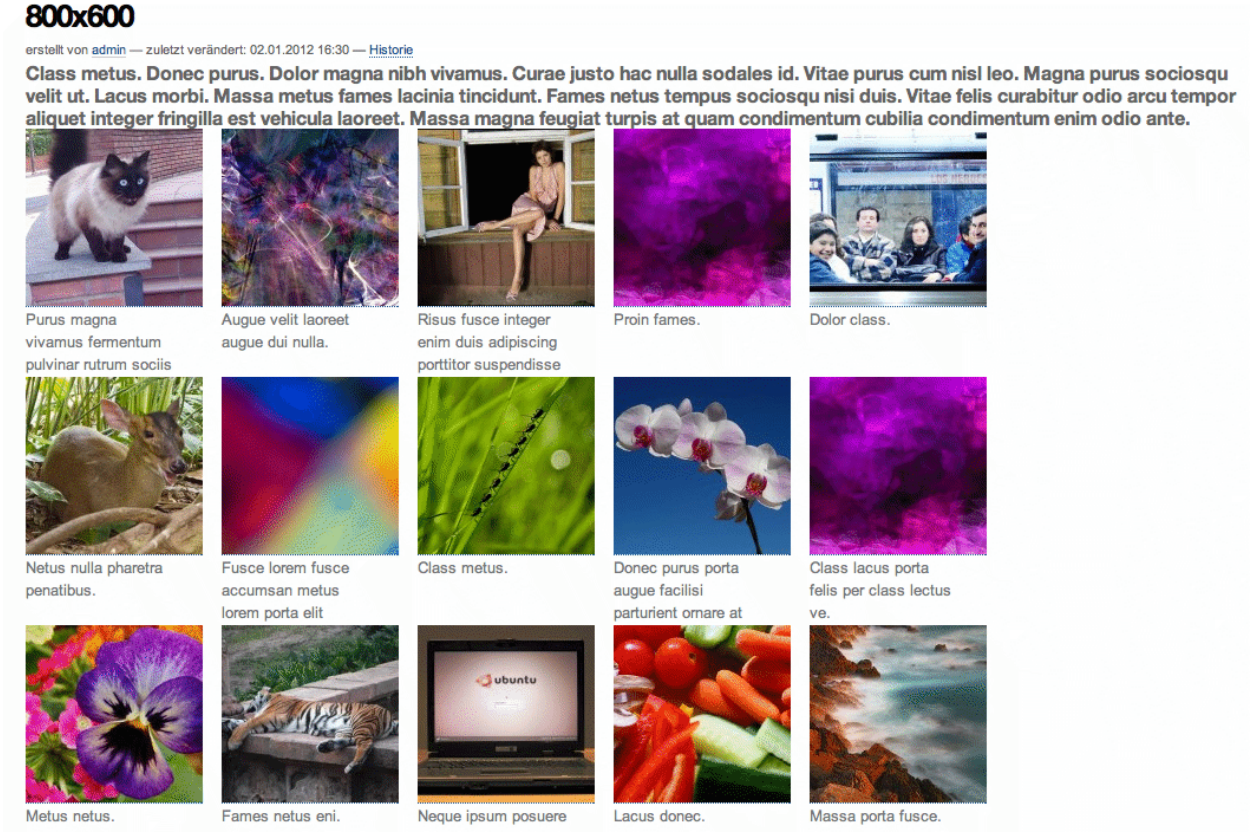
Auch Bilder lassen sich erstellen, wobei die lorempixel-Website verwendet wird. Der Quellcode zur unten abgebildeten Gallerie sieht z.B. so aus:

```
import random
import urllib2

def random_image(width, height):
    url = 'http://lorempixel.com/%d/%d/' % (width, height)
    return urllib2.urlopen(url).read()

def installAssets(self, site):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    service = site.restrictedTraverse('de/service')
    assets = invokeFactory(service, 'Folder', 'Assets')
    for width,height in ((200,200), (400,400), (600, 400), (800, 600), (800,800),␣
↪(1024, 768)):
        imagefolder_id = '%sx%s' % (width, height)
        images = invokeFactory(assets, 'Folder', imagefolder_id)
        for i in range(20):
            img = invokeFactory(images, 'Image')
            img.setImage(random_image(width, height))
            img.reindexObject()
```



Update

Mit zopyx.ipsumplone hat unser Partner Andreas Jung nun ein eigenständiges Produkt zum Erstellen von Dummy-Inhalten entwickelt. Nach der Installation dieses Produkts muss innerhalb einer Plone-Site nur der View `@@demo-content` aufgerufen werden um eine Reihe von Ordnern mit Bilder, Nachrichten, Terminen und Dateien zu erzeugen.

3.15.3 Referenz

zc.buildout

Buildout erlaubt, identische Entwicklungsumgebungen einfach aufzusetzen. Hierzu nutzt Buildout die Fähigkeit der setuptools, automatisch Abhängigkeiten aufzulösen und Aktualisierungen durchzuführen.

Variablensubstitution

Buildout-Konfigurationsdateien erlauben verschiedene Ersetzungen von Variablen, z.B.:

```
[buildout]
parts =
    variables
    source

[variables]
var = ${source:path}/var
logs = ${variables:var}/logs

[source]
path = instance
```

Die Namen von Abschnitten und Optionen in Variablenersetzungen dürfen nur alphanumerische Zeichen, Bindestriche, Punkte und Leerzeichen enthalten.

Der Name des Abschnitts kann im selben Abschnitt weggelassen werden, soll jedoch der aktuelle Name des Abschnitts ermittelt werden, so ist dies mit `_buildout_section_name_` möglich, also z.B.:

```
[variables]
...
base_variables = ${:_buildout_section_name_}
```

Abschnitte erweitern – Makro

Ein Abschnitt kann ein oder mehrere Abschnitte erweitern wobei die Optionen des referenzierten Abschnitts zunächst kopiert und anschließend die Variablen substituiert werden. Dies ermöglicht die Verwendung von Abschnitten als Makros.

Beispiel:

```
[instance-base]
recipe = plone.recipe.zope2instance
...

[instance1]
<=instance-base
http-address = 8081

[instance2]
<=instance
http-address = 8082
```

Optionen hinzufügen und entfernen

Attributen lassen sich Werte hinzufügen und entfernen mit den Operatoren + und −. Folgendes Beispiel kann dies illustrieren:

```
[instance-debug]
<=instance-base
eggs +=
    Products.PDBDebugMode
    z3c.deadlockdebugger
```

oder umgekehrt:

```
[instance1]
<=instance-base
eggs -=
    Products.PDBDebugMode
```

Mehrere Konfigurationsdateien

Eine Buildout-Konfigurationsdatei kann eine andere *erweitern*. Dabei werden die Optionen der erweiterten Konfigurationsdatei gelesen sofern sie nicht bereits definiert sind:

```
[buildout]
extends = base.cfg
```

Weitere Informationen

- Detailed Buildout Documentation

mr.bob

mr.bob ist ein Dateisystem-Template-Renderer.

Einleitung

mr.bob ermöglicht, aus einer Vorlage eine Verzeichnisstruktur zu erstellen, die das Erstellen von Python-Paketen deutlich vereinfacht.

Im `bobtemplate`-Namespace sind u.a. folgende Pakete zu finden:

bobtemplates.plone erstellt Python-Pakete für Plone, ggf. auch mit *nested Namespaces*.

bobtemplates.gillux liefer Vorlagen zum Erstellen von Buildout-Projekten, zum Erstellen eigener bobtemplates und Python-Namespace-Paketen, optional mit nose, coverage und Sphinx-Dokumentationsvorlagen.

bobtemplates.ielectric Pyramid und Python- Basis-Paket.

bobtemplates.niteoweb Plone- und Pyramid-Vorlagen

Installation

`mr.bob``und ``bobtemplates`-Pakete lassen sich einfach mit Buildout installieren, z.B.:

```
[buildout]
parts =
    ...
    mrbob

[mrbob]
recipe = zc.recipe.egg
eggs =
    mr.bob
    bobtemplates.plone
```

Konfiguration

Mit dem folgenden Aufruf können Antworten für zukünftige Pakete gespeichert werden:

```
$ mrbob --remember-answers -O vs.policy bobtemplates:plone_addon
...
```

Anschließend kann diese Konfiguration immer wieder verwendet werden, z.B. mit:

```
$ ../bin/mrbob --config .mrbob.ini -O vs.theme bobtemplates:plone_addon
```

Eine solche Konfigurationsdatei kann auch mit einer URL angespreochen werden, also z.B.:

```
$ ../bin/mrbob --config https://raw.github.com/veit/dotfiles/master/.mrbob.ini␣
↪bobtemplates:plone_addon
```

Alternativ kann auch eine globale Konfigurationsdatei erstellt werden in `~/.mrbob.ini`, z.B.:

```
[mr.bob]
verbose = True

[variables]
package.namespace = vs
author.name = Veit Schiele
author.email = kontakt@veit-schiele.de
author.github.user = veit
author.irc = irc.freenode.org#veit
```

Siehe auch:

- mr.bob's documentation
- Git repository

bobtemplates.plone

bobtemplates.plone liefert Vorlagen für mr.bob template um Pakete für Plone-Projekte zu erstellen.

Erstellen eines Pakets

Um ein Paket zu erstellen wie `vs.policy`, geben Sie folgendes im Terminal ein:

```
$ cd src/
$ ../bin/mrbob -O vs.policy bobtemplates:plone_addon
```

Es können auch Pakete mit verschachtelten Namensräumen erstellt werden, z.B.:

```
$ ../bin/mrbob -O vs.bootstrap.tinymce bobtemplates:plone_addon
```

Anschließend müssen Sie die folgenden Optionen beantworten:

Package Type? Optionen sind

- `Basic`
- `Dexterity`
- `Theme`

Der Standardwert ist `Basic`.

Author's name Hier sollte Ihr Name angegeben werden

Author's email Ihre E-Mail-Adresse

Author's github username Ihr Account-Name bei github

Package description Einzeilige Beschreibung des Pakets.

Der Standardwert ist `An add-on for Plone`

Plone version [4.3.6] Für welche Plone-Version wird das Paket entwickelt?

Dateistruktur

Basic

Die `Basic`-Vorlage liefert die folgende Dateistruktur:

```
vs.policy/
├── bootstrap-buildout.py
├── buildout.cfg
├── CHANGES.rst
├── CONTRIBUTORS.rst
├── docs
│   ├── index.rst
│   ├── LICENSE.GPL
│   └── LICENSE.rst
├── MANIFEST.in
├── README.rst
├── setup.py
├── src
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    └── vs
        ├── __init__.py
        └── policy
            ├── browser
            │   ├── configure.zcml
            │   ├── __init__.py
            │   ├── overrides
            │   └── static
            ├── configure.zcml
            ├── __init__.py
            ├── interfaces.py
            ├── locales
            │   ├── update.sh
            │   └── vs.policy.pot
            ├── profiles
            │   ├── default
            │   │   ├── browserlayer.xml
            │   │   ├── metadata.xml
            │   │   └── vspolicy_default.txt
            │   └── uninstall
            │       ├── browserlayer.xml
            │       └── vspolicy_uninstall.txt
            ├── setuphandlers.py
            ├── testing.py
            └── tests
                ├── __init__.py
                ├── robot
                │   └── test_example.robot
                ├── test_robot.py
                └── test_setup.py
└── travis.cfg
```

buildout.cfg Das Paket enthält eine Buildout-Konfigurationsdatei, die z.B. zum Testen verwendet werden kann.

src/vs/policy/tests/ Das Paket kommt mit einem Test-Setup und Beispieltests zur Installation des Pakets.

Es enthält außerdem in `src/vs/policy/tests/robot/test_example.robot` einen Robot-Test für das Anmelden an der Plone-Site.

Schließlich enthält das Paket mit `travis.cfg` auch eine Konfigurationsdatei, das das Testen des Pakets mit Travis erlaubt.

src/vs/policy/profile Das Paket enthält ein Generic Setup-Profil, das einen Browserlayer installiert.

Für Plone 5 wird daneben noch ein `uninstall`-Profil installiert.

src/vs/policy/Locales Das Paket registriert ein Verzeichnis für die Übersetzungsdateien.

src/vs/policy/browser/overrides Das Paket registriert einen Ordner, in dem Templates etc. mit z3c.jbot überschrieben werden können.

src/vs/policy/setuphandlers.py Diese Datei kann verwendet werden um Code hinzuzufügen, der beim Installieren eines Pakets ausgeführt werden soll.

Für Plone 5 gibt es innerhalb dieser Datei auch eine Methode, die beim Deinstallieren aufgerufen wird.

---

setup.py In `install_requires` werden zusätzlich die folgenden zwei Pakete angegeben:

- `plone.app.theming`

- `plone.app.themingplugins`

`src/vs/theme/configure.zcml` Hier wird der Ordner mit dem Theme konfiguriert:

```
<configure
    ...
    xmlns:plone="http://namespaces.plone.org/plone"
    ...

<plone:static
    directory="theme"
    type="theme"
    name="vs.theme"
    />
```

`src/vs/theme/profiles/default/metadata.xml` Hier wird als Abhängigkeit `plone.app.theming` angegeben:

```
<dependency>profile-plone.app.theming:default</dependency>
```

Dexterity

`setup.py` In `install_requires` wird zusätzlich `plone.app.dexterit` angegeben

`src/vs/task/interfaces.py` Für den Dexterity-Artikeltyp wird ein Interface angegeben, in unserem Fall:

```
from vs.task import _
from zope import schema
from zope.interface import Interface

class ITask(Interface):

    title = schema.TextLine(
        title=_(u"Title"),
        required=True,
    )

    description = schema.Text(
        title=_(u"Description"),
        required=False,
    )
```

`src/vs/task/profiles/default/metadata.xml` Hier wird als Abhängigkeit `plone.app.dexterity` angegeben:

```
<dependency>profile-plone.app.dexterity:default</dependency>
```

bobtemplates.plone

bobtemplates.plone liefert Vorlagen für mr.bob template um Pakete für Plone-Projekte zu erstellen.

Erstellen eines Pakets

Um ein Paket zu erstellen wie `vs.policy`, geben Sie folgendes im Terminal ein:

```
$ cd src/
$ ../bin/mrbob -O vs.policy bobtemplates:plone_addon
```

Es können auch Pakete mit verschachtelten Namensräumen erstellt werden, z.B.:

```
$ ../bin/mrbob -O vs.bootstrap.tinymce bobtemplates:plone_addon
```

Anschließend müssen Sie die folgenden Optionen beantworten:

Package Type? Optionen sind

- `Basic`
- `Dexterity`
- `Theme`

Der Standardwert ist `Basic`.

Author's name Hier sollte Ihr Name angegeben werden

Author's email Ihre E-Mail-Adresse

Author's github username Ihr Account-Name bei github

Package description Einzeilige Beschreibung des Pakets.

Der Standardwert ist `An add-on for Plone`

Plone version [4.3.6] Für welche Plone-Version wird das Paket entwickelt?

Dateistruktur

Basic

Die `Basic`-Vorlage liefert die folgende Dateistruktur:

```
vs.policy/
├── bootstrap-buildout.py
├── buildout.cfg
├── CHANGES.rst
├── CONTRIBUTORS.rst
├── docs
│   ├── index.rst
│   ├── LICENSE.GPL
│   └── LICENSE.rst
├── MANIFEST.in
├── README.rst
├── setup.py
├── src
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    └── vs
        ├── __init__.py
        └── policy
            ├── browser
            │   ├── configure.zcml
            │   ├── __init__.py
            │   ├── overrides
            │   └── static
            ├── configure.zcml
            ├── __init__.py
            ├── interfaces.py
            ├── locales
            │   ├── update.sh
            │   └── vs.policy.pot
            ├── profiles
            │   ├── default
            │   │   ├── browserlayer.xml
            │   │   ├── metadata.xml
            │   │   └── vspolicy_default.txt
            │   └── uninstall
            │       ├── browserlayer.xml
            │       └── vspolicy_uninstall.txt
            ├── setuphandlers.py
            ├── testing.py
            └── tests
                ├── __init__.py
                ├── robot
                │   └── test_example.robot
                ├── test_robot.py
                └── test_setup.py
└── travis.cfg
```

buildout.cfg Das Paket enthält eine Buildout-Konfigurationsdatei, die z.B. zum Testen verwendet werden kann.

src/vs/policy/tests/ Das Paket kommt mit einem Test-Setup und Beispieltests zur Installation des Pakets.

Es enthält außerdem in `src/vs/policy/tests/robot/test_example.robot` einen Robot-Test für das Anmelden an der Plone-Site.

Schließlich enthält das Paket mit `travis.cfg` auch eine Konfigurationsdatei, das das Testen des Pakets mit Travis erlaubt.

src/vs/policy/profile Das Paket enthält ein Generic Setup-Profil, das einen Browserlayer installiert.

Für Plone 5 wird daneben noch ein `uninstall`-Profil installiert.

src/vs/policy/Locales Das Paket registriert ein Verzeichnis für die Übersetzungsdateien.

src/vs/policy/browser/overrides Das Paket registriert einen Ordner, in dem Templates etc. mit z3c.jbot überschrieben werden können.

src/vs/policy/setuphandlers.py Diese Datei kann verwendet werden um Code hinzuzufügen, der beim Installieren eines Pakets ausgeführt werden soll.

Für Plone 5 gibt es innerhalb dieser Datei auch eine Methode, die beim Deinstallieren aufgerufen wird.

---

setup.py In `install_requires` werden zusätzlich die folgenden zwei Pakete angegeben:

- `plone.app.theming`

- `plone.app.themingplugins`

`src/vs/theme/configure.zcml` Hier wird der Ordner mit dem Theme konfiguriert:

```
<configure
    ...
    xmlns:plone="http://namespaces.plone.org/plone"
    ...

<plone:static
    directory="theme"
    type="theme"
    name="vs.theme"
    />
```

`src/vs/theme/profiles/default/metadata.xml` Hier wird als Abhängigkeit `plone.app.theming` angegeben:

```
<dependency>profile-plone.app.theming:default</dependency>
```

Dexterity

`setup.py` In `install_requires` wird zusätzlich `plone.app.dexterit` angegeben

`src/vs/task/interfaces.py` Für den Dexterity-Artikeltyp wird ein Interface angegeben, in unserem Fall:

```
from vs.task import _
from zope import schema
from zope.interface import Interface

class ITask(Interface):

    title = schema.TextLine(
        title=_(u"Title"),
        required=True,
    )

    description = schema.Text(
        title=_(u"Description"),
        required=False,
    )
```

`src/vs/task/profiles/default/metadata.xml` Hier wird als Abhängigkeit `plone.app.dexterity` angegeben:

```
<dependency>profile-plone.app.dexterity:default</dependency>
```

ZopeSkel

ZopeSkel – Einleitung und Installation

ZopeSkel ist eine Sammlung von Vorlagen, mit denen sich schnell Buildout-Projekte und Plone-Erweiterungen erstellen lassen.

Einleitung

Bemerkung: In der Vergangenheit war ZopeSkel ein einziges großes Paket, das nun ab Version 3.0 in mehrere kleine Pakete aufgeteilt wurde, die unter dem `templer`-Namespace erschienen.

Falls Sie ältere Vorlagen benötigen, sollten Sie eine Version von `ZopeSkel < 3.0` verwenden.

Im `templer`-Namespace sind u.a. folgende Pakete zu finden:

templer.core stellt Ihnen `basic_namespace` und `nested_namespace` zum Erstellen von Python-Namespace- und verschachtelten Python-Namespace-Paketen zur Verfügung.

templer.buildout stellt Ihnen `basic_buildout` und `recipe` zum Erstellen von Buildout-Projekten und Rezepten zum Erweitern des Buildout-Systems zur Verfügung.

templer.zope stellt Ihnen `zope2_basic` und `zope2_nested` zum Erstellen von Zope-Namespace- und verschachtelten Zope-Namespace-Paketen zur Verfügung.

templer.plone stellt Ihnen `archetype`, `plone_basic` und `plone_nested` zum Erstellen von Paketen für Plone.

templer.plone.localcommands stellt Ihnen sog. *local commands* zur Verfügung und zwar für die folgenden Vorlagen:

- `archetype`

  `contenttype` Ein Gerüst für einen Archetypes-Artikeltyp

  `schema_field` Ein iterativer Generator für Archetypes-Felder.

- `plone_basic`

  `browserview` Eine Zope-`BrowserView`-Klasse mit Interface und Template

  `browserlayer` Ein Zope-`BrowserLayer`-Interface und dessen GenericSetup-Registrierung

Installation

ZopeSkel kann einfach mit buildout installiert werden:

```
parts =
    ...
    zopeskel

[zopeskel]
recipe = zc.recipe.egg
unzip = true
eggs =
    Paste
    ZopeSkel
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    templer.plone
    templer.plone.localcommands
```

Beim Erstellen des buildout-Projekts werden im `bin/`-Verzeichnis die Skripte `zopeskel` und `paster` erstellt.

`bin/zopeskel --list` gibt eine Liste der verfügbaren Vorlagen aus.

`bin/zopeskel --help` stellt Ihnen eine vollständige Hilfe für ZopeSkel zur Verfügung.

Weitere Informationen

- Templer System Manual

ZopeSkel – Verfügbare Vorlagen und Variablen

Um eine Liste mit allen verfügbaren Vorlagen (Templates) und ausführlichen Beschreibungen zu erhalten, geben sie folgendes ein:

```
$ ./bin/zopeskel --list

Plone Development
-----------------

archetype: A Plone project that uses Archetypes content types

   This creates a Plone project that uses Archetypes content types. It
   has local commands that will allow you to add content types and to
   add fields to your new content types.
...
```

Um nun ein Projekt aus einer dieser Vorlagen zu erstellen, wird ZopeSkel folgendermaßen aufgerufen:

```
$ ./bin/zopeskel <template> <output-name>
```

also z.B.:

```
$ cd src/
$ ../bin/zopeskel archetype vs.registration
```

Es können auch noch weitere Variablen neben dem Projektnamen mitgegeben werden, z.B.:

```
$ ../bin/zopeskel archetype vs.registration author_email=kontakt@veit-schiele.de
```

Dies ist gut geeignet sofern Pakete skriptgesteuert erstellt werden sollen. Eine vollständige Liste der Variablen erhalten Sie mit:

```
$ ./bin/paster create -t <template-name> --list-variables
```

ZopeSkel – Standardeinstellungen

Hierzu können Sie die Datei `.zopeskel` in Ihrem Home-Verzeichnis anlegen, z.B. mit:

```
$ ./bin/zopeskel --make-config-file > ~/.zopeskel
```

Die `.zopeskel`-Datei kann dann z.B. so aussehen:

```
[DEFAULT]
author_email = kontakt@veit-schiele.de
license_name = BSD
master_keywords = Web Python Zope

[[plone_basic]]
expert_mode = all
namespace_package = vs
add_profile = True
keywords = %(master_keywords)s Plone
url = https://github.com/veit/
```

- Sie können im `[DEFAULT]`-Abschnitt bestimmte Werte angeben, die für eine spezifische Vorlage wieder überschrieben werden können. So ist z.B. im `[DEFAULT]`-Abschnitt als Lizenz `BSD` angegeben, diese wird jedoch im `[plone3_theme]`-Abschnitt für diese Vorlage wieder überschrieben.
- Sie können auch Angaben aus dem `[DEFAULT]`-Abschnitt in spezifischen Vorlagen erweitern unter Verweis auf die `master`-Liste.

Bemerkung: Im Gegensatz zum `paster create`-Aufruf kann das `zopeskel`-Skript nicht mit dem Argument `--svn-repository` aufgerufen werden um ein Paket an einer bestimmten Stelle eines SVN-Repository zusammen mit der Verzeichnishierarchie `trunk/`, `tags/` und `branches/` zu erstellen.

ZopeSkel – Local Commands

Neben den Vorlagen für Projekte liefert ZopeSkel auch Vorlagen, die nur in bestimmten Kontexten zur Verfügung stehen, sog. local commands. Mit diesen können Sie bestehende ZopeSkel-Projekte erweitern.

Bemerkung: *Local commands* können aktuell nur mit dem `paster`-Skript aufgerufen werden.

Installation

Die Installation erfolgt in Buildout mit:

```
[buildout]
parts =
    ...
    paster
    zopeskel
...
[paster]
recipe = zc.recipe.egg
eggs =
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
   ZopeSkel
   PasteScript
   PasteDeploy
```

Verwendung

Wenn Sie z.B. ein Archetypes-Paket erstellt haben mit:

```
$ cd src
$ ../bin/zopeskel archetype vs.registration
```

dann können Sie als nächstes die *local commands* hierfür installieren mit:

```
$ cd vs.registration/
$ python setup.py egg_info
```

Schließlich können Sie in das `src/`-Verzechnis dieses Pakets wechseln und dort einen Artikeltyp erstellen:

```
$ cd src/
$ ../../../bin/paster add contenttype Registrant
```

Einen Überblick über alle im Kontext verfügbaren *local commands* erhalten Sie mit:

```
$ ../../../bin/paster add --list
Available templates:
    browserlayer:  A Plone browserlayer
    browserview:   A browser view skeleton
    contenttype:   A content type skeleton
    schema_field:  A handy AT schema builder
```

Siehe auch:

- ZopeSkel with local commands

ZopeSkel Templates erstellen

Erstellen eigener ZopeSkel- und local commands-Templates.

Erstellen eines *local command*-Templates

Die Python-Skripte zum Erstellen eines *local commands*-Templates entsprechen weitgehend denen eines normalen ZopeSkel-Templates. Wenn wir uns die `Portlet`-Klasse in `zopeskel/localcommands/archetype.py` genauer anschauen, stellen wir fest, dass sie von `ArchetypeSubTemplate` abgeleitet wird und diese wiederum von `ZopeSkelLocalTemplate`:

```
import os
from templer.core.vars import var
from templer.localcommands import TemplerLocalTemplate

from Cheetah.Template import Template as cheetah_template

class ArchetypeSubTemplate(TemplerLocalTemplate):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    use_cheetah = True
    parent_templates = ['archetype']

class ContentType(ArchetypeSubTemplate):
    _template_dir = 'templates/archetype/contenttype'
    summary = "A content type skeleton"
...
```

use_cheetah Für *local commands* muss der Wert auf `True` gesetzt werden.

parent_templates Liste der ZopeSkel-Templates, die dieses *local command*-Template aufrufen können.

_template_dir Verzeichnis mit den Template-Dateien.

summary Zusammenfassende Beschreibung des Templates.

Die anschließend folgende `pre`-Methode ermittelt die Variablen `contenttype_classname`, `contenttype_classname`, `contenttype_classname`, `contenttype_name` und `add_permission_name` des übergeordneten Pakets.

Auch die Template-Struktur entspricht weitgehend der von normalen ZopeSkel-Templates mit dem Unterschied, dass alle Dateien mit `_insert` enden. Betrachten wir uns z.B. das `portlet`-Template genauer, entdecken wir folgende Struktur:

```
$ tree ~/.buildout/eggs/templer.plone.localcommands-1.0b1-py2.7.egg/templer/plone/
↪localcommands/templates/archetype/contenttype/
/home/veit/.buildout/eggs/templer.plone.localcommands-1.0b1-py2.7.egg/templer/plone/
↪localcommands/templates/archetype/contenttype/
├── config.py_insert
├── content
│   ├── configure.zcml_insert
│   └── +content_class_filename+.py_tmpl
├── interfaces
│   ├── +content_class_filename+.py_tmpl
│   └── __init__.py_insert
├── profiles
│   └── default
│       ├── factorytool.xml_insert
│       ├── rolemap.xml_insert
│       ├── types
│       │   └── +types_xml_filename+.xml_tmpl
│       └── types.xml_insert
└── README.txt_insert

5 directories, 10 files
```

Die mit `_tmpl` endenden Dateien werden wie normale ZopeSkel-Templates behandelt. Speziell für **local command**-Templates sind die auf `_insert` endenden Dateien. Der Inhalt dieser Dateien wird in die korrespondierenden Dateien des bereits bestehenden Projekts eingefügt. Schauen wir uns nun z.B. `profiles/default/rolemap.xml_insert` genauer an:

```
#<?xml version="1.0"?>
#<rolemap>
#  <permissions>
#    <!-- -*- extra stuff goes here -*- -->
    <permission name="$add_permission_name" acquire="False">
      <role name="Manager" />
      <role name="Contributor" />
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    </permission>
#  </permissions>
#</rolemap>
```

- Existiert in dem Projekt bereits eine Datei `profiles/default/rolemap.xml`, dann werden nur die Zeilen hinzugefügt, die nicht mit # beginnen.
- Existiert noch keine `profiles/default/rolemap.xml`-Datei, wird die Datei aus dem Template ohne die mit # beginnenden Zeilen geschrieben.

Zum Weiterlesen

- Creating your own Paster templates

Testen

Verschiedene Arten von Tests

Hier eine Übersicht über verschiedene Arten von Tests, deren konkrete Unterscheidung jedoch schwierig werden kann.

Unit tests werden aus der Programmierperspektive geschrieben. Sie testen isoliert eine einzelne Methode oder Funktion.

Integrationstests untersuchen die Abhängigkeit von Methoden und Komponenten während Unit Tests meist mit möglichst wenig Abhängigkeiten geschrieben werden. Meist verwenden Unit Tests und Integrationstests jedoch dasselbe Framework.

Funktionale Tests beschreiben meist Nutzungsfälle (Use Cases) und deren Abläufe. Werden sie aus Nutzersicht geschrieben und beziehen sich nur auf die an der Oberfläche angebotenen Eingabemöglichkeiten, werden sie auch *Akzeptanztests* genannt.

Systemtests Auch Systemtests werden aus Nutzersicht geschrieben, jedoch ohne Kenntnis des Systems. Systemtests sollen Nutzer mit ihren üblichen Verhaltensmustern simulieren.

Tests und Dokumentation

Tests liefern häufig eine gute Beschreibung, wie einzelne Komponenten verwendet werden sollen, welche Schnittstellen und Zustände sie aufweisen können. Jim Fulton hat für den Zope-3-Entwicklungsprozess auf docstrings basierende Unit Tests eingeführt, die sich zunehmend auch in anderen Python-Projekten etablieren.

Siehe hierzu auch Tim Peters, Jim Fulton: Literate unit testing: Unit Testing with Doctest.

Konzepte

Test Case testet eine einzelnes Szenario.

Test Fixture ist eine konsistente Testumgebung.

Test Suite ist eine Sammlung mehrerer Test Cases.

Test Runner durchläuft eine Test Suite und stellt die Ergebnisse dar.

Einführung in Unit tests

Unit Tests sind kleine, sich selbst enthaltende Testmethoden, die unabhängig von anderen Methoden ausgeführt werden und sich nicht gegenseitig beeinflussen.

Folgende Regeln für unit tests sind zu beachten:

- Schreiben Sie mindestens einen Test für jede Methode.
- Schreiben Sie zunächst die Interface- und/oder stub-Methoden, dann die Tests. Vergewissern Sie sich, dass die Tests nicht bestanden werden (da der Code ja noch nicht geschrieben ist).
- Erst jetzt sollte das neue Feature implementiert werden mit dem Ziel, den Test zu bestehen.
- Wenn Sie nach einem Release einen Bug entdecken, beheben Sie ihn nicht einfach, sondern
  - schreiben Sie zunächst einen Test, der den Fehler demonstriert,
  - dann erst beseitigen Sie den Bug.

Unit tests erlauben Ihnen,

- Ihre Software auch in entfernten Umgebungen überprüfen zu können;
- beim Implementieren neuer Features nicht bereits bestehende zu kompromitieren;
- bereits behobene Bugs nicht wieder einzuführen;
- Zeit bei der Entwicklung einzusparen, da die Chancen fehlerhaften Code schnell zu erkennen, deutlich steigen;
- Code schreiben und testen in derselben Umgebung auszuführen;
- die Test-Abdeckung immer weiter zu erhöhen.

Unit Tests in Zope und Plone

Unit Tests im Zope-2/Plone-Kontext basieren meist auf *ZopeTestCase*, der das Python unittest-Modul verwendet. Dabei laufen die Unit Tests meistens in einer *Sandbox* (auch *Test-Fixture* genannt) ab.

PloneTestCase basiert auf *ZopeTestCase*, ist jedoch eher ein Integrationstest, der die Integration Ihrer und der zugrundeliegenden Komponenten wie ZODB und ZPublisher überprüft. *PloneTestCase* erstellt eine leere Zope-Instanz mit einer einzelnen Plone Site, einem Nutzer und dem Standard Mitglieder-Ordner. Ist der Test beendet, wird die Transaktion abgebrochen, so dass keine der durch den Test vorgenommenen Änderungen der Plone-Site erhalten bleibt.

Links

- Dive Into Python, Chapter on Unit Testing
- PyUnit Documentation
- How to run Zope Unit Tests
- ZopeTestCase ZopeTestCaseWiki
- Plone Testing Tutorial

Produkte

- ZopeTestCase
- CMFTestCase
- PloneTestCase

Test Runner

Es gibt verschiedene Möglichkeiten, Tests in Zope ablaufen zu lassen.

In den Versionen Zope 2.9 bis 2.11 lässt sich der TestRunner folgendermaßen aufrufen um ein ganzes Paket zu testen:

```
$ ./bin/test -s vs.policy
```

Filter

-s my.package, --package my.package, --dir my.package durchsucht die angegebenen Verzeichnisse nach Tests.

-m test_setup, --module test_setup spezifiziert ein Testmodul als regulären Ausdruck, z.B.:

```
$ ./bin/test -s my.package  -m 'test_setup'
```

-t '.*installed.*', --test test_theme_installed spezifiziert einen Testfilter als regulären Ausdruck, z.B.:

```
$ ./bin/test -s vs.policy -m '.*setup.*' -t '.*installed.*'
```

Hiermit werden im Paket `vs.policy` alle, mit `installed` endenden, Methoden in allen Testmodulen, die auf `setup` enden, durchlaufen.

-u, --unit durchläuft ausschließlich Unit tests und ignoriert andere `layer`- Optionen.

-f, --non-unit durchläuft alle Tests, die keine Unit Tests sind

Report

-v, --verbose führt zu ausführlicherer Ausgabe

--ndiff falls ein Doctest fehlschlägt, wird `ndiff.py` zur Darstellung der Unterschiede verwendet

--udiff falls ein Doctest fehlschlägt, wird Unified Diff zur Darstellung der Unterschiede verwendet

--cdiff falls ein Doctest fehlschlägt, wird Context Diff zur Darstellung der Unterschiede verwendet

Analyse

-d, post-mortem stoppt die Ausführung nach dem ersten nicht-bestandenen Test und ermöglicht *post-mortem*-Debugging, d.h. die Debug-Session wird nur gestartet, wenn ein Test fehlschlägt.

Setup

--path src/my.package fügt einen Pfad zu Pythons Suchpfad hinzu, wobei die Option mehrfach angegeben werden kann.

Weitere Optionen

Diese erhalten Sie mit:

```
$ ./bin/test --help
```

Wenn die relevanten Tests erfolgreich verliefen, sollten schließlich noch alle Tests durchgeführt werden um sicherzustellen, dass nicht an anderer Stelle etwas gebrochen ist. Wenn alle Tests erfolgreich durchlaufen wurden, erscheint eine Meldung:

```
Ran 10 tests with 0 failures and 0 errors in 4.830 seconds.
```

Falls nicht alle Tests erfolgreich durchlaufen wurden, ändert sich die Meldung:

```
Ran 10 tests with 2 failures and 3 errors in 9.688 seconds.
```

Dabei wurden dann zwei Tests nicht bestanden und drei Tests enthielten Fehler.

roadrunner

roadrunner ist ein Testrunner, der die testgetriebene Entwicklung deutlich beschleunigen kann, da er vorab das Standard-Zope- und Plone-Environment für PloneTestCase läd. zur Installation wird einfach folgendes in die `buildout.cfg`-Datei eingetragen:

```
[buildout]
parts =
    ...
    roadrunner

[roadrunner]
recipe = roadrunner:plone
packages-under-test = vs.policy
```

Anschließend kann es wie der reguläre Zope-Testrunner aufgerufen werden:

```
$ ./bin/roadrunner -s vs.policy
```

Testen mehrerer Eggs

Mit `zc.recipe.testrunner` steht ein Buildout-Rezept zum Erstellen eigener TestRunner für mehrere Eggs zur Verfügung. Die Buildout-Konfiguration für das Testen aller in einer Instanz verwendete Eggs kann z.B. so aussehen:

```
[buildout]
...
parts =
    ...
    instance
    test

[test]
recipe = zc.recipe.testrunner
defaults = ['--auto-color', '--auto-progress']
eggs =
    ${instance:eggs}
defaults = ['--auto-color', '--auto-progress', '-q', '--module', '^vs[.]']
initialization =
    import warnings
    warnings.simplefilter('ignore', DeprecationWarning)
```

eggs Liste der zu testenden Eggs wobei jedes Egg in einer neuen Zeile stehen sollte.

defaults Standardoptionen, die üblicherweise als Python list literal angegeben werden.

--ndiff Wenn ein Doctest fehlschlägt, wird das `ndiff.py`-Utility zum Anezigen der Unterschiede verwendet. Alternativen zu dieser Angabe sind:

- `--udiff` für Unified Diffs
- `--cdiff` für Context Diffs.

Weitere Informationen erhalten Sie in der Dokumentation auf PyPI zu zc.recipe.testrunner.

Alle Eggs im Projekt testen

plone.recipe.alltests erlaubt das Testen aller Eggs eines Buildout-Projekts. `bin/alltests` durchläuft alle Tests aller Abhängigkeiten des Hauptprodukts. Hierzu sind lediglich folgende drei Zeilen in der `buildout.cfg`-Datei hinzuzufügen:

```
[buildout]
...
parts =
    ...
    instance
    test
    alltests
...
[alltests]
recipe = plone.recipe.alltests
```

Darüberhinaus können noch folgende Optionen angegeben werden:

eggs Eine Liste von Paketen, die getestet werden sollen.

Der Standardwert sind die im `[tests]`-Abschnitt angegebenen Eggs.

test-script Der Ort im Dateisystem von `zc.recipe.testrunner`.

Der Standardwert ist `bin/test`

exclude Eine Liste von Eggs, die aus dem Testen ausgeschlossen werden sollen. Als Werte können reguläre Ausdrücke angegeben werden, z.B.:

```
[alltests]
recipe = plone.recipe.alltests
exclude =
    repoze.*
```

groups Ein Buildout-Abschnitt mit einem Mapping von Gruppen- zu Paketnamen, z.B.:

```
[alltests]
recipe = plone.recipe.alltests
groups = test-groups

[test-groups]
Zope2 =
    Acquisition
    DateTime
    ExtensionClass
    Persistence
ZODB =
    transaction
    zc.lockfile
chameleon =
    chameleon.core
    cmf.pt
    z3c.pt
```

package-map Ein Buildout-Abschnitt mit einem Mapping von Distributions- zu Paketnamen, z.B.:

```
[alltests]
recipe = plone.recipe.alltests
groups = test-groups

[package-map]
Plone = Products.CMFPlone
```

Testabdeckung (Code Coverage)

Der Zope Test Runner ermöglicht auch, die Testabdeckung eines Produkts zu ermitteln:

```
./bin/test -s vs.registration --coverage=coverage
```

Für eine schnelle Statusübersicht erhalten Sie folgendes Ergebnis:

```
  Ran 19 tests with 0 failures and 0 errors in 43.924 seconds.
Tearing down left over layers:
  Tear down Products.PloneTestCase.layer.PloneSite in 2.009 seconds.
  Tear down Products.PloneTestCase.layer.ZCML in 0.012 seconds.
lines   cov%   module   (path)
   14    57%   vs.registration.__init__   (/home/veit/myproject/src/vs.registration/
↪vs/registration/__init__.py)
   66    63%   vs.registration.browser.enquiry   (/home/veit/myproject/src/vs.
↪registration/vs/registration/browser/enquiry.py)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    7   100%   vs.registration.browser.registrant    (/home/veit/myproject/src/vs.
↪registration/vs/registration/browser/registrant.py)
   26    76%   vs.registration.browser.registration    (/home/veit/myproject/src/vs.
↪registration/vs/registration/browser/registration.py)
   37   100%   vs.registration.content.registrant    (/home/veit/myproject/src/vs.
↪registration/vs/registration/content/registrant.py)
   44   100%   vs.registration.content.registration    (/home/veit/myproject/src/vs.
↪registration/vs/registration/content/registration.py)
   22   100%   vs.registration.interfaces    (/home/veit/myproject/src/vs.registration/
↪vs/registration/interfaces.py)
   64   100%   vs.registration.portlets.registrants    (/home/veit/myproject/src/vs.
↪registration/vs/registration/portlets/registrants.py)
   25    96%   vs.registration.setuphandlers    (/home/veit/myproject/src/vs.
↪registration/vs/registration/setuphandlers.py)
   17   100%   vs.registration.tests.base    (/home/veit/myproject/src/vs.registration/
↪vs/registration/tests/base.py)
   12    91%   vs.registration.tests.test_doctest    (/home/veit/myproject/src/vs.
↪registration/vs/registration/tests/test_doctest.py)
   74   100%   vs.registration.tests.test_portlet_registrants    (/home/veit/myproject/
↪src/vs.registration/vs/registration/tests/test_portlet_registrants.py)
   22   100%   vs.registration.tests.test_setup    (/home/veit/myproject/src/vs.
↪registration/vs/registration/tests/test_setup.py)
```

Ausführlichere Informationen für jede Testdatei sind im Verzeichnis `coverage` enthalten, z.B. für `vs.registration.browser.registration` in `coverage/vs.registration.browser.registration.cover`:

```
       """Define a browser view for the Registration content type. In the FTI
       configured in profiles/default/types/*.xml, this is being set as the default
       view of that content type.
    1: """

    1: from Acquisition import aq_inner
    1: from Products.Five.browser import BrowserView
    1: from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile

    1: from Products.CMFCore.utils import getToolByName

    1: from vs.registration.interfaces import IRegistration
    1: from vs.registration.interfaces import IRegistrant

    1: from plone.memoize.instance import memoize

    2: class RegistrationView(BrowserView):
           """Default view of a registration
           """

           # This template will be used to render the view. An implicit variable
           # 'view' will be available in this template, referring to an instance
           # of this class. The variable 'context' will refer to the registration
           # being rendered.

    1:     __call__ = ViewPageTemplateFile('registration.pt')

    1:     @memoize
            def registrations(self):
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
>>>>>>          context = aq_inner(self.context)
>>>>>>          catalog = getToolByName(context, 'portal_catalog')
>>>>>>          return [ dict(url=registration.getURL(),
                              title=registration.Title,
                              description=registration.Description,)
                         for registration in
                            catalog(object_provides=IRegistration.__identifier__,
>>>>>>                                path=dict(query='/'.join(context.
↪getPhysicalPath()),
>>>>>>                                          depth=1),
>>>>>>                                sort_on='sortable_title')
                       ]

...
```

Dies entspricht exakt der Datei, jedoch ist den meisten Zeilen eine Zahl vorangestellt:

n diese Zeile wurde n-mal aufgerufen.

>>>>>> diese Zeile wurde nicht aufgerufen.

Die Anzahl der durch den Test nicht aufgerufenen Zeilen durch die gesamte Anzahl der Zeilen ergibt die Testabdeckung in Prozent.

Testabdeckung für mehrere Eggs

Auch für mehrere Eggs lässt sich die Testabdeckung mit `zc.recipe.testrunner` angeben:

```
[buildout]
...
parts =
    ...
    test
    coverage

[test]
recipe = zc.recipe.testrunner
eggs =
    ${instance:eggs}

[coverage]
recipe = zc.recipe.egg
eggs = coverage
initialization =
    include = '--source=${buildout:directory}/src'
    sys.argv = sys.argv[:] + ['run', include, 'bin/test', '--all']
```

Report erstellen

`coverage` erstellt einen übersichtlichen Report über die Testabdeckung. Um einen solchen Report zu erhalten, kann einfach ein entsprechender Abschnitt in der `buildout.cfg`-Datei eingetragen werden:

```
[buildout]
...
parts =
    ...
    report
...
[report]
recipe = zc.recipe.egg
eggs = coverage
scripts = coverage=report
initialization =
    sys.argv = sys.argv[:] + ['html', '-i']
```

Anschließend können Sie `file:///home/veit/vs_buildout/htmlcov/index.html` in Ihrem Browser öffnen und erhalten einen übersichtlichen Report:

Coverage report: 16%

Module	*statements*	*missing*	*excluded*	*coverage*
src/vs.org/vs/__init__	5	3	0	40%
src/vs.org/vs/org/__init__	11	3	0	73%
src/vs.org/vs/org/config	4	0	0	100%
src/vs.org/vs/org/tests/__init__	0	0	0	100%
src/vs.org/vs/org/tests/base	58	56	0	3%
src/vs.org/vs/org/tests/test_department	22	19	0	14%
src/vs.org/vs/org/tests/test_employee	36	34	0	6%
src/vs.org/vs/org/tests/test_institution	62	58	0	6%
src/vs.org/vs/org/tests/test_numbers	43	41	0	5%
src/vs.org/vs/org/validators/__init__	3	0	0	100%
src/vs.org/vs/org/validators/validators	31	16	0	48%
Total	**275**	**230**	**0**	**16%**

coverage.py v3.4

Coverage for **src/vs.org/vs/org/validators/validators** : 48%
31 statements | 15 run | 16 missing | 0 excluded

```
1  ############################################################
2  # vs.org (C) 2011, Veit Schiele
3  ############################################################
4
5  # $Id$
6
7  import re
8
9  from zope.interface import implements
10 from Products.validation.interfaces.IValidator import IValidator
11 from Products.validation.validators.RegexValidator import RegexValidator
12
13 class VSAddressValidator(object):
14
15     implements(IValidator)
16
17     def __init__(self, name, title='', description=''):
18         self.name = name
19         self.title = title or name
20         self.description = description
21         self.re_poBox = re.compile('^\d+$')
22         self.re_zipCode = re.compile('^\d+$')
23         self.re_geoCode = re.compile('^\s*\d+\.\d+\s*,\s*\d+\.\d+\s*$')
24
25     def __call__(self, value, *args, **kwargs):
26         error_msg = ""
27         for val in value[:-1]:
28             if val['poBox']:
29                 if val['street']:
30                     error_msg = "Dont use street and postalBox together"
31
```

Unit Tests schreiben

Tests erstellen

Statt der bereits angelegten Datei `src/vs.policy/vs/policy/tests.py` erstellen wir ein eigenes `tests`-Modul:

```
$ rm -rf src/vs.policy/vs/policy/tests.py
$ mkdir src/vs.policy/vs/policy/tests
$ touch src/vs.policy/vs/policy/tests/__init__.py
```

Test-Fixture

Anschließend definieren wir im neu erstellten `tests`-Ordner zunächst ein Test-Fixture, eine gleichbleibende Testumgebung mit der Basisklasse `TestCase`, die an den Layer `VS_POLICY_INTEGRATION` gebunden wird. Hierzu erstellen wir im `tests`-Verzeichnis die Datei `base.py` mit folgendem Inhalt:

```
import unittest2 as unittest

from plone.testing import z2
from plone.app.testing import TEST_USER_NAME
from plone.app.testing import TEST_USER_PASSWORD

from vs.policy.tests import layer

def get_browser(app, loggedIn=True):
    browser = z2.Browser(app)
    if loggedIn:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        auth = 'Basic %s:%s' % (TEST_USER_NAME, TEST_USER_PASSWORD)
        browser.addHeader('Authorization', auth)
    return browser

class TestCase(unittest.TestCase):
    layer = layer.VS_POLICY_INTEGRATION

class FunctionalTestCase(unittest.TestCase):
    layer = layer.VS_POLICY_FUNCTIONAL
```

In `layer.py` werden anschließend die Test-Layer `VS_POLICY_INTEGRATION` und `VS_POLICY_FUNCTIONAL` definiert, die beide auf `VS_POLICY_LAYER` basieren:

```
from plone.app.testing import applyProfile
from plone.app.testing import PloneFixture
from plone.app.testing import PloneSandboxLayer
from plone.app.testing import PloneTestLifecycle
from plone.app.testing import setRoles
from plone.app.testing import TEST_USER_ID
from plone.testing import z2
from zope.configuration import xmlconfig

class VsPolicyFixture(PloneFixture):
    # No sunburst please
    extensionProfiles = ()

VS_POLICY_FIXTURE = VsPolicyFixture()

class VsPolicyTestLifecycle(PloneTestLifecycle):
    defaultBases = (VS_POLICY_FIXTURE, )

class IntegrationTesting(VsPolicyTestLifecycle, z2.IntegrationTesting):
    pass

class FunctionalTesting(VsPolicyTestLifecycle, z2.FunctionalTesting):
    pass

class VsPolicyLayer(PloneSandboxLayer):
    defaultBases = (VS_POLICY_FIXTURE, )

    def setUpZope(self, app, configurationContext):
        import vs.policy

        xmlconfig.file("configure.zcml", vs.policy,
                       context=configurationContext)
        z2.installProduct(app, 'vs.policy')

    def tearDownZope(self, app):
        z2.uninstallProduct(app, 'vs.policy')

    def setUpPloneSite(self, portal):
        applyProfile(portal, 'vs.policy:default')

        setRoles(portal, TEST_USER_ID, ['Manager'])
        portal.invokeFactory('Folder', 'test-folder')
        setRoles(portal, TEST_USER_ID, ['Member'])
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
VS_POLICY_LAYER = VsPolicyLayer()
VS_POLICY_INTEGRATION = IntegrationTesting(
    bases=(VS_POLICY_LAYER, ), name="VsPolicyLayer:Integration")
VS_POLICY_FUNCTIONAL = FunctionalTesting(
    bases=(VS_POLICY_LAYER, ), name="VsPolicyLayer:Functional")
```

Tests

Die eigentlichen Tests werden in der Datei `test_test.py` definiert:

```
from vs.policy.tests.base import FunctionalTestCase

class TestTest(FunctionalTestCase):

    def test_test(self):
        self.assertTrue(True)
```

Unit Tests, die auf dem Python unittest-Modul, ZopeTestCase und PloneTestCase basieren, müssen sich an einige Namenskonventionen halten:

- Alle Testdateien müssen mit `test` beginnen, z.B. `test_setup.py`.
- In den Testdateien werden Klassen für Testfälle definiert, die ein oder mehrere Testmethoden enthalten können, die ebenfalls mit `test` beginnen müssen, z.B. `test_portal_title`.
- Zunächst wird die Basisklasse importiert, dann die Klassen für die Testfälle und schließlich die Test Suite selbst definiert.
- Jede Testsuite kann aus mehreren Testklassen bestehen. Wird die Testsuite ausgeführt, werden alle Testmethoden aller Testklassen der Test-Suite ausgeführt.
- Innerhalb einer Testklasse kann die `afterSetUp()`-Methode unmittelbar vor jedem Test aufgerufen werden um Testdaten für diesen Test anzugeben. Nachdem der Test durchgeführt wurde, werden die Transaktionen zurückgenommen, so dass normalerweise keine Artefakte zurückbleiben.
- Werden jedoch Änderungen außerhalb von Zope vorgenommen, müssen diese mit der Methode `beforeTearDown()` aufgeräumt werden.
- Die in einer Testklasse verwendeten Methoden wie `self.assertEqual()` oder `self.failUnless()` sind Assertion-Methoden, und wenn eine von ihnen fehlschlägt, gilt der ganze Test als fehlgeschlagen.

Test- und Hilfsmethoden

Testmethoden überprüfen, ob etwas wahr oder falsch ist. Daher kann aus den Tests auch herausgelesen werden, wie sich Ihr Produkt verhalten soll, welche Fähigkeiten es enthält. Die Liste der Testmethoden ist ausführlich in der Python-Dokumentation für unittest.TestCaseObjects enthalten. Die häufigsten sind:

`failUnless(expr)` stellt sicher, dass der Ausdruck `expr` wahr ist.

`assertEqual(expr1, expr2)` stellt sicher,dass `expr1` gleich `expr2` ist.

`assertRaises(exception, callable, ...)` stellt sicher, dass beim Aufruf von `callable` die Fehlermeldung `exception` ausgegeben wird.

Bemerkung: `callable` sollte der Name einer Methode oder ein aufrufbares Objekt sein, nicht ein aktueller Aufruf, z.B.:

```
self.assertRaises(AttributeError, myObject.myMethod, someParameter)
```

`fail()` Dies ist sinnvoll, wenn ein Test noch nicht fertiggestellt ist oder in einem `if`-Statement, das deutlich macht, dass der Test fehlgeschlagen ist.

ZopeTestCase und PloneTestCase fügen zu den Assertion-Methoden noch weitere hilfreiche Methoden und Variablen hinzu, die mit Zope interagieren. Hier nur kurz die wesentlichen Variablen:

`self.portal` Die PloneSite, in der der Test ausgeführt wird.

`self.folder` Der `member`-Ordner des Mitglieds, als der die Tests ausgeführt werden.

Und hier die wesentlichen Hilfsmethoden:

`self.logout()` abmelden, d.i. die Rolle `anonymous` bekommen;

`self.login()` sich erneut anmelden; wird ein Nutzername mit übergeben, erfolgt die Anmeldung als dieser Nutzer.

`self.setRoles(roles)` durchläuft eine Liste von Rollen, die angenommen werden sollen.

`self.setRoles((Manager,))` lässt Sie beispielsweise die Rolle des Managers für eine bestimmte Zeit annehmen.

`self.setPermissions(permissions)` analog können auch Berechtigungen für den Testnutzer in `self.folder` angegeben werden;

`self.setGroups(groups)` eine Liste von Gruppen, der der aktuelle Nutzer angehören soll.

Mehr über Unit Tests in Python erfahren Sie in der unittest-Python- Dokumentation.

Testen

Der Testrunner kann nun gestartet werden mit:

```
$ ./bin/test -s vs.policy
```

Wären die Tests geschrieben worden, bevor die Profile erstellt wurden, hätten beide Tests fehlschlagen müssen und der Testrunner folgendes ausgeben:

```
AssertionError:"Welcome to Veit Schiele != ''
...
AssertionError:'Veit Schiele != 'Plone site'
Ran 2 tests with 2 failures and 0 errors
```

Nachdem die Profile angelegt wurden, sollte jedoch keiner der Tests fehlschlagen:

```
Ran 2 tests with 0 failures and 0 errors.
```

Filter

-s my.package, --package my.package, --dir my.package durchsucht die angegebenen Verzeichnisse nach Tests.

-m test_setup, --module test_setup spezifiziert ein Testmodul als regulären Ausdruck, z.B.:

```
$ ./bin/test -s my.package  -m 'test_setup'
```

-t '.*installed.*', --test test_theme_installed spezifiziert einen Testfilter als regulären Ausdruck, z.B.:

```
$ ./bin/test -s vs.policy -m '.*setup.*' -t '.*installed.*'
```

Hiermit werden im Paket `vs.policy` alle, mit `installed` endenden, Methoden in allen Testmodulen, die auf `setup` enden, durchlaufen.

-u, --unit durchläuft ausschließlich Unit tests und ignoriert andere `layer`- Optionen.

-f, --non-unit durchläuft alle Tests, die keine Unit Tests sind

Report

-v, --verbose führt zu ausführlicherer Ausgabe

--ndiff falls ein Doctest fehlschlägt, wird `ndiff.py` zur Darstellung der Unterschiede verwendet

--udiff When there is a doctest failure, show it as a unified diff. falls ein Doctest fehlschlägt, wird Unified Diff zur Darstellung der Unterschiede verwendet

--cdiff falls ein Doctest fehlschlägt, wird Context Diff zur Darstellung der Unterschiede verwendet

Analyse

-d, post-mortem stoppt die Ausführung nach dem ersten nicht-bestandenen Test und ermöglicht *post-mortem*-Debugging, d.h. die Debug-Session wird nur gestartet, wenn ein Test fehlschlägt.

Setup

--path src/my.package fügt einen Pfad zu Pythons Suchpfad hinzu, wobei die Option mehrfach angegeben werden kann.

Weitere Optionen

Diese erhalten Sie mit:

```
$ ./bin/test --help
```

Wenn die relevanten Tests erfolgreich verliefen, sollten schließlich noch alle Tests durchgeführt werden um sicherzustellen, dass nicht an anderer Stelle etwas gebrochen ist. Wenn alle Tests erfolgreich durchlaufen wurden, erscheint eine Meldung:

```
Ran 10 tests with 0 failures and 0 errors in 4.830 seconds.
```

Falls nicht alle Tests erfolgreich durchlaufen wurden, ändert sich die Meldung:

```
Ran 10 tests with 2 failures and 3 errors in 9.688 seconds.
```

Dabei wurden dann zwei Tests nicht bestanden und drei Tests enthielten Fehler.

roadrunner

roadrunner ist ein Testrunner, der die testgetriebene Entwicklung deutlich beschleunigen kann, da er vorab das Standard-Zope- und Plone-Environment für PloneTestCase läd. zur Installation wird einfach folgendes in die `buildout.cfg`-Datei eingetragen:

```
[buildout]
parts =
    ...
    roadrunner

[roadrunner]
recipe = roadrunner:plone
packages-under-test = vs.policy
```

Anschließend kann es wie der reguläre Zope-Testrunner aufgerufen werden:

```
$ ./bin/roadrunner -s vs.policy
```

Tipps & Tricks

- Übernehmen Sie Tests z.B. aus Plone wenn diese Ihren eigenen Absichten entsprechen.
- Dummy-Implementierungen sind häufig der einzige Weg um bestimmte Funktionen zu testen. Siehe auch CMFPlone/tests/dummy.py für einige Dummy-Objekt-Beispiele.
- Tests können auch verwendet werden um Dinge auszuprobieren – sie sind eine sichere Umgebung.
- Während des Debugging können `print`-Statements in den Test eingefügt werden um nachvollziehbare Hinweise im Terminal zu erhalten.
- Es kann jedoch auch gleich der Python-Debugger in die Testmethoden importiert werden mit:

  ```
  import pdb; pdb.set_trace()
  ```

  Anschließend können Sie mit `r` schrittweise durch den Testkode gehen.

  Mehr zum Python-Debugger erfahren Sie in Debugging und in der Python-Dokumentation.

DocTests

DocTests bieten eine interessante Möglichkeit, um Python-Code zu testen indem die Geschichte einer Methode oder Klasse erzählt wird. In Zope 3 sind DocTests weit verbreitet und werden oft für Unit Tests verwendet. Zudem sind DocTests oft eine gute Möglichkeit, gleichzeitig Dokumentationen und Tests zu schreiben.

Meist werden DocTests in eine einzige Datei geschrieben. Dabei ist jedoch zu beachten, dass nicht strikt zwischen den einzelnen Schritten getrennt wird und daher keine sauberen Unit Tests durchgeführt werden. Alternativ kann auch für jede Methode oder Klasse ein DocTest in deren docstring geschrieben werden. Die Syntax der DocTests ist zwar identisch, aber jeder docstring wird als eigene *Test Fixture* durchgeführt, die eine saubere Trennung der Tests erlaubt.

Hier nun ein einfaches Beispiel aus `/Products/CMFPlone/PloneTool.py`:

```
def normalizeString(self, text, relaxed=False):
    """Normalizes a title to an id.

    normalizeString() converts a whole string to a normalized form that
    should be safe to use as in a url, as a css id, etc.

    If relaxed=True, only those characters that are illegal as URLs and
    leading or trailing whitespace is stripped.

    >>> ptool = self.portal.plone_utils

    >>> ptool.normalizeString("Foo bar")
    'foo-bar'

    ...

    """
    return utils.normalizeString(text, context=self, relaxed=relaxed)
```

Dabei ist zu beachten, dass der docstring der Methode sowohl den einfachen Text enthält, der beschreibt, was die Methode tut, als auch Python-Statements, wie man sie aus einem interaktiven Python-Interpreter kennt.

Und tatsächlich führt der *DocTest Runner* jede Zeile, die mit >>> beginnt, im Python-Interpreter aus. Folgt diesem Statement eine Zeile, die genausoweit eingerückt ist, nicht leer ist und nicht mit >>> beginnt, so wird dies als Ergebnis des Statements erwartet. Stimmen sie nicht überein, gibt doctest eine Fehlermeldung aus.

Hinweis 1 Wird kein Ausgabewert angegeben, wird von der Methode keine Ausgabe erwartet. Gibt die Methode dennoch etwas aus, wirft doctest eine Fehlermeldung aus.

Hinweis 2 `...` bedeutet »beliebig viele Zeichen«. Dies ist sinnvoll für Werte, die nicht vorhergesagt werden können wie automatisch generierte IDs, die auf dem aktuellen Datum oder zufälligen Zahlen beruhen.

Testen mit DocTest

DocTests sind ein Feature von Python 2 und daher kann Python's `unittest`-Library für einfache Doctests verwendet werden. Zope 3 erweitert die Funktionalität noch um das `zope.testing`-Modul:

```
import unittest
from zope.testing import doctest

def test_suite():
    return unittest.TestSuite((
        doctest.DocFileSuite('README.txt'),
        ))

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Wird die Datei z.B. als `test_doctests.py` im `tests`-Verzeichnis Ihres Produkts gespeichert, kann es mit den üblichen Aufrufen gestartet werden. Wenn Sie sich Zope3-DocTests anschauen, können Sie häufig feststellen, dass in den ersten Zeilen die Komponentenarchitektur oder einzelne Komponenten explizit geladen werden.

Um die Testumgebung für DocTests anzugeben, können Sie z.B. folgendes eingeben:

```
import unittest
import doctest

from zope.testing import doctestunit
from zope.component import testing, eventtesting

from Testing import ZopeTestCase as ztc

from vs.registration.tests import base

def test_suite():
    return unittest.TestSuite([

        # Demonstrate the main content types
        ztc.ZopeDocFileSuite(
            'README.txt', package='vs.registration',
            test_class=base.RegistrationFunctionalTestCase,
            optionflags=doctest.REPORT_ONLY_FIRST_FAILURE | doctest.NORMALIZE_
↪WHITESPACE | doctest.ELLIPSIS),

        ])

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Dieser Test ist aus `vs.registration/tests/test_doctest.py` entnommen. Er führt dabei die `README.txt`-Datei aus `vs.registration/vs/registration/` aus.

Hinweis 3 Mit diesem Setup referenziert die Variable `self` auf eine PloneTestCase-Instanz. Demzufolge können Sie z.B. folgendes angeben:

```
>>> self.portal.invokeFactory('Registration', 'my-first-registration')
```

um ein Dokument im Wurzelverzeichnis Ihrer Site anzulegen. Auch alle anderen Methoden aus PloneTestCase und ZopeTestCase sollten innerhalb von DocTests laufen.

Hinweis 4 In `optionflags` lassen sich Optionen und Anweisungen für `docteststs` angeben, u.a.:

`doctest.NORMALIZE_WHITESPACE` Wenn angegeben, werden alle Abfolgen von Leerzeichen und/oder Zeilenumbrüchen als gleich betrachtet.

`doctest.ELLIPSIS` Wenn angegeben, kann eine Ellipse `...` in der erwarteten Ausgabe auf jede beliebige Zeichenfolge passen.

`doctest.REPORT_ONLY_FIRST_FAILURE` Wenn angegeben, wird der erste fehlgeschlagene Test angezeigt, nicht jedoch der Ausgang der weiteren Tests. Hiermit wird verhindert, dass `doctest` fehlgeschlagene Tests aufgrund von vorangehend gescheiterten Tests ausgibt.

Eine vollständige Übersicht über alle Optionen finden Sie in Option Flags and Directives

DocTest Tipps & Tricks

Dokumentation von DocTest lesen Das DocTest-Modul kommt mit einer umfangreichen Dokumentation.

Ein Test ist eine Reihe von Python-Statements. Sie können z.B. auf Hilfsmethoden in Ihrem Produkt verweisen, angenommen Ihr Produkt enthält die Methode `reset(self)` in `my.package.tests.utils`, so kann diese Methode mit DocTest aufgerufen werden:

```
>>> from my.package.tests.utils import reset
>>> reset()
```

Die Testsuite kann zusätzliche Funktionen einführen Möchten Sie z.B. ein Produkt in obigem Beispiel verfügbar machen, müssen Sie nur `ZopeTestCase.installProduct()` in der testsuit-Datei Ihres Produkts aufrufen.

Debugging Wenn Sie `import pdb; pdb.set_trace()` in Ihren DocTest einfügen, können Sie zwar nicht schrittweise durch Ihren Kode gehen, aber Variablen und der Status der *Test Fixture* kann mit `print` ausgegeben werden.

Dabei sollten Sie jedoch beachten, dass sich `locals` auf Interna von `doctest` bezieht:

```
(Pdb) locals()
{'__return__': None, 'self': <zope.testing.doctest._OutputRedirectingPdb instance␣
↪at 0x5a7c8f0>}
```

Das gewohnte Verhalten von `pdb` erhalten Sie indem Sie im Stack eine Ebene nach oben gehen:

```
(Pdb) up
> /Users/veit/vs_buildout/src/Products.PloneGetPaid/Products/PloneGetPaid/
↪notifications.py(22)__call__()
-> import pdb ; pdb.seT_trace()
(Pdb) locals()
{'settings': <Products.PloneGetPaid.preferences.StoreSettings object at 0x5f631b0>
↪, 'store_url': 'http://nohost/plone', 'self': <Products.PloneGetPaid.
↪notifications.MerchantOrderNotificationMessage object at 0x56c30d0>, 'order_
↪contents': u'11 pz @84.00 total: US$924.00\n22 ph @59.00 total: US$1298.00\n12␣
↪pf @98.00 total: US$1176.00\n23 pX @95.00 total: US$2185.00\n3 pM @89.00 total:␣
↪US$267.00\n22 po @60.00 total: US$1320.00\n23 pj @39.00 total: US$897.00\n15 po␣
↪@34.00 total: US$510.00\n5 pS @76.00 total: US$380.00\n1 pm @70.00 total: US$70.
↪00', 'template': u'To: ${to_email}\nFrom: "${from_name}" <${from_email}>\
↪nSubject: New Order Notification\n\nA New Order has been created\n\nTotal Cost:
↪${total_price}\n\nTo continue processing the order follow this link:\n${store_
↪url}/@@admin-manage-order/${order_id}/@@admin\n\nOrder Contents\n\n${order_
↪contents}\n\nShipping Cost: ${shipping_cost}\n\n', 'pdb': <module 'pdb' from '/
↪Users/moo/code/python-macosx/parts/opt/lib/python2.4/pdb.pyc'>}
(Pdb)
```

Weitere Informationen zum Debugging erhalten Sie in der Python Dokumentation.

Interlude erlaubt die Verwendung einer interaktiven Shell innerhalb von Doctests, wobei die oben beschriebene Besonderheit nicht auftritt:

```
>>> from interlude import interact
>>> interact(locals())
```

Wenn der `testrunner` nun `interact` durchläuft, erhalten Sie eine interaktive Python-Konsole.

Exceptions (Ausnahmen) ausgeben Folgender Code gibt Exceptions aus:

```
>>> try:
...     someOperation()
... except:
...     import pdb; pdb.set_trace()
>>> # continue as normal
```

Unit Tests als DocTests schreiben

DocTests als Unit Tests haben den Vorteil, dass sie direkt in docstrings beim auszuführenden Code stehen. Darüberhinaus bietet die Syntax von DocTests viele Annehmlichkeiten, die das Schreiben von Tests angenehmer und schneller machen.

Im folgenden zeige ich ein einfaches Beispiel aus Archetypes:

```
[...]

class ReferenceSource(object):
    """A base implementation of an IReferenceSource adapter that
    relies on a IReferenceQuery local utility to look up targets and
    on an IReferenceStorage adapter to add references.

    Basic setup:

        >>> import archetypes.testing
        >>> archetypes.testing.setupPortal()
        >>> archetypes.testing.wireUp()

    We create an object that we adapt to our ReferenceSource:

        >>> obj = archetypes.testing.FakeObject('sourceobj')
        >>> source = ReferenceSource(obj)

    We don't have any targets or relationships at this point:

        >>> source.getTargets()
        []
        >>> source.getRelationships()
        set([])

    [...]

    Teardown:

        >>> archetypes.testing.teardownPortal()
    """

    [...]
```

Der Test Runner schaut nach DocTests innerhalb von docstrings, und führt jeden von ihnen als seine eigene Test Fixture, d.h. unabhängig von anderen Tests, aus. In diesem Beispiel werden Tests für jede Klasse geschrieben, die zugleich dokumentieren, wie die Klasse verwendet wird.

docstring DocTests ausführen

Um docstring-DocTests ausführen zu können, muss noch eine Test Suite hinzugefügt werden, die dem Test Runner sagt, dass er nach diesen docstrings suchen soll. In den meisten Fällen genügt eine einfache Test Suite, die das Zope-3-Framework nutzt, wie z.B. test_doctests.py, die auf die o.g. `base.py` referenziert:

```
import unittest
from zope.testing import doctest

optionflags = doctest.REPORT_ONLY_FIRST_FAILURE | doctest.ELLIPSIS

def test_suite():
    from zope.testing.doctestunit import DocTestSuite
    return unittest.TestSuite((
            DocTestSuite('archetypes.reference.base',
                         optionflags=optionflags),
            ))

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Wenn Sie PloneTestCase benötigen, sieht Ihre Test Suite z.B. so aus:

```
import unittest
from zope.testing import doctest
from Testing.ZopeTestCase import ZopeDocTestSuite

from Products.pluggablecatalog.tests import common
common.setupPloneSite()

from Products.PloneTestCase import PloneTestCase

optionflags =  (doctest.ELLIPSIS |
                doctest.NORMALIZE_WHITESPACE |
                doctest.REPORT_ONLY_FIRST_FAILURE)

def test_suite():
    return unittest.TestSuite(
        [ZopeDocTestSuite(module,
                          test_class=PloneTestCase.PloneTestCase,
                          optionflags=optionflags)
         for module in ('Products.pluggablecatalog.tool',)]
       )
```

Diese Test Suite ist testDoctests des pluggablecatalog-Produkts entnommen.

Und `common` enthält einfach:

```
def setupPloneSite():
    from Products.PloneTestCase import PloneTestCase
    PloneTestCase.installProduct('pluggablecatalog')
    PloneTestCase.setupPloneSite()
```

DocTest Unit Tests in separaten Dateien

Falls Ihnen die Tests zu umfangreich erscheinen oder Sie sie gern in einer separaten Datei hätten, können Dummy-Methoden in einem normalen Unit Test erstellt werden, die docstrings mit DocTests enthalten. Diese DocTests sollten beschreibenden Text **und** die entsprechenden Testmethoden enthalten.

Das folgende Beispiel ist aus der i18n-Architektur von Plone 3 entnommen: test_countries.py. Es bezieht sich stark auf die Zope 3-Komponentenarchitektur, lädt ZCML-Dateien und bezieht das Test-Setup von `zope.component.testing`:

```
import unittest

[...]

from zope.component.testing import setUp, tearDown
from zope.configuration.xmlconfig import XMLConfig
from zope.testing import doctest
from zope.testing.doctestunit import DocTestSuite

def configurationSetUp(self):
    setUp()
    XMLConfig('meta.zcml', zope.component)()
    XMLConfig('meta.zcml', zope.app.publisher.browser)()
    XMLConfig('configure.zcml', plone.i18n.locales)()

def testAvailableCountries():
    """
      >>> util = queryUtility(ICountryAvailability)
      >>> util
      <plone.i18n.locales.countries.CountryAvailability object at ...>

      >>> countrycodes = util.getAvailableCountries()
      >>> len(countrycodes)
      243

      >>> 'de' in countrycodes
      True

      >>> countries = util.getCountries()
      >>> len(countries)
      243

      >>> de = countries['de']
      >>> de['name']
      'Germany'

      >>> de['flag']
      '/@@/country-flags/de.gif'
    """

def test_suite():
    return unittest.TestSuite((
        DocTestSuite('plone.i18n.locales.countries'),
        DocTestSuite(setUp=configurationSetUp,
                     tearDown=tearDown,
                     optionflags=doctest.ELLIPSIS | doctest.NORMALIZE_WHITESPACE),
        ))
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
if __name__ == '__main__':
    unittest.main(defaultTest="test_suite")
```

Funktionale und Systemtests mit zope.testbrowser

Während unit tests und DocTests die Gültigkeit von einzelnen Methoden und Modulen überprüfen, überprüfen funktionale Tests die Anwendungen als Ganzes. Häufig nehmen sie dabei die Sicht des Nutzers ein und üblicherweise orientieren sie sich an den Nutzungsfällen (Use cases) der Anwendung. Systemtests hingegen testen die Anwendung als Blackbox.

Mit Zope 3 kommt die Bibliothek `zope.testbrowser`, die es ermöglicht, DocTests zu schreiben, die sich wie ein Webbrowser verhalten. Sie können URLs öffnen, Links anklicken, Formularfelder ausfüllen und abschicken und dann die zurückgelieferten HTTP headers, URLs und Seiteninhalte überprüfen.

Bemerkung: Da der `testbrowser` momentan kein JavaScript unterstützt, empfiehlt sich zum Testen dynamischer User Interfaces selenium. Selenium wird auch vom Robot Test Automatisation Framework verwendet.

Funktionale Tests sind kein Ersatz für Unit tests sondern überprüfen den funktionalen Ablauf, wie ihn der Nutzer wahrnimmt. Ein funktionaler Test überprüft z.B., ob ein Delete-Button vorhanden ist und wie erwartet funktioniert. Um die Tests überschaubar zu halten, wird meist nur überprüft, ob die entsprechenden Templates vorhanden sind und für Nutzer mit verschiedenen Rollen und Rechten wie erwartet funktioniert.

Im folgenden nun ein Auszug aus einem `zope.testbrowser`-Test aus `vs.registration`:

```
Setting up and logging in
-------------------------

    >>> from Products.Five.testbrowser import Browser
    >>> browser = Browser()
    >>> portal_url = self.portal.absolute_url()
    [...
    >>> from Products.PloneTestCase.setup import portal_owner, default_password

    >>> browser.open(portal_url + '/login_form?came_from=' + portal_url)
    >>> browser.getControl(name='__ac_name').value = portal_owner
    >>> browser.getControl(name='__ac_password').value = default_password
    >>> browser.getControl(name='submit').click()
```

`zope.testbrowser` wird verwendet um die Interaktion mit einem Browser zu simulieren. Dies sind keine reinen funktionalen Tests, da auch der Zustand der ZODB überprüft und verändert wird.

Anschließend meldet sich der Test als Eigentümer des Portals im `login_form`- Formular an.

Alle Aktionen finden im `browser`-Objekt statt. Dies simuliert einen Webbrowser mit einer Schnittstelle zum Finden und Ausführen von Form controls und Links. Mit den Variablen `browser.url` und `browser.contents` lassen sich sowohl die URL als auch die Inhalte überprüfen. `zope.testbrowser` bietet eine umfangreiche Dokumentation in seiner *README.txt <http://svn.zope.org/Zope3/trunk/src/zope/testbrowser/README.txt?view=auto>*-Datei. Die bedeutendsten Methoden des IBrowser interface sind:

`open(url)` öffnet eine gegebene URL.

`reload()` lädt die aktuelle Seite erneut.

`goBack(count=1)` simuliert, wie häufig der zurück-Button gedrückt wird.

getLink(text=None, url=None, id=None) sucht nach einen Link entweder mit dem in <a>-Tags angegebenen Inhalt, der URL im `href`-Attribut oder der ID des Links.

Mit `click()` kann dieser Link ausgeführt werden.

getControl(label=None, name=None, index=None) gibt den inhalt eines Formularfeldes aus, das entweder durch sein Label oder seinen Namen gefunden wird. Das `index`-Argument wird verwendet, wenn mehrere form controls vorliegen (`index=0` ist z.B. die erste form control).

Und auch hier kann mit `click()` das Klicken auf das Kontroll-Objekt ausgeführt werden.

Das IBrowser-Interface bietet einige Eigenschaften (Properties), die zum Abfragen des Zustands der aktuellen Seite verwendet werden können:

url die vollständige URL der aktuellen Seite.

contents Den vollständigen Inhalt der aktuellen Seite als string (normalerweise mit HTML-Tags).

headers die HTTP headers.

Detailliertere Angaben zu weiteren Methoden, Attributen, Schnittstellen und Beispielen für die verschiedenen Arten von Links und Controls erhalten Sie in Interfaces und der README.txt-Datei.

Funktionale Tests ablaufen lassen

Da ein Test mit dem `testbrowser` ein normaler DocTest ist, kann er mit üblichem testrunner und Testsuite-Setup laufen.

Bemerkung: Die Site mit dem `zope.testbrowser` aufzusetzen wäre unnötig kompliziert, zumal damit nur Plone getestet würde und nicht `vs.registration`.

Funktionale Tests debuggen

Wenn der `testbrowser` zu einer Zope-Fehlermeldung führt, kann es schwierig werden, die Ursache zu ermitteln. Zwei Änderungen vereinfachen dies deutlich.

Zunächst sollten Sie sicherstellen, dass die Fehler auch angezeigt werden:

```
>>> browser.handleErrors = False
```

Damit werdenIhnen die vollständigen Zope-Exceptions angezeigt.

Und wenn PloneTestCase verwendet wird, kann auch Plone's Error log verwendet werden:

```
>>> self.portal.error_log._ignored_exceptions = ()
```

Damit werden Fehler wie *NotFound* und *Unauthorized* im Error log angezeigt werden. Zudem kann es sinnvoll sein, *Verbose Security* zu aktivieren.

Anschließend kann statt einer Zeile, die zu einem Fehler führte, z.B. folgender Kode eingegeben werden:

```
>>> try:
...     browser.getControl('Save').click()
... except:
...     print self.portal.error_log.getLogEntries()[0]['tb_text']
...     import pdb; pdb.set_trace()
>>> # continue as normal
```

Damit wird der letzte Eintrag in das Error log ausgegeben und ein PDB break point gestzt.

Funktionale Tests vs. Systemtests

Ein Systemtest überprüft ein System als sog. Blackbox. Ein funktionaler Test konzentriert sich auf die geforderten Funktionsabläufe, die meist in Nutzungsfällen (Use Cases) beschrieben sind.

Für einen funktionalen Test mag es akzeptabel sein, Annahmen auf einem festgelegten Status einer Site, der Testsuite, zu machen. Der Systemtest macht hingegen keine solchen Annahmen. Daher benötigt ein `zope.testbrowser`-Test idealerweise nicht das PloneTestCase-Test fixture:

```
import unittest
from zope.testing import doctest

def test_suite():
    return unittest.TestSuite((
        doctest.DocFileSuite('TestSystem.txt'),
        ))

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Abgesehen davon bleiben die verwendeten Methoden für einen Systemtest dieselben.

Aufzeichnen funktionaler Tests mit zope.testrecorder

Der Zope-Testrecorder zeichnet funktionale Tests im Browser auf und speichert sie als ausführbare Tests. Funktionale Tests, die den Zope-Testbrowser nutzen, ersparen uns das erneute Herumklicken im Browser zum Testen des User Interfaces. Der Zope-Testrecorder kann auch Selenium-Tests, eine andere Form von funktionalen Tests, die in Ihrem Webbrowser direkt vor Ihren Augen ablaufen und daher auch JavaScript unterstützen.

Download und Installation

Hierzu müssen Sie `zope.testrecorder` nur in der `buildout.cfg`-Datei eintragen:

```
[buildout]
...
eggs =
    ...
    zope.testrecorder
...

[instance]
...
zcml =
     ...
     zope.testrecorder
```

Anschließend wird das Buildout-Skript aufgerufen und die Instanz neu gesteartet.

Testrecorder zum Aufzeichnen funktionaler Tests

1. Nun sollte der `zope.testrecorder` verfügbar sein unter folgender URL:

   ```
   http://localhost:8080/++resource++recorder/index.html
   ```

   Sie sollten nun folgende Seite sehen können:

   |testrecorder Start|

2. Geben Sie nun die Adresse Ihrer Plone-Site ein, z.B. `http://localhost:8080/mysite`, und klicken auf *Go*.

3. Wollen Sie freien Text in Ihr DocTest-Dokument schreiben, klicken Sie auf die *Add comment*-Taste.

4. Wenn Sie überprüfen wollen, ob bestimmter Text auf Ihrer Seite erscheint, markieren Sie diesen Text, klicken dann bei gedrückter Umschalt-Taste auf diesen Text und wählen *Check text appears on page*.

   |testrecorder Popup zum Überprüfen von Text|

5. Schließlich können Sie die Aufzeichnung beenden und sie als Python DocTest abspeichern. Dann erhalten Sie ungefähr dies:

   ```
   Create the browser object we'll be using.

       >>> from zope.testbrowser import Browser
       >>> browser = Browser()
       >>> browser.open('http://localhost:8080/mysite')

   A test comment.

       >>> 'start writing' in browser.contents
       True
   ```

6. Anschließend können Sie diesen Text in eine DocTest-Datei schreiben und gegebenenfalls Änderungen vornehmen.

7. Lesen Sie `INSTALL.txt` für weitere Anweisungen.

Tipps für den Zope-Testrecorder

Testplan Sie sollten ein Skript erstellen bevor Sie mit dem Aufzeichnen von Tests beginnen. Kommentieren Sie Ihre Tests um festzuhalten, was getestet werden soll.

Bedenken Sie, welchen Link Sie ausführen Überlegen Sie sich, welche Aktionen die allgemeingültigsten und zuverlässigsten Tests bieten.

Setzen Sie die Site vorher auf Wie beim Zope-Testbrowser, sollte auch beim Zope-Testrecorder zunächst eine Site aufgesetzt werden, bevor Sie mit dem Aufzeichnen der Tests beginnen.

Räumen Sie Ihren DocTest auf Bevor Sie das Schreiben von DocTests mit dem Zope-Testrecorder beenden, sollten Sie den Kode noch von Textstellen befreien, deren Konsistenz nicht gewährleistet werden kann, wie z.B.

- aus dem Datum generierte IDs von Inhaltstypen
- und identische oder nahezu identische Textstellen, die in Ihrem DocTest erscheinen.

Javascripts testen

Behavior-Driven-Development mit Jasmine

Jasmine ist ein Behavior-Driven-Development-Framework zum Testen von JavaScript-Code. Dabei benötigt es kein DOM und mit der enfachen Syntax lassen sich leicht Tests erstellen.

Suites und Specs

Erstellen einer Test-Suite

Eine Test-Suite beginnt mit dem Aufruf der globalen Jasmine-Funktion und zwei Paremetern:

1. Der Name oder Titel einer `spec`-Suite
2. Die Funktion, die die Suite implementiert

Specs

Spezifikationen (*specs*) sind definiert durch die globale Jasmine-Funktion *ìt`* mit den Parametern Titel und Funktion.

Dabei kann die Funktion sowohl eine *spec* sein als auch ein Test. Eine *spec* enthält eine oder mehrere Erwartungen (*expectations*), deren Überprüfung entweder wahr oder falsch sein können.

Bewahrheiten sich alle Erwartungen einer *spec* so wird diese bestanden.

Da `describe` und `it` Funktionen sind, können sie jeden ausführbaren Code enthalten, der zur Implementierung eines Tests erforderlich ist. Dabei lassen sich in `describe` definierte Variablen in jedem *ìt*`-Block innerhalb der Suite verwenden.

Asynchrone Specs

Jasmine unterstützt auch das Testen asynchroner Operationen. Entsprechende *Specs* können mit einer Reihe von Blöcken mit `runs`-Aufrufen geschrieben werden, die üblicherweise asynchron abgearbeitet werden.

`waitsFor` wird verwendet mit einer Latch-Funktion, einer Fehlermeldung und einem Timeout.

> Die Latch-Funktion stellt Anfragen bis es `true` zurückerhält oder der Timeout überschritten ist. Wenn der Timeout überschritten wird, ist die *Specs* fehlgeschlagen und gibt die Fehlermeldung aus.

Gruppieren von *Specs* mit `describe`

Mit der `describe`-Funktion lassen sich einfach *specs* gruppieren wobei die Namen der *specs* sich idealerweise zu einem Satz aneinanderreihen lassen.

Setup und Teardown

Jasmine stellt die globalen Funktionen `beforeEach` und `afterEach` bereit, die vor bzw. nach jeder *spec* in `describe` aufgerufen werden. So lässt sich der Initialisierungscode in die `beforeEach`- Funktion verschieben und das Zurücksetzen der Variablen in die `afterEach`-Funktion.

Verschachteln von `describe`-Blöcken

`describe`-Blöcke können verschachtelt werden sodass eine Test-Suite als Funktionsbaum zusammengestellt werden kann. Bevor eine *Spec* ausgeführt wird, läuft Jasmine durch den Baum und führt alle `beforeEach`- Funktionen in der Reihenfolge aus. Umgekehrt werden, nachdem die *Spec* ausgeführt wurde, alle `afterEach`-Funktionen durchlaufen.

Deaktivieren von *Specs* und *Suites*

Suites und *Specs* können deaktiviert werden mit den `xdescribe` und `xit`-Funktionen.

Matchers

Matchers implementieren einen boolschen Vergleich zwischen aktuellen und erwarteten Werten und teilen anschließend Jasmine mit, ob die Erwartungen erfüllt wurden oder nicht.

`toBe` entspricht ===

`toEqual` für einfache Literale, Variablen und Objekte

`toMatch` für reguläre Ausrücke

`toBeDefined` vergleicht mit `undefined`

`toBeUndefined` vergleicht mit `undefined`

`toBeNull` vergleicht mit Null

`toBeTruthy` vergleicht mit boolscher Wahrscheinlichkeit

toBeFalsy vergleicht mit boolscher Wahrscheinlichkeit

toContain vergleicht, ob ein Item in einem Array enthalten ist

toBeLessThan` mathematischer Vergleich

toBeGreaterThan mathematischer Vergleich

toBeCloseTo Präzision des mathematischen Vergleichs

toThrow überprüft, ob eine Funktion eine Fehlermeldung ausgibt.

Jeder *Matcher* kann auch eine negative Annahme überprüfen indem bei `èxpect` ein `not` dem *matcher* vorangestellt wird.

Zwar bringt Jasmine bereits eine ganze Reihe von *Matchers* mit, es gibt jedoch auch die Möglichkeit, eigene *Matchers* für spezifische Annahmen zu schreiben.

Spies

Jasmine's Test-Doubles werden *Spy* genannt. Wesentlich sind momentan sog. Test stubs möglich, für die Jasmine spezielle *Matchers* bereitstellt:

toHaveBeenCalled ist wahr wenn der *Spy* aufgerufen wird

toHaveBeenCalledWith ist wahr wenn die Liste der Argumente übereinstimmt mit den aufgezeichneten Aufrufen des *Spy*.

andCallThrough lässt den *Spy* alle Aufrufe nachverfolgen und zusätzlich an die aktuelle Implementierung übertragen

andReturn gibt einen spezifischen Wert beim Aufruf der Funktion aus

andCallFake gibt die Aufrufe an die angebotene Funktion weiter

createSpy `jasmine.createSpy` kann einen minimalen *Spy* erzeugen, der Aufrufe und Argumente nachverfolgt etc.

createSpyObj `jasmine.createSpyObj` erzeugt ein Mock-Objekt mit mehreren *Spies*

jasmine.any

`jasmine.any` ist wahr, wenn der Name des Konstruktors oder der Klasse dem erwarteten Wert entspricht.

Mock der JavaScript-Clock

Zum Testen von Timeouts und Intervallen kann `setTimeout-` und `setInterval` verwendet werden.

Runner und Reporter

Jasmine ist in JavaScript geschrieben und muss daher in eine JS-Umgebung eingebunden werden.

1. Hierfür wird eine HTML-Seite geschrieben, die die Javascript-Dateien mit dem `<script>`-Tag einbindet sodass alle *Specs* mit Jasmine durchlaufen und aufgezeichnet werden. Daher ist diese HTML-Seite der *Test-Runner*. Sehen Sie hierzu SpecRunner.html.

   Dabei werden folgende Schritte durchlaufen:

   1. Zunächst wird ein `HTMLReporter` erstellt um die Ergebnisse jeder *Spec* und jeder Test-Suite aufzuzeichnen. Der Reporter ist auch für die spätere Darstellung der Ergebnisse zuständig.
   2. Auswählen einzelner Test-Suites oder *Specs*, die Durchlaufen werden sollen.
   3. Durchlaufen aller ausgewählten Tests.

   Diese Seite sollte im `tests`-Modul unseres Pakets unter dem Namen `testRunner.html` abgespeichert werden.

2. Anschließend passen wir die Verweise auf die Quelldateien an, da die von uns zu testenden Javascript-Dateien nicht im `tests`-Modul selbst sondern im `browser`- oder `skins`-Modul liegen werden.
3. Nun kopieren wir noch die folgenden Dateien in `tests/jasmine` und passen die Pfade in `testRunner.html` entsprechend an:
   - jasmine-html.js
   - jasmine.css
   - jasmine.js
   - jasmine_favicon.png
4. Schließlich können wir noch unsere *Specs* schreiben wobei sich bewährt hat, die Javascript- Dateinamen im `tests`-Modul beizubehalten.

Statische Code-Analyse mit jsHint

gocept.jslint integriert jsHint in das Python-Unittest-Modul.

Es bietet eine `JSLint`-TestCase-Klasse, die konfigurierbar alle Javascript-Dateien einsammelt und dynamisch für jede dieser Dateien eine Testmethode generiert. Diese Methoden können einfach verwendet werden, z.B. mit:

```
class MyJSLintTest(gocept.jslint.TestCase):

    include = ('my.package.browser:js',
               'my.package.browser:js/lib')
    options = (gocept.jslint.TestCase.options +
               ('browser', 'jquery',))
```

include ist eine Liste von Pfasen zu Ressourcen, ausgehend von `packagename:path`.

options ist eine Liste von Optinen für jsHint, s.a. Enforcing Options.

predefined Liste globaler Variablen, die mit der `undef`-Option verwendet werden können.

exclude Liste von Dateinamen, die *nicht* getestet werden sollen.

Anforderungen

- Python 2.6 oder Python 2.7
- node.js 0.3
- `jshint`-npm-Modul:

```
$ npm install jshint -g
```

Achten Sie darauf, dass das jshint-Binary in `$PATH` verfügbar ist.

Weitere Ressourcen

- Rebecca Murphey: Writing Testable JavaScript
- Christian Johansen: Test-Driven JavaScript Development
- Anthony Colangelo: The Design of Code: Organizing JavaScript
- cajs: Component Architecture Tools for Javascript
- Grunt

Robot-Framework

Das Robot Framework ist ein generisches Framework zur Durchführung von automatisierten Softwaretests, v.a. Akzeptanztests.

Robot verwendet für das Keyword- Driven Testing eine tabellenartige Struktur zur Verwaltung der Testdaten.

Selenium2Library ermöglicht, die Selenium 2 (WebDriver)-Bibliothek im Robot-Framework zu verwenden.

Weitere Informationen zur Selenium2Library finden Sie im Wiki.

Installation

1. Beim Erstellen eines neuen Pakets sollte darauf geachtet werden, dass bei der Frage nach `robot tests` mit `true` geantwortet wird:

```
$ ../bin/zopeskel plone_basic vs.registration
...
Expert Mode? (What question mode would you like? (easy/expert/all)?) ['easy']: all
...
robot tests (should the default robot test be included) [false]: true
```

Bei einem bestehenden Paket sollte folgendes in die `setup.py`-Datei eingetragen werden:

```
extras_require={
    'test': ['plone.app.testing[robot]>=4.2.2']
},
```

Damit wird neben plone.app.testing noch die robotsuite und die robotframework- selenium2library zum Testen installiert.

Zudem ist in `vs_buildout/src/vs.registration/src/vs/registration/tests/` eine `test_robot.py`-Datei angelegt worden mit:

```
from  vs.registration.testing import VS_REGISTRATION_FUNCTIONAL_TESTING
from plone.testing import layered
import robotsuite
import unittest

def test_suite():
    suite = unittest.TestSuite()
    suite.addTests([
        layered(robotsuite.RobotTestSuite("robot_test.txt"),
                layer=VS_REGISTRATION_FUNCTIONAL_TESTING)
    ])
    return suite
```

Die zugehörige `robot_test.txt`-Datei sieht dann so aus:

```
*** Settings ***

Library  Selenium2Library  timeout=10  implicit_wait=0.5

Suite Setup  Start browser
Suite Teardown  Close All Browsers

*** Variables ***

${BROWSER} =  firefox

*** Test Cases ***

Plone site
    [Tags]  start
    Go to  http://localhost:55001/plone/
    Page should contain  Plone site

*** Keywords ***

Start browser
    Open browser  http://localhost:55001/plone/  browser=${BROWSER}
```

2. Schließlich kann der Test aufgerufen werden mit:

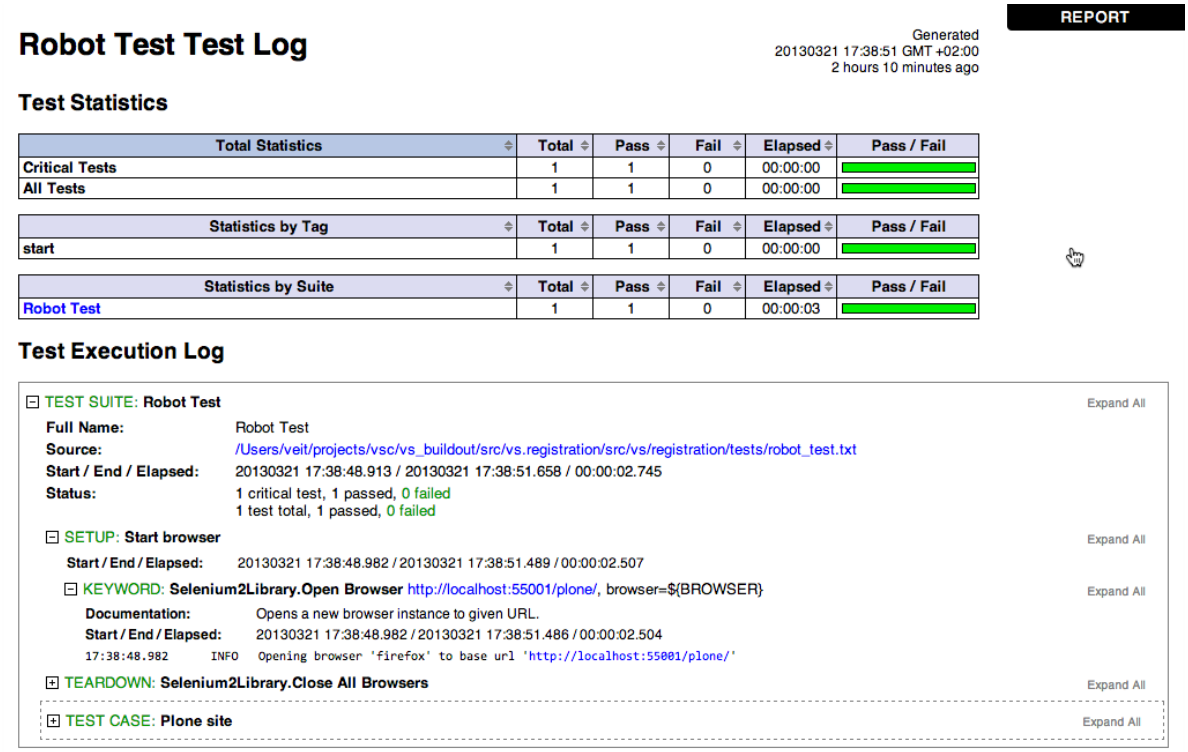
```
$ cd vs_buildout/src/vs.registration
$ python bootstrap.py
$ ./bin/buildout
$ ./bin/test
Running vs.registration.testing.VsregistrationLayer:Functional tests:
  Set up plone.testing.zca.LayerCleanup in 0.000 seconds.
  Set up plone.testing.z2.Startup in 0.394 seconds.
  Set up plone.app.testing.layers.PloneFixture in 10.463 seconds.
  Set up vs.registration.testing.VsregistrationLayer in 0.464 seconds.
  Set up plone.testing.z2.ZServer in 0.503 seconds.
  Set up vs.registration.testing.VsregistrationLayer:Functional in 0.000 seconds.
  Running:

  Ran 1 tests with 0 failures and 0 errors in 3.026 seconds.
Running vs.registration.testing.VsregistrationLayer:Integration tests:
  Tear down vs.registration.testing.VsregistrationLayer:Functional in 0.000␣
↪seconds.
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  Tear down plone.testing.z2.ZServer in 5.152 seconds.
  Set up vs.registration.testing.VsregistrationLayer:Integration in 0.000 seconds.
  Running:

  Ran 1 tests with 0 failures and 0 errors in 0.004 seconds.
Tearing down left over layers:
  Tear down vs.registration.testing.VsregistrationLayer:Integration in 0.000␣
↪seconds.
  Tear down vs.registration.testing.VsregistrationLayer in 0.002 seconds.
  Tear down plone.app.testing.layers.PloneFixture in 0.092 seconds.
  Tear down plone.testing.z2.Startup in 0.007 seconds.
  Tear down plone.testing.zca.LayerCleanup in 0.004 seconds.
Total: 2 tests, 0 failures, 0 errors in 20.510 seconds.
```

Daneben werden noch Log-Dateien erstellt in `parts/test/`, z.B. `robot_log.html`:



REPORT

Robot Test Test Log

Generated
20130321 17:38:51 GMT +02:00
2 hours 10 minutes ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:00	
All Tests	1	1	0	00:00:00	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
start	1	1	0	00:00:00	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Robot Test	1	1	0	00:00:03	

Test Execution Log

TEST SUITE: **Robot Test** Expand All
Full Name: Robot Test
Source: /Users/veit/projects/vsc/vs_buildout/src/vs.registration/src/vs/registration/tests/robot_test.txt
Start / End / Elapsed: 20130321 17:38:48.913 / 20130321 17:38:51.658 / 00:00:02.745
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

SETUP: **Start browser** Expand All
Start / End / Elapsed: 20130321 17:38:48.982 / 20130321 17:38:51.489 / 00:00:02.507

KEYWORD: **Selenium2Library.Open Browser** http://localhost:55001/plone/, browser=${BROWSER} Expand All
Documentation: Opens a new browser instance to given URL.
Start / End / Elapsed: 20130321 17:38:48.982 / 20130321 17:38:51.486 / 00:00:02.504
17:38:48.982 INFO Opening browser 'firefox' to base url 'http://localhost:55001/plone/'

TEARDOWN: **Selenium2Library.Close All Browsers** Expand All

TEST CASE: **Plone site** Expand All

Siehe auch:

- Asko Soukka: Getting started with Robot Framework and plone.app.testing
- plone.act documentation
- Asko Soukka: Meet the Robot family (for Plone developers)
- Writing Robot Framework tests for Plone
- Beispiele:
  - plone.act Robot-Framework-Ressourcen zum Testen von Plone
  - plone.app.toolbar
  - plone.app.deco

  - plone.app.collection
- Selenium2Library keywords
- Robot Framework built-in keywords

Sphinx-Integration

Sphinx- und ReadTheDocs-Unterstützung für das Robot-Framework.

sphinxcontrib-robotdoc ist eine Erweiterung von Sphinx, zur Übernahme der Robot-Framework-Tests.

Dabei werden zwei neue Docutils- Direktiven eingeführt: `robot_tests` und `robot_keywords`. Beide Direktiven erlauben die folgenden Angaben:

- Filter in Form regulärer Ausdrücke
- Pfadangabe zu den Robot-Framework-Testdaten und -Ressourcen.
- Auswahl der Tests in Form einer Komma-separierten Tag-Liste

Beispiele

Die folgenden Beispiele können z.B. eingebunden werden in `vs_buildout/src/vs.registration/docs/index.rst`.

- Einbinden aller Tests einer Testsuite:

```
.. robot_tests::
   :source: ../src/vs/registration/tests/my_suite.txt
```

- Alles Tests einer Testsuite, die mit `Log` beginnen:

```
.. robot_tests:: Log.*
   :source: ../src/vs/registration/tests/my_suite.txt
```

- Einbinden aller Tests, die mit `login` oder `logout` getaggt sind:

```
.. robot_tests::
   :source: ../src/vs/registration/tests/my_suite.txt
   :tags: login, logout
```

- Einbinden aller *user keywords* eines Tests oder eine Ressource:

```
.. robot_keywords::
   :source: ../src/my_package/tests/acceptance/my_suite.txt
```

- Einbinden aller *user keywords*, die mit `Log` beginnen:

```
.. robot_keywords:: Log.*
   :source: ../src/my_package/tests/acceptance/my_suite.txt
```

ReadTheDocs-Unterstützung

ReadTheDocs unterstützt eigene Sphinx-Plugins:

1. Zunächst wird das Plugin in der Sphinx-Konfigurationsdatei (`conf.py`) in die Liste der *èxtensions*` eingetragen:

```
extensions = ['sphinxcontrib_robotdoc']
```

2. Das Plugin sollte auf PyPI veröffentlicht worden sein, siehe sphinxcontrib-robotdoc.
3. Desweiteren soll das ReadTheDocs-Project mit `virtualenv` erstellt werden:

```
Use virtualenv
[x]  Install your project inside a virtualenv using setup.py install
```

4. Es muss eine pip requirements-Datei geben, das das Sphinx- Plugin (und ggf. die mindestens erforderliche Version) enthält:

```
sphinxcontrib-robotdoc>=0.3.4
```

5. Ggf. kann die `requirements`-Datei nur für ReadTheDocs bereitgestellt werden indem sie in einem Unterverzeichnis erstellt wird, z.B. in `./docs/requirements.txt`.
6. Schließlich geben Sie im ReadTheDocs-Dashboard den Pfad zu Ihrer `requirements`-Datei an:

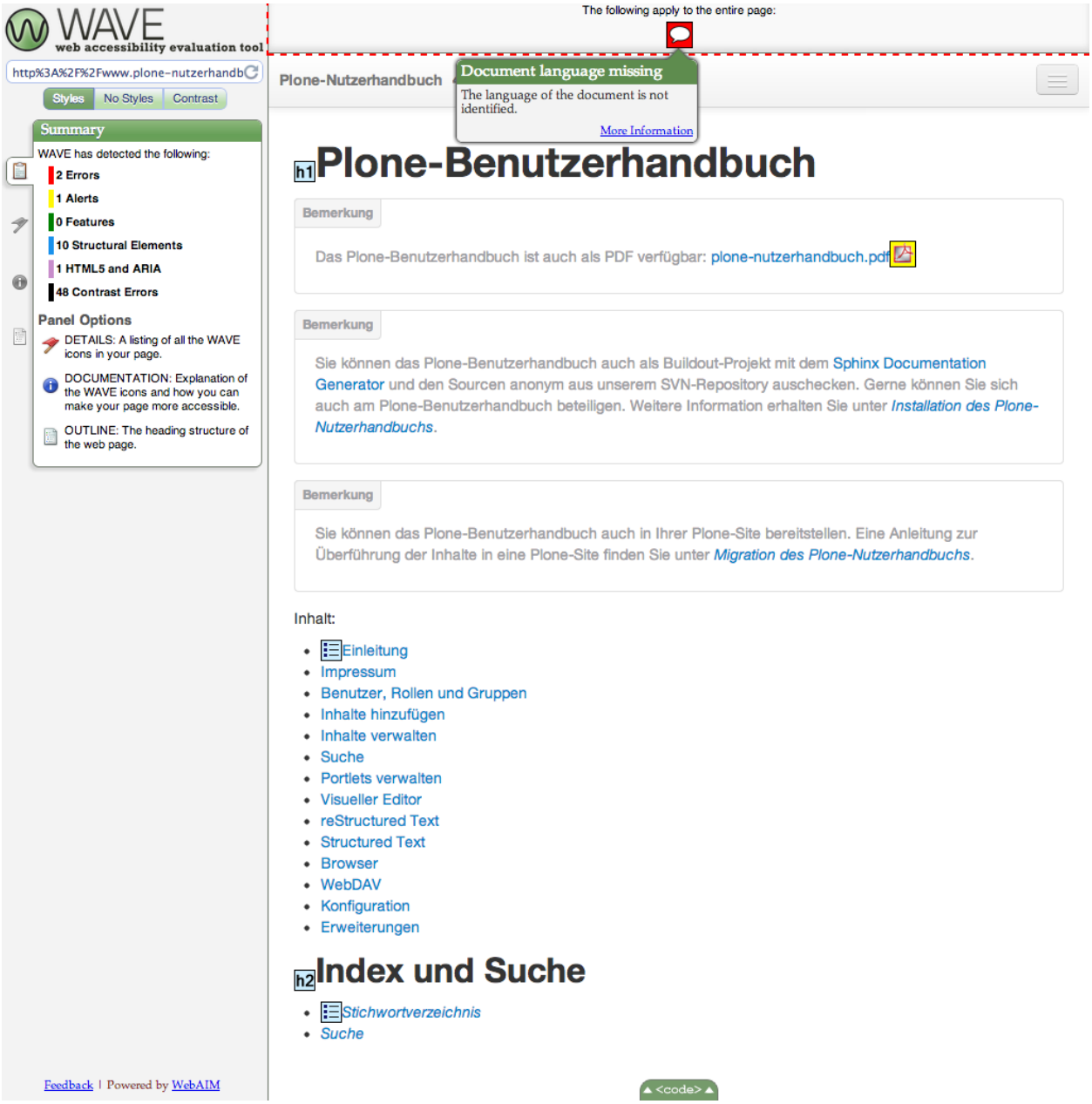
```
Requirements file:
docs/requirements.txt
```

Weitere Informationen

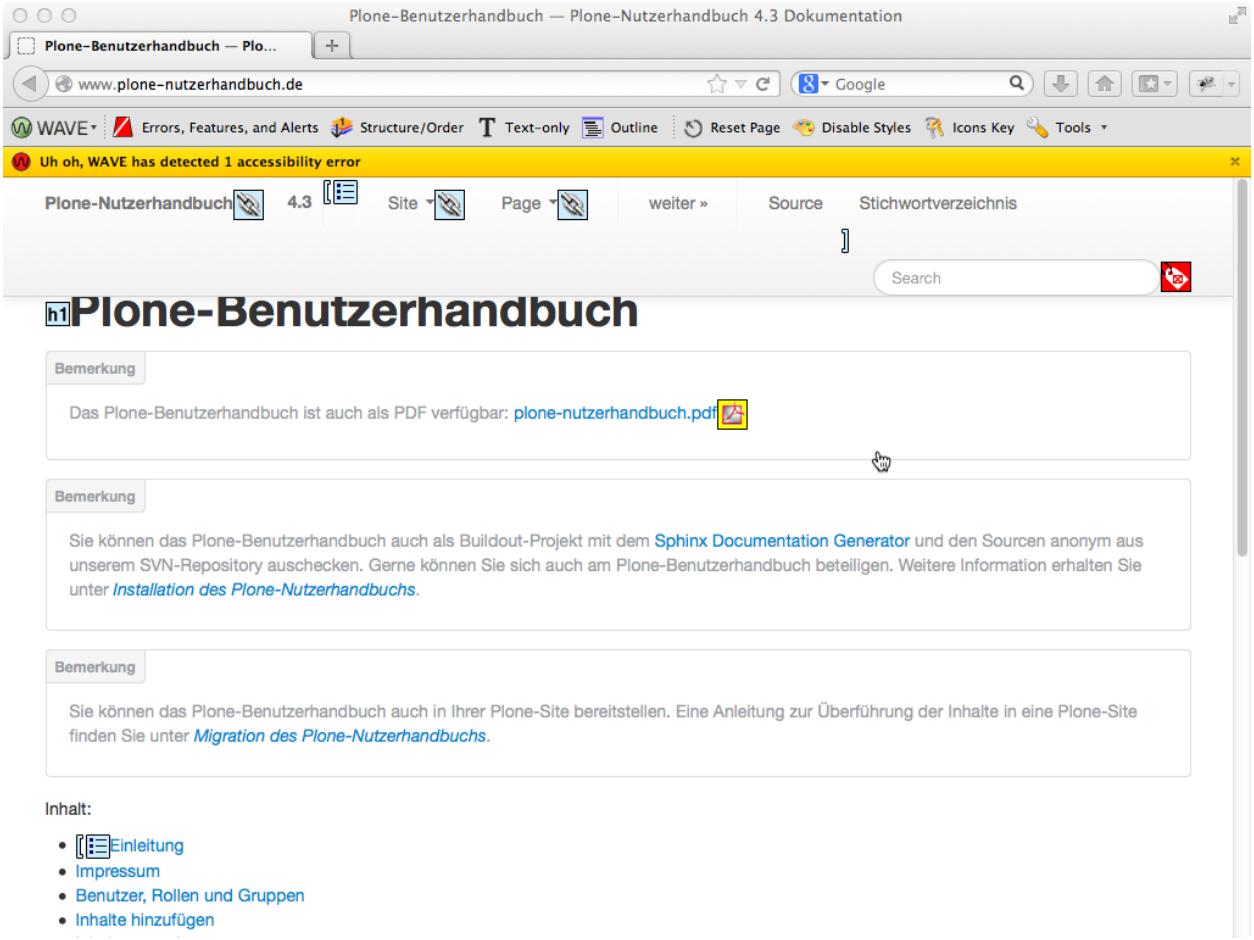
- Asko Soukka: Embedding Robot Framework tests and keywords into Sphinx documentation

iAccessibility-Analyse

WAVE ist ein Werkzeug zum Analysieren der Barrierefreiheit:



Die WAVE Toolbar ist eine Offline-Version dieses Dienstes als Firefox-Plugin:



WAVELibrary

robotframework-wavelibrary ist eine Bibliothek für das Robot-Framework, die die WAVE-Analyse Robot-Framework-Test durchführt.

Installation

In die Buildout-Konfiguration wird der Abschnitt `pybot` mit dem Python-Egg `robotframework-wavelibrary` hinzugefügt:

```
[buildout]
parts =
    ...
    pybot

[pybot]
recipe = zc.recipe.egg
eggs =
     robotframework
     robotframework-wavelibrary
```

Anschließend wird die Bibliothek installiert mit:

```
$ python bootstrap.py
$ ./bin/buildout
```

Accessibility-Tests

Hierfür können wir z.B. eine Datei `plone.robot` mit folgendem Inhalt schreiben:

```
*** Settings ***

Library  WAVELibrary

Resource  plone/app/robotframework/server.robot

Suite Setup  Setup
Suite Teardown  Teardown

*** Variables ***

${START_URL}  about:

*** Keywords ***

 Setup
     Setup Plone site  plone.app.robotframework.testing.AUTOLOGIN_ROBOT_TESTING
     Import library  Remote  ${PLONE_URL}/RobotRemote
     Enable autologin as  Site Administrator
     Set autologin username  test-user-1

 Teardown
     Teardown Plone Site

 *** Test Cases ***

 Test new page form tabs
     [Template]  Check new page tabs for accessibility errors
     default
     categorization
     dates
     creators
     settings

 *** Keywords ***

 Check new page tabs for accessibility errors
     [Arguments]  ${fieldset}
     Go to  ${PLONE_URL}/createObject?type_name=Document
     ${location} =  Get location
     Go to  ${PLONE_URL}
     Go to  ${location}#fieldsetlegend-${fieldset}
     Check accessibility errors
```

Eine vollständige Liste der WAVE-Library-Keywords erhalten Sie unter Keywords. Daneben können auch weiterhin die robot-Keywords und die Selenium-Leywords verwendet werden.

Dieser Test kann nun aufgerufen werden mit:

```
$ ./bin/pybot plone.robot
```

Siehe auch:

- Stay accessible: Robot Framework library for WAVE Toolbar

Jenkins Continuous Integration Server

Mit einem Continuous Integration Server können periodisch Unit- und Integrationstests durchlaufen werden.

Der Jenkins Continuous Integration Server kann Sie zudem sofort informieren, wenn ein Test fehlschlägt. Dies wird vor allem dann bedeutsam, wenn Sie mit anderen an derselben Code-Basis arbeiten. Dabei informiert Sie Jenkins, welches Checkin fehlschlug mit Revisionsnummer, Cheickin-Nachricht und Autor.

Mit collective.xmltestreport kommt ein Test-Runner, dessen XML-Ausgabe von Jenkins zur Erstellung von Grafiken und Trends verwendet werden kann.

Jenkins-Installation

Bemerkung: Eine Anleitung zum Installieren von Jenkins finden Sie in Installing Jenkins. Falls jedoch keine dieser Möglichkeiten zutreffend sein sollte, können Sie den Jenkins Continous Integration Server auch einfach mit Buildout und dem Rezept jarn.jenkins installieren.

```
[buildout]

   [buildout]

   parts =
      jetty-download
      jenkins-download
      jenkins

   [jetty-download]
   recipe = hexagonit.recipe.download
   url =  http://ftp.halifax.rwth-aachen.de/eclipse//jetty/stable-9/dist/jetty-
↪distribution-9.2.2.v20140723.tar.gz
   strip-top-level-dir = true

   [jenkins-download]
   recipe = hexagonit.recipe.download
   url = http://mirrors.jenkins-ci.org/war/latest/jenkins.war
   download-only = true

   [jenkins]
   recipe = jarn.jenkins
   jetty-location = ${jetty-download:location}
   jenkins-location = ${jenkins-download:location}
   host = localhost
   port = 8070
```

Hiermit wird sowohl jetty als auch Jenkins heruntergeladen und ein ausführbare Jetty-Umgebung in `parts/jenkins` erstellt. Außerdem wird mit `bin/jenkins` ein Skript erstellt, mit dem sich der Jenkins-Server starten und Stoppen lässt.

Um die Installation zu testen können Sie einfach folgendes angeben:

```
$ ./bin/jenkins fg
```

Damit wird der Jetty-Server am Port `8070` gestartet. Die Jenkins-Instanz ist dann erreichbar unter `http://127.0.0.1:8070/jenkins/`.

Log-Datei Diese wird von Jenkins schreibt nach `var/jenkins/log` geschrieben.

Konfiguration Die Konfiguration einschließlich der auszuführenden Jobs wird von Jenkins nach `var/jenkins/data` geschrieben. Dabei entspricht der Verzeichnisname in `var/` dem Namen des Abschnitts, das das Rezept `jarn.jenkins` verwendet.

Erstellen eines Jobs

Verwenden Sie in Jenkins die *Free Style*-Vorlage um einen neuen Job zu erstellen.

1. Alte Builds verwerfen

   Wie lange sollen *Builds* aufbewahrt werden? Hiermit lässt sich der Festplattenverbrauch von Jenkins steuern. Jenkins bietet hierzu zwei Strategien an:

   Nach Alter Jenkins löscht Aufzeichnungen, sobald sie ein bestimmtes Alter erreichen, z.B. 7 Tage alt sind.

   Nach Anzahl Jenkins bewahrt nur die `N` neuesten *Builds* auf. Wenn ein neuer Build gestartet wird, löscht Jenkins den ältesten.

   Jenkins erlaubt darüberhinaus, dass einzelne *Builds* markiert werden mit*Dieses Protokoll für immer aufbewahren*, sodass wichtige *Builds* von der automatischen Löschung ausgeschlossen werden.

2. Erweiterte Projekteinstellungen

   Hier können Sie die Anzahl der Wiederholungen bei fehlgeschlagenen Checkouts angeben.

3. Source-Code-Management (SCM)

   Hier können Sie z.B. die URL Ihres Subversion-Repository, die Check-Out-Strategie und den Repository-Browser angeben.

4. Build-Auslöser

   Hier können Sie die Zeitpläne angeben, zu denen die Builds gestartet werden sollen.

5. Buildverfahren

   Als Buildverfahren wählen Sie ein Shell-Skript, das z.B. folgenden Inhalte haben kann:

   ```
   cd /home/veit/my_buildout
   ./bin/develop up
   ./bin/buildout
   ./bin/test --xml -s vs.registration
   ```

   `./bin/develop up` aktualisiert die Quellen von `mr.developer`.

   `./bin/test --xml -s vs.registration` Das Buildout hat einen `[test]`-Abschnitt, der folgendermaßen aussieht:

   ```
   [test]
   recipe = collective.xmltestreport
   eggs =
       vs.registration
   ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
extra-paths = ${zope2:location}/lib/python
defaults = ['--exit-with-status', '--auto-color', '--auto-progress']
```

Das Rezept `collective.xmltestreport` ist eine spezielle Version von `zc.recipe.testrunner` um Testreports im XML-Format zu schreiben wie es von JUnit/Ant verwendet wird. Dies erlaubt es Jenkins, die Testergebnisse zu analysieren.

6. Post-Build-Aktionen

   Veröffentliche JUnit-Testergebnisse Es sind reguläre Ausdrücke wie z.B. `parts/test-jenkins/testreports/*.xml` erlaubt. Das Ausgangsverzeichnis ist der Arbeitsbereich.

   Plot build data Mit dem Plot plugin können Sie sich Trends grafisch darstellen lassen.

   E-Mail-Benachrichtigung Hier können Sie eine Liste der Empfänger angeben, die bei jedem fehlgeschlagenen Build informiert werden sollen.

   Darüberhinaus können auch diejenigen informiert werden, die einen Build fehlschlagen ließen.

tox/detox

Auf virtualenv basierende Automatisierung von Tests.

tox kann verwendet werden

- zur Überprüfung, ob Ihr Paket mit verschiedenen Python-Versionen und Interpretern installiert werden kann
- zum Ausführen der Tests in verschiedenen Environments und mit verschiedenen Testwerkzeugen
- als Frontend für Continous Integration Server, das unnötige Wiederholungen vermeidet sowie CI- und Shell-basierte Tests verbindet.

detox erlaubt das verteilte Aufrufen von `tox`, sodass Tests parallel ausgeführt werden können. Die Optionen und die Konfigurationsmöglichkeiten entsprechen denen von `tox`.

Weitere Informationen erhalten Sie in der Dokumentation zum tox automation project. Dort ist auch die Verwendung zusammen mit dem Jenkins Integration Server beschrieben: Using Tox with the Jenkins Integration Server.

Travis CI

Travice CI ist ein hosted Continuous-Integration-Service.

Travis CI ist ein gehosteter Continous Integration Service, mit dem sich Plone-Zusatzprodukte einfach testen lassen.

Installation

1. Travis lässt sich mit Ruby Gems installieren:

   ```
   $ gem install travis
   ```

2. Damit unser Paket getestet werden kann, muss zunächst die `setup.py`- Datei dieses Produkts erweitert werden:

```
...
install_requires=[
    'setuptools',
    'Products.CMFPlone>=4.2',
    ],
extras_require={
    'test': ['plone.app.testing'],
    },
...
```

3. Aufsetzen von Travis

   Zunächst melden Sie sich einfach mit Ihrem Github-Account an: App Authorization.

4. Anschließend konfigurieren Sie Ihren Travis CI-Server mit einer `travis.yml` -Datei im Wurzelverzeichnis Ihres Repository:

```
---
language: python
python: '2.7'
install:
- mkdir -p buildout-cache/eggs
- mkdir -p buildout-cache/downloads
- python bootstrap.py -c travis.cfg
- ./bin/buildout -N -t 3 -c travis.cfg
script: ./bin/test
```

   Weitere Konfigurationsmöglichkeiten erhalten Sie in Travis CI-Konfiguration.

5. Die Datei `travis.cfg` sieht dann z.B. so aus:

```
[buildout]
extends =
    http://svn.plone.org/svn/collective/buildout/plonetest/test-4.x.cfg

parts = test

package-name = vs.registration
package-extras = [test]
#test-eggs = Pillow

allow-hosts +=
    code.google.com
    robotframework.googlecode.com

[environment]
ZSERVER_HOST = 0.0.0.0
ROBOT_ZOPE_HOST = 0.0.0.0

[test]
environment = environment

[test]
eggs =
    ${buildout:package-name} ${buildout:package-extras}
    ${buildout:test-eggs}
```

`test-eggs = Pillow` Diese Zeile sollte auskommentiert werden sofern PIL für die Tests benötigt wird.

6. Schließlich können Sie noch ein Status-Bild in Ihre `README.txt`-Datei einfügen:

```
.. image:: https://secure.travis-ci.org/collective/vs.registration.png
:target: http://travis-ci.org/collective/vs.registration
```

Siehe auch:

- Héctor Verlarde: Integrating Travis CI with your Plone add-ons hosted on GitHub
- Python Testing Tools Taxonomy

Apps, Clients and Tools

- Travis CI Apps, Clients and Tools
  - Travis CLI
  - TravisPy

Travis CI-Konfiguration

Travice CI bietet umfangreiche Konfigurationsmöglichkeiten.

Die Konfiguration des Travis CI-Server erfolgt in der `travis.yml` -Datei im Wurzelverzeichnis Ihres Repository.

Hier stehen Ihnen u.a. zusätzliche Konfigurationsmöglichkeiten zur Verfügung, u.a.:

`language` gibt nicht nur die Sprache an, sondern ggf. auch die Sprachversionen, mit denen getestet werden soll:

```
language: python
python:
  - 2.6
  - 2.7
```

`env` Umgebungen, in der die Tests ausgeführt werden sollen, also z.B.:

```
env:
  - PLONE_VERSION=4.2
  - PLONE_VERSION=4.3
```

Ein vollständiges Beispiel hierfür finden Sie in https://github.com/plone/plone.api/blob/master/.travis.yml.

`services` Dienste, die vor dem Testen bereitstehen sollen, z.B.:

```
- riak     # will start riak
- rabbitmq # will start rabbitmq-server
- memcache # will start memcached
```

Eine vollständige Liste der zur Verfügung stehenden Services finden Sie in Configure Your Projects to Use Services in Tests.

`before_install` Aufruf, der vor der Installation ausgeführt werden soll, z.B. um ein GUI *headless* testen zu können:

```
- "export DISPLAY=:99.0"
- "sh -e /etc/init.d/xvfb start"
```

Weitere Informationen hierzu erhalten Sie in GUI & Headless browser testing on travis-ci.org.

install Installation der Testumgebung, z.B.:

```
- unzip Sauce-Connect-latest.zip
- java -jar Sauce-Connect.jar $SAUCE_USERNAME $SAUCE_ACCESS_KEY -i $TRAVIS_JOB_ID␣
↪-f CONNECTED &
- JAVA_PID=$!
```

before_script Aufruf, der unmittelbar vor einem Test aufgerufen werden soll, z.B.:

```
bash -c "while [ ! -f CONNECTED ]; do sleep 2; done"
```

test Testaufruf, z.B.:

```
./bin/test
```

after_script Aufruf, der unmittelbar nach dem Test durchgeführt wird, z.B.:

```
kill $JAVA_PID
```

branches Blacklist und Whitelist von zu testenden Branches, z.B.:

```
# blacklist
branches:
  except:
    - legacy
    - experimental

# whitelist
branches:
  only:
    - master
    - stable
```

notifications Benachrichtigungen, , z.B.:

```
notifications:
  irc:
    email:
      - "kontakt@veit-schiele.de"
    channels:
      - "irc.freenode.org#sprint"
    on_success: never
    on_failure: always
    template:
      - "%{repository}#%{build_number} (%{branch} - %{commit} : %{author}): %
↪{message}"
      - "Change view : %{compare_url}"
      - "Build details : %{build_url}"
```

Skip build

Falls nach einem `push` kein neuer Travis-Build erstellt werden soll, z.B. bei Änderungen in der Dokumentation, so kann dies einfach beim Commit angegeben werden:

```
$ git commit -m 'Typo in README.rst [ci skip]'
```

Travis CI Sauce Labs-Support

Die Nutzung von Sauce Labs zusammen mit dem Robot- Framework ist ähnlich einem eigenen Selenium-Grid. Vor allem erfordert es, dass die Browser Passwörter eingeben können. Dies kann mit einigen wenigen Variablen in `vs_buildout/src/vs.registration/src/vs/registration/tests/robot_test.txt` konfiguriert werden:

```
*** Settings ***

Library  Selenium2Library  timeout=10  implicit_wait=0.5

Suite Setup  Start browser
Suite Teardown  Close All Browsers

*** Variables ***

${ZOPE_HOST} =  localhost
${ZOPE_PORT} =  55001
${ZOPE_URL} =  http://${ZOPE_HOST}:${ZOPE_PORT}

${PLONE_SITE_ID} =  plone
${PLONE_URL} =  ${ZOPE_URL}/${PLONE_SITE_ID}

${BROWSER} =  Firefox
${REMOTE_URL} =
${DESIRED_CAPABILITIES} =  platform:Linux
${BUILD_NUMBER} =  manual

*** Test Cases ***

Plone site
    [Tags]  start
    Go to  ${PLONE_URL}
    Page should contain  Plone site

*** Keywords ***

Start browser
    ${BUILD_INFO} =  Set variable
    ...           build:${BUILD_NUMBER},name:${SUITE_NAME} | ${TEST_NAME}
    Open browser  ${PLONE_URL}  ${BROWSER}
    ...           remote_url=${REMOTE_URL}
    ...           desired_capabilities=${DESIRED_CAPABILITIES},${BUILD_INFO}
```

Die Variablen bedeuten im Einzelnen:

`ZOPE_HOST` Angabe for den Host des ZServer.

> Dar Standardwert ist `localhost`. For Tests mit dem Internet Explorer ist jedoch die Angabe `0.0.0.0` erforderlich.

ZOPE_PORT Angabe des Ports, an dem der ZServer lauscht.

Der Standardwert ist `55001`.

ZOPE_URL Root-Variable für die URL der Zope-Anwendung.

PLONE_SITE_ID ID der Plone-Site.

PLONE_URL URL der Plone-Site.

BROWSER Browser, mt dem der Test durchgeführt werden soll.

REMOTE_URL URL des zu verwendenden Selenium-Hubs.

DESIRED_CAPABILITIES spezifiziert verschiedene Parameter des Selenium-Hubs, z.B. die Browser- Version.

BUILD_NUMBER Travis-CI-Build auf Sauce Labs.

Nun wird eine `.travis.yml`-Datei erstellt um Travis-CI mitzuteilen, welches Envirinment verwendet und welche Tests ausgeführt werden sollen:

```
---
language: python
python: '2.7'
install:
- mkdir -p buildout-cache/eggs
- mkdir -p buildout-cache/downloads
- python bootstrap.py -c travis.cfg
- ./bin/buildout -N -t 3 -c travis.cfg
- curl -O http://saucelabs.com/downloads/Sauce-Connect-latest.zip
- unzip Sauce-Connect-latest.zip
- java -jar Sauce-Connect.jar $SAUCE_USERNAME $SAUCE_ACCESS_KEY -i $TRAVIS_JOB_ID -f␣
↪CONNECTED &
- JAVA_PID=$!
before_script:
- bash -c "while [ ! -f CONNECTED ]; do sleep 2; done"
script: ./bin/test
after_script:
- kill $JAVA_PID
env:
  global:
  - ROBOT_BUILD_NUMBER=travis-$TRAVIS_BUILD_NUMBER
  -  ROBOT_REMOTE_URL=http://$SAUCE_USERNAME:$SAUCE_ACCESS_KEY@ondemand.saucelabs.
↪com:80/wd/hub
  matrix:
  - ROBOT_BROWSER=firefox ROBOT_DESIRED_CAPABILITIES=tunnel-identifier:$TRAVIS_JOB_ID
  - ROBOT_BROWSER=chrome ROBOT_DESIRED_CAPABILITIES=tunnel-identifier:$TRAVIS_JOB_ID
  - ROBOT_BROWSER=internetexplorer ROBOT_DESIRED_CAPABILITIES=tunnel-identifier:
↪$TRAVIS_JOB_ID
```

SAUCE_USERNAME und SAUCE_ACCESS_KEY Nutzername und Passwort verschlüsselt als Umgebungsvariable.

`travis encrypt` schreibt die verschlüsselten Werte direkt in die `.travis.yml`-Datei:

```
$ travis encrypt SAUCE_USERNAME=myusername -r mygithubname/example.product --add␣
↪env.global
$ travis encrypt SAUCE_ACCESS_KEY=myaccesskey -r mygithubname/example.product --
↪add env.global
```

matrix Aktuell erlaubt Sauce Labs drei gleichzeitige Verbindungen für Open-Source- Projekte, z.B. für drei verschiedene Browser.

Achten Sie bei Open-Source-Projekten darauf, dass Sie nicht Ihren privaten Zugang nutzen sondern denjenigen des Projekts. Hierfür ist die öffentliche URL des Repository erforderlich.

Schließlich sollten die Travis-CI-Tests für Ihr Produkt auf Travis-CI.org oder GitHub eingerichtet werden.

Siehe auch:

- Asko Soukka: Cross-browser test your Plone add-on with Robot Framework, Travis-CI and Sauce Labs

Generic Setup Tool

Das Generic Setup Tool vereinfacht die Vorkonfigurierung einer Site.

Jede konfigurierbare Komponente stellt Handler zum Im- und Export von Profilen bereit.

Profile sind Konfigurationsdateien für bestimmte Komponenten einer Website. Diese können z.B. Rollen, Berechtigungen, Skin-Layer und vieles mehr festlegen. Dabei werden im wesentlichen zwei Arten von Profilen unterschieden:

base profile Profil für die Basiskonfiguration einer Site. `Products.CMFPlone` bringt ein solches Profil mit, das die Standardkonfiguration einer Plone-Site enthält.

extension profile Profil, das auf einem *base profile* aufbaut und an einigen Stellen die Standardkonfiguration ändert und neue Im- und Export-Schritte bereitstellen kann.

Registrieren eines Profils

GenericSetup-Profile können einfach mit ZCML registriert werden, z.B. in der `configure.zcml`-Datei von `vs.theme`:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
    i18n_domain="vs.theme">

  <genericsetup:registerProfile
      name="default"
      title="Theme for the Websites of Veit Schiele Communications"
      directory="profiles/default"
      description="Default profile for vs.theme"
      provides="Products.GenericSetup.interfaces.EXTENSION"
      />

</configure>
```

`name` Bestandteil der ID des GenericSetup-Profils. Die vollständige ID lautet `profile-<package_name>:<profile_name>`, in unserem Fall also `profile-vs.theme:default`.

`title` Der Titel des Profils, der Ihnen im Generic Setup Tool beim Import angezeigt wird und im Quickinstaller beim Aktivieren eines Pakets.

`directory` Relative Pfadangabe zum Verzeichnis mit den Profilinformationen. Meist entspricht der Verzeichnisname dem Profilnamen.

`description` Die Beschreibung des Profils sollte eine kurze Zusammenfassung für die Verwendung des Profils geben.

`provides` Die Art des Profils, also `EXTENSION` oder `BASE`.

Tipps

1. Häufig ist die einfachste Möglichkeit ein Profil zu schreiben diejenige, in einer Site Änderungen an der Konfiguration vorzunehmen und anschließend die Profile derjenigen Tools zu exportieren, deren Konfiguration geändert wurde.
2. Anschließend sollte dieses Profil jedoch nicht unmittelbar übernommen werden sondern nur diejenigen Teile, die auch tatsächlich geändert wurden.
3. Üblicherweise ersetzen die Werte von `EXTENSION`-Profilen die bereits bestehenden Werte. Mit `purge="False"` kann dieses Verhalten jedoch geändert werden, z.B.:

```
<?xml version="1.0"?>
<object name="portal_properties">
    <object name="navtree_properties">
        <property name="metaTypesNotToList" type="lines" purge="False">
            <element value="Registration"/>
        </property>
    </object>
</object>
```

Hierdurch wird `Registration` der Liste der Artikeltypen hinzugefügt. Ohne `purge="False"` würde nur der `Registration`-Artikeltyp nicht in der Navigation angezeigt werden.

Metadaten

Das Profil in `metadata.xml` kann z.B. so aussehen:

```
<?xml version="1.0"?>
<metadata>
  <description>Policy for the Website of Veit Schiele Communications</description>
  <version>1.0dev $LastChangedRevision$ </version>
  <dependencies>
    <dependency>profile-vs.theme:default</dependency>
    <dependency>profile-Products.LinguaPlone:LinguaPlone</dependency>
  </dependencies>
</metadata>
```

`description` Kurze Erläuterung des Profils

`version` Die Versionsnummer des Profils

Diese wird auch verwendet um Upgrade-Schritte durchzuführen. Upgrades können immer nur zwischen definierten Versionsnummern durchgeführt werden.

`dependencies` Voraussetzungen für dieses Profil. Profile, die hier genannt werden, werden beim Import zuerst ausgeführt.

Siehe auch:

- Plone Developer Manual: Generic Setup

Content rules

Content rules lassen sich in neueren Plone-Versionen mit einem Generic Setup-Profil erstellen. Wie ein solches Profil aussehen kann, sehen Sie in der Datei `contentrules.xml`.

In diesem Profil ist eine Regel definiert:

```
<rule
    name="rule-1"
    title="Mail notification"
    description=""
    enabled="True"
    event="Products.CMFCore.interfaces.IActionSucceededEvent"
    stop-after="False">
```

`name` Name (ID) der Regel

`title` Titel der Regel

`description` Beschreibung der Regel

`enabled` Ist die Regel aktiv?

`event` Welches Ereignis löst die Regel aus.

`stop-after` Sollen weitere Regeln nach dieser Regel ausgeführt werden?

Anschließend folgen drei weitere Abschnitte:

1. Eine Liste von Bedingungen (`conditions`) für diese Regel
2. Eine Liste von Aktionen (`actions`) für diese Regel
3. Die Zuweisung von Regeln einem bestimmten Kontext

Bedingungen (`conditions`)

`plone.app.contentrules` kommt mit den folgenden Bedingungen:

Artikeltyp (`plone.conditions.PortalType`) Mit dieser Bedingung legen Sie fest, dass eine Aktion nur bei bestimmten Artikeltypen ausgeführt wird.

Dateiendung (`plone.conditions.FileExtension`) Mit dieser Bedingung können Sie festlegen, dass eine Aktion nur bei bestimmten Dateiendungen ausgeführt wird.

Stadien (`plone.conditions.WorkflowState`) Mit dieser Bedingung legen Sie fest, dass eine Aktion nur bei Artikeln angewendet wird, die sich in einem bestimmten Status befinden.

Übergänge (`plone.conditions.WorkflowTransition`) Mit dieser Bedingung legen Sie fest, dass eine Aktion nur bei bestimmten Workflow-Übergängen (`transitions`) angewendet wird.

Gruppe (`plone.conditions.Group`) Mit dieser Bedingung legen Sie fest, dass eine Aktion nur ausgeführt wird, wenn der aktuelle Benutzer Mitglieder in einer bestimmten Gruppe ist.

Rolle (`plone.conditions.Role`) Mit dieser Bedingung legen Sie fest, dass eine Aktion nur ausgeführt wird, wenn der Benutzer eine bestimmte Rolle hat.

Aktionen (`actions`)

Name des Protokolls (`plone.actions.Logger`) protokolliert ein bestimmtes Ereignis

Nachricht (`plone.actions.Notify`) gibt eine Nachricht im Browser des Nutzers aus.

Kopieren (`plone.actions.Copy`) kopiert den Artikel in einen bestimmten Ordner.

Verschieben (`plone.actions.Move`) verschiebt den Artikel in einen bestimmten Ordner.

Löschen (`plone.actions.Delete`) löscht den Artikel.

Statusänderung (`plone.actions.Workflow`) ändert den Status des Artikels.

Mail versenden (`plone.actions.Mail`) Versenden einer E-Mail unter Angabe von Betreff, Absender, Empfänger und Nachrichtentext.

Dabei können Sie für diese Felder folgende Variablen verwenden:

`${absolute_url}` URL des Artikels

`${user_email}` E-Mail-Adresse des Nutzers

`${user_fullname}` Name des Nutzers

`${user_id}` Id des Nutzers

`${contributors}` Beteiligte

`${created}` Erstellungsdatum

`${creators}` Ersteller

`${description}` Beschreibung

`${effective}` Veröffentlichungsdatum

`${expires}` Ablaufdatum

`${format}` Format

`${identifier}` Identifier (URI)

`${keywords}` Betreff

`${language}` Sprache

`${modified}` Änderungsdatum

`${rights}` Veröffentlichungsrechte

`${subject}` Betreff

`${title}` Titel

`${type}` Artikeltyp

`${manager_emails}` E-Mails an Verwalter

`${member_emails}` E-Mail an Mitglieder

`${owner_emails}` E-Mail an Eigentümer

`${reviewer_emails}` E-Mail an Redakteure

`${change_authorid}` Geänderter Name des Autors

`${change_comment}` Kommentar

`${change_title}` Geänderter Titel

${change_type} Geänderter Artikeltyp

${review_state} Geänderter Status

Zuweisung (assignment)

```
<assignment
    location="/news"
    name="rule-1"
    enabled="True"
    bubbles="False"
    insert-before="*"
/>
```

location (erforderlich) Der Ort,an dem die Regel greifen soll. Üblicherweise sind dies in Plone Ordner, die mit dem `IRuleAssignable`-Interface markiert werden.

Hier wird eine Pfadangabe relativ zu *portal root* erwartet.

name (erforderlich) Der Name der Regel, die zugewiesen werden soll.

enabled (optional) Soll die Regel an dem angegebenen Ort aktiv sein?

bubbles (optional) Soll die Regel auch in Unterordnern zugewiesen werden?

Der Standardwert ist `False` wodurch die Regel nicht auf passende Events in Unterordnern angewendet wird.

insert-before (optional) Dies kann verwendet werden um die Reihenfolge, in der die Zuweisungen für einen bestimmten Ort ausgeführt werden sollen, zu beeinflussen.

Wird hier nichts angegeben, wird die Regel nach allen anderen Regeln an diesem Ort ausgeführt.

`*` bewirkt, dass die Regel als erste ausgeführt wird.

Siehe auch:

- Content rules
- Creating Content Rule Conditions and Actions
- Time based workflow transitions

Repositorytool

Ab Plone 4.1 lässt sich das Repositorytool über ein Generic-Setup-Profil konfigurieren.

Die `repositorytool.xml`-Datei kann z.B. so aussehen:

```
<?xml version="1.0"?>
<repositorytool>
    <policymap>
        <type name="MyType">
            <policy name="at_edit_autoversion"/>
            <policy name="version_on_revert"/>
        </type>
        <type name="AnotherType">
            <policy name="at_edit_autoversion"/>
            <policy name="version_on_revert"/>
        </type>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    </policymap>
</repositorytool>
```

Soll das Produkt auch in Plone < 4.1 funktionieren, sollte in der `setuphandlers.py`-Datei eine Bedingung angegeben werden. Sehen Sie hierzu Generic-Setup-Profil für die Versionierung.

Toolset

Mit der `toolset.xml`-Datei lassen sich *Portal Tools* instantiieren oder entfernen, z.B.:

```
<?xml version="1.0"?>
<tool-setup>
  <required tool_id="portal_foo" class="dotted.path.to.Foo" />
  <forbidden tool_id="portal_bar" />
</tool-setup>
```

In diesem Beispiel wird `portal_foo` instantiiert mit der Klasse `Foo`. Zudem wird das `portal_bar`-Tool entfernt sofern vorhanden.

Bemerkung: Die `toolset.xml`-Datei kann nicht nur im Paket verwendet werden, das das jeweilige Tool bereitstellt sondern ist vor allem auch für Policy-Pakete gedacht.

Dexterity

Referenz zu den Dexterity Feldtypen und -eigenschaften, Widgets, Behaviors, Form-Directives sowie Eigenschaften und Methoden der Inhaltsobjekte.

Felder

Die gebräuchlichsten Feldtypen und -eigenschaften, die in Dexterity verwendet werden.

Eine Anleitung zum Erstellen eines Schemas erhalten Sie in *Schema Interfaces*.

Feldeigenschaften

Interface	Eigenschaft	Typ	Beschreibung
IField	title	unicode	Der Titel des Feldes, der im Widget verwendet wird.
	description	unicode	Die Beschreibung des Feldes, die im Widget verwendet wird.
	required	bool	Es wird überprüft, ob das Feld eine Angabe enthält. Der Standarwert ist `True`.
	readonly	bool	Ist der Wert `True`, so kann das Feld nur gelesen werden. Der Standarwert ist `False`.
	default		Der Standardwert eines Feldes. Dieser Wert kann ggf. auch als Fallback verwendet werden, falls keine Angabe gemacht wurde. Dieser Wert muss eine valide Angabe für dieses Feld sein. Der Standardwert ist `None`.
	missing_value		Ein wert, der verdeutlicht, dass dieses Feld nicht ausgefüllt wurde. Dieser Wert wird bei der Validierung des Formulars verwendet. Der Standarwert ist `None`. Für Listen und Tuples kann es gelegentlich nützlich sein, eine leere Liste oder ein leeres Tuple zu setzen.
IMinMaxLen	min_length	int	Die minimale Länge der Eingabe. Wird für `string`-Felder verwendet. Der Standardwert ist `0`.
	max_length	int	Die maximale Länge der Eingabe. Wird für `string`-Felder verwendet. Der Standardwert ist `0`.
IMinMax	min		Der minimal erlaubte Wert. Dies muss ein valider Wert für dieses Feld sein. Der Standardwert ist `None`.
	max		Der maximal erlaubte Wert. Dies muss ein valider Wert für dieses Feld sein. Der Standardwert ist `None`.
ICollection	value_type		Erlaubte Werte einer Liste, eines Tuples oder einer anderen Sammlung. Muss für jedes `collection`-Feld gesetzt werden. Häufig wird als Wert `Choice` angegeben um ein Multi-Selection-Feld mit einem Vokabular zu erstellen.
	unique	bool	Ob die Werte in der Kollektion eindeutig sein müssen oder nicht. Wird meist nicht direkt gesetzt, sondern es wird ein `Set` oder ein `Frozenset` verwendet um die Eindeutigkeit zu garantieren.
IDict	key_type		Beschreibt die erlaubten Schlüssel in einem Dictionary. Ähnlich dem `value_type` in Kollektionen. Muss gesetzt werden.
	value_type		Beschreibt die erlaubten Werte in einem Dictionary. Ähnlich `value_type` in Kollektionen. Muss gesetzt werden.
IObject	schema	Interface	Ein Interface, das von jedem Objekt, das in diesem Feld gespeichert wird bereitgestellt werden muss.
IRichText	default_mime_type	string	Standard-MIME-Typ für den Eingabetext eines Rich Text-Felds. Der Standard ist `text/html`.
	output_mime_type	string	Standard-MIME-Typ für den transformierten Text eines Rich Text-Felds. Der Standard ist `text/x-html-safe`.
	allowed_mime_types	tuple	Eine Liste aller erlaubten MIME-Typen für die Eingabe. Der Standardwert ist `None`, wobei den die Einstellungen für die gesamte Website übernommen werden.

Feldtypen

Die folgende Tabelle listet die am häufigsten verwendeten Feltypen auf, sortiert nach dem Modul, von dem sie importiert werden können.

Felder in `zope.schema`

Felder in `plone.namedfile.field`

Weitere Informationen erhalten Sie unter plone.namedfile und plone.formwidget.namedfile.

Name	Typ	Beschreibung	Typ
`NamedFile`	`NamedFile`	Eine hochzuladende Binärdatei. Üblicherweise wird das Widget aus `plone.formwidget.namedfile` verwendet.	`IField`
`NamedImage`	`NamedImage`	Ein hochzuladendeis Bild. Üblicherweise wird das Widget aus `plone.formwidget.namedfile` verwendet.	`IField`
`NamedBlobFile`	`NamedBlobFile`	Eine hochzuladende Binärdatei, die als ZODB-BLOB gespeichert wird. Üblicherweise wird das Widget aus `plone.formwidget.namedfile` verwendet.	`IField`
`NamedBlobImage`	`NamedBlobImage`	Ein hochzuladendes Bild, das als ZODB-BLOB gespeichert wird. Üblicherweise wird das Widget aus `plone.formwidget.namedfile` verwendet.	`IField`

Felder in `z3c.relationfield.schema`

Weitere Informationen erhalten Sie unter z3c.relationfield.

Name	Typ	Beschreibung	Typ
`Relation`	`RelationValue`	Speichert den Wert einer einzelnen Relation.	`IField`
`RelationList`	`list`	List-Feld für `RelationValue`.	Siehe `List`.
`RelationChoice`	`RelationValue`	Choice-Feld für `RelationValue`.	Siehe `Choice`

Felder in `plone.app.textfield`

Weitere Informationen erhalten Sie unter plone.app.textfield.

Name	Typ	Beschreibung	Typ
`RichText`	`RichTextValue`	Speichert einen `RichTextValue`, der den Raw-Text, den MIME-Typ und eine gecachte Version des in den Standard-MIME- Typ konvertierten Textes enthält.	`IField`, `IRichText`

Widgets

Standard-Widgets und Third-Party-Widgets.

Die gebräuchlichsten Widgets werden von `z3c.form` bereitgestellt. Weitere Informationen zu diesen Widgets erhalten Sie in der z3c.form-Dokumentation.

Die Tabelle führt die häufig verwendeten Widgets auf:

Widget	Importiert von	Feld	Beschreibung
`WysiwygFieldWidget`	`plone.app.z3cform.wysiwyg`	`Text`	Hiermit erhalten Sie den Standard-WYSIWYG-Editor von Plone für dieses Feld.
`RichTextWidget`	`plone.app.textfield.widget`	`RichText`	Dieses Feld erlaubt neben dem Standard-WYSIWYG-Editor auch Text-basiertes Markup wie bei reStructured-Text.
`AutocompleteFieldWidget`	`plone.formwidget.autocomplete`	`Choice`	Autocpmplete-Widget, das auf jQuery-Autocomplete basiert. Erfordert ein Choice-Feld mit Angabe von `source`. Siehe Vokabularien.
`AutocompleteMultiFieldWidget`	`plone.formwidget.autocomplete`	`Collection`	Multi-Select-Version für Tuple, Listen,Sets oder Frozensets mit dem WErtetyp `Choice`.
`ContentTreeFieldWidget`	`plone.formwidget.contenttree`	`RelationChoice`	Content-Browser. Erfordert eine Quelle, die nach Objekten angefragt werden kann.
`MultiContentTreeFieldWidget`	`plone.formwidget.contenttree`	`RelationList`	Content-Browser. Erfordert eine Quelle, die nach Objekten angefragt werden kann.
`NamedFileFieldWidget`	`plone.formwidget.namedfile`	`NamedFile`	Ein Widget zum Hochladen von Dateien.
`NamedImageFieldWidget`	`plone.formwidget.namedimage`	`NamedImage`	Ein Widget zum Hochladen von Bildern.
`TextLinesFieldWidget`	`z3c.form.textlines`	*Collection`*	Listen-Eintrag für List, Tuple, Set oder Frozenset-Felder. Erfordert als Wertetyp `TextLine` oder `ASCIILine`.
`SingleCheckBoxFieldWidget`	`z3c.form.browser.checkbox`	`Bool`	Checkbox für wahr/falsch.
`CheckBoxFieldWidget`	`z3c.form.browser.checkbox`	`Collection`	Ein Set von Checkboxen. Wird verwendet für Set- oder Frozenset-Felder mit `Choice` als WErtetyp und einem Vokabular.

Behaviors

Eine Liste der gebräuchlichsten Dexterity-Behaviors.

Dexterity kommt mit einer Reihe von Behaviors. Die folgende Tabelle liefert die Interfaces, die in den Factory Type Information (FTI) angegeben werden um die jeweiligen Behaviors zu verwenden.

Interface	Beschreibung
`plone.app.dexterity.behaviors.metadata.IBasic`	Fügt die Standardfelder Titel und Beschreibung hinzu.
`plone.app.dexterity.behaviors.metadata.ICategorization`	Fügt das Fieldset Kategorisierung mit dessen Feldern hinzu
`plone.app.dexterity.behaviors.metadata.IPublication`	Fügt das Datum-Fieldset und dessen Felder hinzu.
`plone.app.dexterity.behaviors.metadata.IOwnership`	Fügt das Urheber-Fieldset und dessen Felder hinzu.
`plone.app.dexterity.behaviors.metadata.IDublinCore`	Fügt alle DublinCore-Felder der oben genannten Behaviors hinzu.
`plone.app.content.interfacess.INameFromTitle`	Berechnet den Namen aus dem Titel-Attribut.
`plone.app.dexterity.behaviors.metadata.IRelatedItems`	Fügt ein Related Items-Feld zum Kategorisierung-Fieldset hinzu.

Form-Directives

Directives ermöglichen die Konfiguration der Formulare aus den Schemata.

Dexterity verwendet plone.autoform um die auf z3c.form basierenden Hinzufügen- und Bearbeiten-Formulare zu konfigurieren. Somit lassen sich Schema mit Anmerkungen versehen, die das Formular konfigurieren.

Diese Anmerkungen können einfach mit Direktiven aus plone.directives.form und plone.directives.dexterity einem mit `plone.directives.form.Schema` erstellten Schema hinzugefügt werden.

Ein einfaches Beispiel kann so aussehen:

```
from zope import interface, schema
from plone.directives import form
...

class IExcludeFromNavigation(form.Schema):
    """Interface to exclude items from navigation.
    """

    form.fieldset('settings', label=u"Settings",
                  fields=['exclude_from_nav'])

    exclude_from_nav = schema.Bool(
                title=_(u'label_exclude_from_nav', default=u'Exclude from navigation
↪'),
                description=_(u'help_exclude_from_nav', default=u'If selected, this␣
↪item will not appear in the navigation tree'),
                default=False
                )

    form.omitted('exclude_from_nav')
```

Form-Directives

Im folgenden eine Übersicht über alle Form-Directives aus plone.directives.form:

Name	Beschreibung
`widget`	Spezifiziert ein alternatives Widget für ein Feld. Dabei wird der Feldname als Key und der Widget-Name als Value angegeben. Das Widget kann entweder eine Instanz aus z3c.form.widget sein oder ein Dotted Name eines Widgets.
`omitted`	Spart ein oder mehrere Felder aus einem Formular aus. Als Parameter kann eine Sequenz der Feldnamen angegeben werden.
`mode`	Folgende Modi sind möglich: `input`, `display``oder ``hodden`. Dabei wird der Feldname als Key und der Modus als Value angegeben.
`order_before`	Spezifiziert, dass das betreffende Feld vor einem anderen gerendert werden soll. Ist das Feld in einem zusätzlichen Schema definiert (z.B. in einem Behavior), ist der Name z.B. `ICategorization.language`. Alternativ kann auch `*` verwendet werden um das Feld am Anfang des Formulars anzuzeigen.
`order_after`	Spezifiziert, dass das betreffende Feld nach einem anderen gerendert werden soll. Ist das Feld in einem zusätzlichen Schema definiert (z.B. in einem Behavior), ist der Name z.B. `ICategorization.language`. Alternativ kann auch `*` verwendet werden um das Feld am Ende des Formulars anzuzeigen.
`primary`	Markiert ein bestimmtes Feld als Primary Field in einem Schema. Dies wird beim Zugriff per WebDAV für das Marshalling des Objekts verwendet.
`fieldset`	Erstellt ein Fieldset, das in Plone als Reiter im Bearbeiten- Formular angezeigt wird.

Security-Directives

Im folgenden die Security-Directives aus `plone.directives.dexterity`:

Name	Beschreibung
`read_permission`	Setzt die Zope3-Permission, die zum Lesen des Feldwerts erforderlich ist. Dabei wird der Feldname als Schlüssel und die Berechtigung als Wert angegeben.
`write_permission`	Setzt die Zope3-Permission, die zum Schreiben des Feldwerts erforderlich ist. Dabei wird der Feldname als Schlüssel und die Berechtigung als Wert angegeben.

Eigenschaften und Methoden der Inhaltsobjekte

Alle Felder von Dexterity Inhaltstypen sind als Eigenschaft eines Objekts verfügbar.

Im folgenden eine Liste der gebräuchlichsten Eigenschaften und Methoden von Dexterity Artikeltypen:

Eigenschaft/Methode	Typ	Beschreibung
`__name__`	`unicode`	Der Name (ID) des Objekts in seinem Container. Dies kann eine Unicode-Zeichenkette sein wobei aktuell Zope2 jedoch nur ASCII-Zeichen in URLs erlaubt.
`id`	`str`	Der Name (ID) des Objekts in seinem Container. Dies ist das ASCII-ENcoding von `__name__`.
`getId()`	`str`	Gibt den Wert der ID-Eigenschaft aus.
`isPrincipiaFolderish`	`boolean` `int`	`True` oder `1` wenn das Objekt ein Ordner ist, `False` oder `0` wenn das Objekt kein Ordner ist.
`portal_type`	`str`	Der Artikeltyp dieser Instanz.
`meta_type`	`str`	Zope2-spezifische Art, eine Klasse zu beschreiben
`title_or_id()`	`str`	Gibt den Wert des Titels aus oder sofern dieser nicht gesetzt ist, die ID-Eigenschaft.
`absolute_url()`	`str`	Die vollständige URL des Inhaltsobjekts. Berücksichtigt Virtual Hosting und die aktuelle Domain.
`getPhysicalPath()`	`tuple`	Eine Sequenz von Pfadelementen ab dem Wurzelverzeichnis der Anwendung. Die Angabe sollte nicht verwendet werden um relative URLs zu konstruieren, da Virtual Hosting nicht berücksichtigt wird.
`getIcon()`	`str`	Gibt eine Zeichenkette zurück, die als `src`-Attribut in einem `<img />`-Tag verwendet werden kann.
`title`	`unicode` `str`	Titel-Eigenschaft des Objekts. Üblicherweise Teil des Schemas eines Objekts, das durch das *ÌBasic*`-Behavior`` bereitgestellt wird.
`Title()`	`iunicode` `str`	DublinCore-Accessor für die Titel-Eigenschaft. Es kann auch ``setTitle()``verwendet werden.
`listCreators()`	`tuple`	Eine Liste von User-IDs, Der erste Ersteller ist üblicherweise der Eigentümer des Objekts. Mit `setCreators()` kann die Liste verändert werden.
`Creator()`	`str`	Der erste Ersteller, der aus der `listCreators()`-Methode ausgegeben wird. Üblicherweise wird hier der Eigentümer des Objekts ausgegeben.
`Subject()`	`tuple`	DublinCore-Accessor für Schlagwörter. Die Liste kann bearbeitet werden mit der `setSubject()`-Methode.
`Description()`	`unicode` `str`	DublinCore-Accessor für die Beschreibung, die üblicherweise mit dem `ÌBase`-Behavior mitkommt. Die Beschreibung kann geändert werden mit der `setDescription()`-Methode.
`listContributors()`	`tuples`	DublinCore-Accessor für die Liste der an dem Objekt Beteiligten. Die Beschreibung kann geändert werden mit der `setContributors()`-Methode.
`Date()`	`str`	DublinCore-Accessor für das Datum des Artikels im ISO-Format. Sofern vorhanden wird `EffectiveDate` verwendet, andernfalls `ModificationDate`.
`CreationDate()`	`str`	DublinCore-Accessor für das Erstellungsdatum des Artikels im ISO-Format.
`EffectiveDate()`	`str`	DublinCore-Accessor für das Veröffentlichungsdatum des Artikels im ISO-Format. Das Veröffentlichungsdatum kann geändert werden mit der `setEffectiveDate()`-Methode.
`ExpirationDate()`	`Date`	DublinCore-Accessor für das Ablaufdatum des Artikels im ISO-Format. Das Ablaufdatum kann geändert werden mit der `setExpirationDate()`-Methode.
`ModificationDate()`	`str`	DublinCore-Accessor für das Änderungsdatumdes Artikels im . ISO-Format.
`Language()`	`str`	DublinCore-Accessor für die Sprache des Artikels. Diese kann geändert werden mit der `setLanguage()`-Methode.
`Rights()`	`str`	DublinCore-Accessor für die Copyright-Angabe. Diese kann geändert werden mit der `setRights()`-Methode.
`created()`	`DateTime`	Gibt die Zope2-DateTime-Angabe für das Erstellungsdatum zurück. Falls diese nicht gesetzt ist, wird `January 1st, 1970` ausgegeben.
`modified()`	`DateTime`	Gibt die Zope2-DateTime-Angabe für das Änderungsdatum zurück. Falls diese nicht gesetzt ist, wird `January 1st, 1970` ausgegeben.
`effective()`	`DateTime`	Gibt die Zope2-DateTime-Angabe für das Veröffentlichungsdatum zurück. Falls diese nicht gesetzt ist, wird `January 1st, 1970` ausgegeben.
`expires()`	`DateTime`	Gibt die Zope2-DateTime-Angabe für das Ablaufdatum zurück. Falls diese nicht gesetzt ist, wird `January 1st, 1970` ausgegeben.

ZODB

Einführung in die ZODB

Relationale Datenbanken sind gut geeignet, eine große Anzahl homogener Daten zu verwalten. Sie sind jedoch wenig geeignet um hierarchische Daten abzubilden.

ORMs wie SQLAlchemy erlauben ein objektorientiertes Arbeiten wobei die Daten in einer relationalen Datenbank gespeichert werden. Die Restriktionen relationeller Datenmodelle bleiben jedoch auch hier erhalten.

Hierarchische Datenbanken z.B. LDAP oder ein Dateisystem sind sehr viel besser geeignet zur Speicherung flexibler hierarchischer Strukturen, wie sie von Content Management Systemen im allgemeinen gefordert werden. Diese Datenbankenbeherrschen jedoch im allgemeinen jedoch keine transaktionalen Semantiken.

ZODB ist transparent bei der persistenten Speicherung von Python-Objektem.

Transaktionen

Die ZODB kommt mit einem Transaktionssystem, das Nebenläufigkeit (Concurrency) und Atomarität unterstützt.

Nebenläufigkeit (Concurrency)

Beim Entwickeln für Anwendungen auf Basis der ZODB ist gewährleistet, dass gleichzeitige Anfragen, die zu einem Konflikt führen könnten, weitgehend vermieden und die Daten in der ZODB konsistent gespeichert werden. Gelegentlich auftretende *ConflictErrors* beim Schreiben können verringert werden durch Datenstrukturen, die eigene Konfliktlösungsstrategien mitbringen wie z.B. *B-Trees*.

Eine weitere Quelle für *ConflictErrors* unter hoher Schreiblast sind einige Indextypen, die keine Konfliktlösungsstrategien mitbringen. Eine Möglichkeit ist hier, nicht zu jedem Zeitpunkt den Katalog aktuell zu halten sondern asynchron den Katalog zu aktualisieren. Ein Produkt, das dies erlaubt, ist collective.indexing.

Atomarität

Atomarität bedeutet, dass eine Transaktion entweder erfolgreich abgeschlossen wird oder fehlschlägt, die Daten jedoch nie in einem inkonsistenten Status verbleiben. Falls bei einer Transaktion ein *ConflictErrors* auftritt, wiederholt Zope üblicherweise bis zu drei Mal den Versuch, diese Transaktion erneut durchzuführen.

Wird mit einem externen System wie z.B. dem SQLAlchemy-Wrapper collective.lead gearbeitet, so sollte dieser mit dem Transaktionssystem der ZODB zusammenarbeiten.

Skalierbarkeit

ZEO

Zwar ist Python beschränkt auf eine CPU, ZEO erlaubt jedoch die Verwendung mehrerer Zope-Applikationsserver, die sich eine Datenbank teilen können. Dabei sollte jedem dieser Zope-Clients eine andere CPU verwenden.

Mount-Points

ZODB unterstützt *Mount-Points*, womit sich Daten über mehrere Storages verteilen lassen. Dabei lassen sich die Memory-Cache-Settings für jede Datenbank getrennt angeben. Wird z.B. der Katalog in eine eigene Datenbank ausgelagert, so kann für diese ein deutlich höherer Memory-Cache angegeben werden um die Performance zu verbessern (siehe auch Katalog in eigener ZODB).

Storages

FileStorage schreibt die Daten in eine einzelne Datei auf dem Dateisystem. Diese Datei ist im wesentlichen ein großes Transaktionslog.

RelSotrage schreibt die Daten in eine relationale Datenbank.

DirectoryStorage Für jede Revision eines Objekts wird eine eigene Datei angelegt.

DemoStorage bietet inkrementelle Updates einer existierenden Datenbank ohne diese selbst zu aktualisieren.

Weitere Funktionen

Savepoints früher `sub-transactions`

erlaubt feingranulare Fehlersuche und *garbage collections* während einer Transaktion.

verringert den Speicherverbrauch

Undo bezeiht sich ausschließlich auf eine einzelne Datenbank. Falls also z.B. der Katalog in eine eigene Datenbank ausgelagert wurde, ist dieser nach einem Undo nicht mehr synchron mit dem Datenbestand.

Pack entfernt alte Revisionen eines Objekts.

Tipps

- Beim Lesen der `Data.fs` sollte nicht gleichzeitig geschrieben werden.
  - Vermeiden Sie `setDefault`.
  - Inplace-Migrationen sollten vermeiden werden.
- Verwenden Sie skalierbare Datenstrukturen wie `BTrees`.

Siehe auch:

- ZODB-Tutorial
- ZODB programming guide
- The ZODB Book

Releases erstellen

PyPI-Releases

Python-Pakete werden üblicherweise auf dem Python Package Index (PyPI) veröffentlicht.

Registrieren am Python Package Index (PyPI)

Falls Sie noch nicht am Python Package Index (PyPI) registriert sind, tragen Sie sich bitte zunächst im Registrierungsformular ein. Neben Name, Passwort und E-Mail-Adresse können Sie sich ggf. auch noch mit dem PGP-Key authentifizieren.

Anschließend sollten Sie Username und Passwort in einer `.pypirc`-Datei in Ihrem Home-Verzeichnis speichern, z.B.:

```
[server-login]
username:veit
password:secret
```

`distutils` benötigt diese Information beim `register`-Befehl.

Metadaten

In der `setup.py`-Datei müssen bestimmte Metadaten angegeben werden. Beim Aufruf von `python setup.py register` übermittelt das Skript diese Angaben auf `python.org`. Weitere Informationen zu diesen Metainformationen erhalten Sie in der PEP 241: Metadata for Python Software Packages.

Einige der Metadaten wie `name` und `version` werden verwendet um den Dateinamen für die Distributionen zu erstellen. Andere, wie die Trove classifiers werden ausschließlich von PyPI verwendet.

Folgende Metadaten sind notwendig:

Name Der Name des Pakets

Version Eine Versionsnummer, z.B. `4.0.1` oder `4.1rc3`.

Summary Eine einzeilige Beschreibung des Pakets.

Home-page Die URL der Homepage des Pakets.

Author Der Name des Autors des Pakets.

Author-email Die E-Mail-Adresse des Autors.

PEP 241 nennt die E-Mail-Adresse als eindeutigen Schlüssel für Paket-Kataloge.

License Der Name der Lizenz unter der das Paket veröffentlicht wird. Ggf. kann auch eine URL einer Lizenz angegeben werden.

Description Eine ausführlichere Beschreibung des Pakets.

Laut PEP 241 ist diese Beschreibung optional, doch erleichtert dies anderen das Paket im Python package Index zu finden.

Platform Eine Komma-separierte-Liste unterstützter Plattformen.

In vielen Fällen sollte hier `Any` der richtige Eintrag sein.

Außerdem können noch Trove classifiers für die Beschreibung der Software anhand eines vordefinierten Vokabulars angegeben werden. Mit ihnen lassen sich z.B. *Audience* und *Development status* des Pakets angeben.

Beispiel

Schauen wir uns als konkretes Beispiel die setup.py-Datei von vs.event an:

```
from setuptools import setup, find_packages
import os

version = '0.2.19'

setup(name='vs.event',
      version=version,
      description="An extended event content-type for Plone (and Plone4Artists␣
↪calendar)",
      long_description=open("README.txt").read() + "\n" +
                       open(os.path.join("docs", "HISTORY.txt")).read(),
      # Get more strings from https://www.python.org/pypi?%3Aaction=list_classifiers
      classifiers=[
        "Framework :: Plone",
        "Programming Language :: Python",
        "Topic :: Software Development :: Libraries :: Python Modules",
        ],
      keywords='Zope Plone Event Recurrence Calendar Plone4Artists',
      author='Veit Schiele, Anne Walther, Andreas Jung',
      author_email='vs.event@veit-schiele.de',
      url='http://svn.plone.org/svn/collective/vs.event',
      license='GPL',
      packages=find_packages(exclude=['ez_setup']),
      namespace_packages=['vs'],
      include_package_data=True,
      zip_safe=False,
      install_requires=[
          'setuptools',
          'python-dateutil',
          'dateable.chronos',
          'dateable.kalends',
          'collective.calendarwidget',
          'Products.DataGridField',
          'zope.app.annotation',
          # -*- Extra requirements: -*-
      ],
      entry_points="""
      # -*- Entry points: -*-
      """,
      )
```

Beachten Sie bitte, dass `long_description` aus zwei externen Dateien zusammengesetzt wird, nämlich `README.txt` und `HISTORY.txt`.

Dateien hinzufügen

include_package_data Ist der Wert auf `True` gesetzt, so fügt `setuptools` automatisch alle Dateien innerhalb des Paketverzeichnisses hinzu, die entweder unter CVS- oder SVN-Versionskontrolle stehen oder in einer `MANIFEST.in`-Datei beschrieben sind. Für aktuellere Versionen von SVN sowie weitere Versionsverwaltungen wie git und Mercurial sind jedoch Plugins erforderlich:

- setuptools_subversion
- setuptools_bzr
- setuptools_hg
- setuptools-git

Weitere Informationen erhalten Sie in Including Data Files.

Überrüfen

PKG-INFO

Distutils `PKG-INFO` kann verwendet werden umd die Metadaten aus `setup.py` zu überprüfen. Beim Aufruf von `python setup.py sdist` erstellt distutils einen sog. *source tarball* im `dist`-Verzeichnis. Der *tarball* enthält eine `PKG-INFO`-Datei auf oberster Verzeichnisebene.

PyPI-Testing-Site

Schließlich kann zum Testen kann auch die PyPI-Testing-Site verwendet werden.

Registrieren

Falls Sie einen PyPI-Account haben und die Zugangsdaten eingetragen sind in `~/.pypirc` kann das Paket bei PyPI registriert werden mit:

```
$ python setup.py register
```

Überprüfen der Registrierung

Rufen Sie mit Ihrem Webbrowser https://www.python.org/pypi auf. Dort sollten Sie Ihr Paket in der Liste der letzten 20 aktualisierten Pakete sehen. Sofern Sie angemeldet sind, erscheint Ihr Paket auch in der linken Navigation unter der Überschrift *Your Packages*.

Wenn Sie nun auf den Namen Ihres Pakets klicken, wird Ihnen das Paket angezeigt. In dieser Ansicht erhalten Sie auch einen *Edit*-Link mit dem Sie die generierte Seite korrigieren können.

Upload

```
$ python setup.py sdist upload
```

pythonpackages.com

pythonpackages.com bietet eine alternative Möglichkeit, Releases Ihrer Python-Packages aus einem Github-Repository zu erstellen und zu testen. Weitere Informationen hierzu erhalten Sie unter Introduction. Darüberhinaus bietet `pythonpackages.com` auch einen Test Package Index:

```
$ cd my.package
$ python setup.py sdist upload -r http://index.pythonpackages.com
$ pip install my.package -i http://index.pythonpackages.com
```

Siehe auch

jarn.mkrelease Einfache Integration von Releases in Buildout-Projekte mit gepinnten Versionen.

jarn.viewdoc erstellt eine Voransicht der Dokumentation eines Pakets, bevor eine Release erstellt wird.

zest.releaser automatisiert die Aktualisierung von Versionsnummer, Änderungshistorie und Tagging.

gocept.zestreleaser.customupload Plugin für zest.releaser, das das Kopieren eines zuvor erstellten Egg zu einem konfigurierbaren Ziel erlaubt.

Private Releases

sdistmaker

Mit sdistmaker gibt es eine einfache Möglichkeit, *sdist tarballs* in svn-Repositories bereitzustellen.

sdistmaker übernimmt folgende Aufgaben:

#. Es durchsucht die `tags`-Verzeichnisse Ihres svn-Repository. Dabei kann sdistmaker auch auf bestimmte Bereiche Ihres Repository eingeschränkt werden. #. Für jeden Tag erstellt es eine *source distribution* mit ``python setup.py sdist``.

1. Die *source distribution* wird anschließend in einem Unterverzeichnis Ihres Projekts gespeichert, ähnlich wie in https://pypi.python.org/simple/.

Installation

`sdistmaker` lässt sich einfach installieren mit:

```
$ easy_install sdistmaker
```

Anschließend stehen Ihnen zwei Skripte zur Verfügung:

make_sdist ist im wesentlichen für Testzwecke gedacht. Dabei können Sie unter Angabe der `tag`-URL und des Zielverzeichnisses einzelne Releases erstellen.

sdists_from_tags Es durchsucht die svn-Struktur nach geeigneten `tags`-Verzeichnissen und erstellt aus diesen entsprechende Releases.

Konfiguration

Zunächst sollte sdistmaker seine eigene Basiskonfiguration erstellen mit:

```
$ sdists_from_tags --print-example-defaults
```

Speichern Sie die Ausgabe in eine Konfigurationsdatei, z.B. `defaults.py`.

Anschließend können Sie diese Konfiguration verwenden mit:

```
$ sdists_from_tags --defaults-file=defaults.py
```

Schließlich werden Sie `sdists_from_tags` regelmäßig aufrufen wollen, entweder als Cron-Job, svn post-commit-hook etc.

Verwendung in Buildout

`sdistmaker` kann mit Buildout folgendermaßen verwendet werden:

```
[buildout]
parts = sdists

[sdists]
recipe = zc.recipe.egg
eggs = sdistmaker
scripts = sdists_from_tags
# arguments =
#      defaults_file='${buildout:directory}/defaults.py'
```

Dabei wird die `defaults.py`-Datei auf dieselbe Weise erzeugt wie oben beschrieben.

sdistmaker und PyPI

Üblicherweise kann immer nur ein Index für `easy_install` angegeben werden. Um nun neben dem `sdistmaker`-Index auch den von PyPI verwenden zu können, kann z.B. eine Redirect-Anweisung definiert werden, falls im `sdistmaker`-Index nichts gefunden wird:

```
# Allow indexing
Options +Indexes
IndexOptions FancyIndexing VersionSort

# Start of rewriterules to use our own var/private/* packages
# when available and to redirect to pypi if not.
RewriteEngine On
# Use our robots.txt:
RewriteRule ^/robots.txt - [L]
# Use our apache's icons:
RewriteRule ^/icons/.* - [L]
# We want OUR index.  Specified in a weird way as apache
# searches in a weird way for index.htm index.html index.php etc.
RewriteRule ^/index\..* - [L]

# Use our var/private/PROJECTNAME if available,
# redirect to pypi otherwise:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
RewriteCond /path/on/server/var/private/$1 !-f
RewriteCond /path/on/server/var/private/$1 !-d
RewriteRule ^/([^/]+)/?$ http://pypi.python.org/pypi/$1/ [L]

# Use our var/private/PROJECTNAME/project-0.1.tar.gz if available,
# redirect to pypi otherwise:
RewriteCond /path/on/server/var/private/$1 !-d
RewriteRule ^/([^/]+)/([^/]+)$ http://pypi.python.org/pypi/$1/$2 [L]
```

Verwenden des Index

Dieser Index kann nun sowohl mit EasyInstall als auch mit Buildout aufgerufen werden:

EasyInstall

```
$ easy_install -i https://packages.veit-schiele.de/ vs.event
```

Buildout Sie können den Index in der Buildout-Konfigurationsdatei angeben:

```
[buildout]
index = https://packages.veit-schiele.de/
parts =
    ...
```

gocept.zestreleaser.customupload

gocept.zestreleaser.customupload ist ein Plugin für zest.releaser, das das Hochladen erstellter Eggs via Secure copy (SCP) zu vorher konfigurierten Zielen erlaubt.

Um es zu verwenden, kann in `~/.pypirc` z.B. folgendes konfiguriert werden:

```
[gocept.zestreleaser.customupload]
vs = scp://download.veit-schiele.de:/var/www/packages
example = https://dav.veit-schiele.de:/var/www/example
```

Falls das veröffentlichte Paket mit einem der Schlüsselwörter (`vs`, `example`) beginnt, werden Sie gefragt, ob das Egg auf den angegebenen Server hochgeladen werden soll.

Zum Weiterlesen

Allgemein

The Hitchhiker's Guide to Packaging Umfassende Anleitung zur Erstellung eines Python-Pakets.

Versionsschema

PEP 386-kompatibles Versionsschema Damit die Versionen von anderen Werkzeugen wie distutils und setuptools auch in Zukunft automatisiert ausgewertet werden können, sollten das Versionsschema des PEP 386 eingehalten werden.

Metadata 1.2-Schema Das in Python 3.3 verwendete distutils2 wird dieses Metadatenschema unterstützen.

3.15.4 Glossar

Acquisition Acquisition ist ein Mechanismus, der es Objekten erlaubt, Attribute aus ihrer Umgebung zu erhalten. Eine ausführliche Beschreibung, wie in Zope Acquisition verwendet werden kann, finden Sie im Zope Book.

Adapter In der Zope Component Architecture (ZCA) sind Adapter Komponenten, die aus anderen Komponenten erstellt werden um sie einem bestimmten Interface zur Verfügung zu stellen:

```
>>> class IPerson(interface.Interface):
...     name = interface.Attribute("Name")
>>> class PersonGreeter:
...
...     component.adapts(IPerson)
...     interface.implements(IGreeter)
...
...     def __init__(self, person):
...         self.person = person
...
...     def greet(self):
...         print "Hello", self.person.name
```

Die Klasse definiert einen Constructor, der ein Argument für jedes adaptierte Objekt nimmt.

`component.adapts` deklariert, was angepasst werden soll.

`adaptedBy` gibt eine Liste der Objekte aus, die adaptiert werden:

```
>>> list(component.adaptedBy(PersonGreeter)) == [IPerson]
True
```

`provideAdapter` Sofern nur ein Interface angeboten wird, kann dieses einfach bereitgestellt werden mit:

```
>>> component.provideAdapter(PersonGreeter)
```

Ebenso können spezifische Argumente zum Registrieren eines Adapters angegeben werden:

```
>>> class VeitPersonGreeter(PersonGreeter):
...     name = 'Veit'
...     def greet(self):
...         print "Hello", self.person.name, "my name is", self.name
>>> component.provideAdapter(
...                         VeitPersonGreeter, [IPerson], IGreeter, 'veit')
```

oder als *keyword arguments*:

```
>>> class ChrisPersonGreeter(VeitPersonGreeter):
...     name = "Chris"
>>> component.provideAdapter(
...     factory=ChrisPersonGreeter, adapts=[IPerson],
...     provides=IGreeter, name='chris')
```

`queryAdapter` oder `getAdapter` kann für For *named adapters* verwendet werden:

```
>>> component.queryAdapter(Person("Chris"), IGreeter, 'veit').greet()
Hello Chris my name is Veit
>>> component.getAdapter(Person("Chris"), IGreeter, 'veit').greet()
Hello Chris my name is Veit
```

Falls kein Adapter vorhanden ist, gibt `queryAdapter` einen Standardwert zurück wohingegen `getAdapter` eine Fehlermeldung ausgibt:

```
>>> component.queryAdapter(Person("Chris"), IGreeter, 'daniel')
>>> component.getAdapter(Person("Chris"), IGreeter, 'daniel')
...
Traceback (most recent call last):
...
ComponentLookupError: (...Person...>, <...IGreeter>, 'daniel')
```

`queryMultiAdapter` oder `getMultiAdapter` gibt die Adapter mehrerer Objekte zurück.

Wenn wir z.B. einen Adapter mit mehreren Objekten erstellen:

```
>>> class TwoPersonGreeter:
...
...     component.adapts(IPerson, IPerson)
...     interface.implements(IGreeter)
...
...     def __init__(self, person, greeter):
...         self.person = person
...         self.greeter = greeter
...
...     def greet(self):
...         print "Hello", self.person.name
...         print "my name is", self.greeter.name
```

können wir diesen Multi-Adapter anfragen mit `queryMultiAdapter` oder `getMultiAdapter`:

```
>>> component.queryMultiAdapter((Person("Chris"), Person("Veit")),
...                                 IGreeter).greet()
Hello Chris
my name is Veit
```

AJAX Asynchronous JavaScript and XML.

API Application Programming Interface.

Schnittstelle, die Funktionen eines Programms zugänglich macht.

Archetypes Archetypes ist ein Framework um neue Artikeltypen in Plone aus Schemadefinitionen zu erstellen. Die Seiten zur Ansicht und zum Editieren lassen sich dabei automatisch generieren.

ATCT Mit Archetypes geschriebene Artikeltypen, die zusammen mit Plone ausgeliefert werden.

Browserlayer Browserlayer vereinfachen die Registrierung visueller Elemente wie Views, Viewlets etc. sodass diese Elemente nur in den Sites erscheinen, in denen sie explizit installiert wurden.

1. Zunächst wird ein Marker-Interface z.B. in `vs.theme/vs/theme/browser/interfaces.py` erstellt:

```
from plone.theme.interfaces import IDefaultPloneLayer
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
class IThemeSpecific(IDefaultPloneLayer):
    """Marker interface that defines a Zope 3 browser layer.
       If you need to register a viewlet only for the
       "vs.theme" theme, this interface must be its layer.
    """
```

2. Anschließend kann dieses Marker-Interface regsitriert werden in `vs.theme/vs/theme/profiles/default/browserlayer.xml`, z.B.:

```
<layers>
    <layer name="vs.theme"
           interface="vs.theme.interfaces.IThemeSpecific" />
</layers>
```

3. Schließlich können visuelle Komponenten für diesen Browserlayer registriert werden in `vs.theme/vs/theme/browser/configure.zcml`, z.B.:

```
<browser:page
    for="Products.CMFCore.interfaces.ISiteRoot"
    name="dashboard"
    permission="plone.app.portlets.ManageOwnPortlets"
    class="plone.app.layout.dashboard.dashboard.DashboardView"
    template="templates/dashboard.pt"
    layer=".interfaces.IThemeSpecific"
    />
```

Buildout Buildout erlaubt, identische Entwicklungsumgebungen einfach aufzusetzen. Hierzu nutzt buildout die Fähigkeit der setuptools, automatisch Abhängigkeiten aufzulösen und Aktualisierungen durchzuführen (s.a.: Buildout's documentation).

Catalog Der Katalog ist ein interner Index der Inhalte einer Plone-Site. Dabei kann auf den Catalog auch über das ZMI als `portal_catalog` zugegriffen werden.

Collective Collective ist ein Subversion-Repository für die Plone-Community um Zusatzprodukte bereitzustellen.

CSS CSS ist ein Web-Standard zur Darstellung von Inhalten. Der Standard ist beschreiben auf der W3C-Website. Eine Einführung in CSS finden Sie unter

Siehe auch: Einführung in Cascading Style Sheets (CSS)

Decorator Ein sog. Wrapper um eine Python-Funktion oder -Klasse, die die Funktion oder Klasse als sein erstes Argument nimmt und ein beliebiges Objekt zurückgibt. In Plone werden verschiedene Decorator verwendet, so z.B. memoize zum Caching der Werte von Funktionen und Methoden und profilehooks für das Erstellen von Profilen einzelner Funktionen

Sehen Sie auch PEP 318.

Distribution Eine Distribution besteht in Python aus einem Verzeichnis mit einer `setup.py`-Datei und anderen Ressourcen. Die Metaangaben in der `setup.py`-Datei können u.a. die Versionsnummer, Abhängigkeiten und Lizenzinformationen enthalten.

Werkzeuge wie Setuptools, Distribute oder auch Buildout können die Metainformationen verwenden um Installationen in verschiedenen Versionen zu erhalten, Abhängigkeiten aufzulösen etc.

DocFinderTab DocFinderTab ist ein Produkt, das alle Klassen und Methoden eines Objekts im Zope Management Interface (ZMI) auflistet.

Doctest Eine spezielle Syntax zum Schreiben von Tests. Ein Vorteil von Doctests ist, dass sie mit dem Test auch gleich die Dokumentation mitliefern. Als Nachteilig hat sich herausgestellt, dass nicht eine Untermenge der

Doctests durchlaufen werden kann. Zudem werden beim Fehlschlagen eines Tests die weiteren Tests nicht mehr durchlaufen. Schließlich wird der Code auf eine besondere Weise ausgeführt, die schwieriger nachzuvollziehen und zu analysieren sind.

DTML DTML ist eine serverseitige Template-Sprache, mit der sich dynamische Inhalte erstellen lassen. Plone verwendet für die Erstellung von HTML jedoch ZPT, sodass DTML nur noch für nicht XML-konforme Inhalte wie SQL-Anfragen, Mail- und CSS-Generierung verwendet wird.

Easy Install Easy Install ist ein Python-Modul mit dem der Python Package Index durchsucht werden kann und das die Pakete in die globale Python-Umgebung installiert. Neben Buildout werden wir nur noch ZopeSkel mit `easy_install` installieren, alle weiteren Eggs werden von Buildout in das lokale Buildout-Projekt heruntergeladen, unter anderem um Versionskonflikte zu vermeiden.

Egg Ein binäres Distributionsformat, das von den Setuptools und Distribute verwendet wird. Dabei wird ür jede Plattform und jede Python-Version ein spezifisches Egg erstellt.

Daher können Source-Distributionen, die meist nur ein komprimiertes Archiv des Codes und der Metaangaben sind, flexibler eingesetzt werden. Umgekehrt muss für Paketen, die binäre Abhängigkeiten besitzen (wie z.B. in C geschriebene Erweiterungen) die notwendigen Compiler und Bibliotheken verfügbar sein, um die Source-Distribution installieren zu können.

Event Die Zope Component Architecture (ZCA) ermöglicht, *Events* an bestimmte Handler zu schicken.

1. Im Folgenden erstellen wir zwei Beispielklassen, die `zope.component.event` für das Dispatching benötigen:

```
>>> import zope.component.event

>>> class Event1(object):
...     pass
>>> class Event2(Event1):
...     pass
```

2. Anschließend werden zwei *Handler* für diese Event-Klassen erstellt:

```
>>> called = []
>>> import zope.component
>>> @zope.component.adapter(Event1)
... def handler1(event):
...     called.append(1)
>>> @zope.component.adapter(Event2)
... def handler2(event):
...     called.append(2)
```

3. Diese Handler werden nun registriert mit:

```
>>> zope.component.provideHandler(handler1)
>>> zope.component.provideHandler(handler2)
```

4. Nun Überprüfen wir, ob die *Handler* auch tatsächlich aufgerufen wurden:

```
>>> from zope.event import notify
>>> notify(Event1())
>>> called
[1]
>>> del called[:]
>>> notify(Event2())
>>> called.sort()
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
>>> called
[1, 2]
```

Siehe auch:

- Events
- Object events

File descriptor Ein *file descriptor* ist ein abstrakter Indikator für den Zugriff auf eine Datei.

Dabei verwendet ein ZEO-Server für jeden Storage je Client-Verbindung 3 dieser Deskriptoren. Bei einem ZEO-Server mit zehn Storages und 36 ZEO-Clients würden also 10×36×3 *file sescriptors* benötigt. Üblicherweise verwendet der ZEO-Server jedoch nicht die erforderlichen 1080 *file descriptors* sondern nur 1025. Dies führt dann dazu, dass ZEO-Clients mit der Zeit keine Seiten mehr ausliefern und in deren Log-Dateien `ECONNREFUSED`-Meldungen erscheinen.

Die Ursache hierfür ist, dass Python üblicherweise kompiliert wird mit einer maximalen Anzahl von 1024 *file descriptors* je Prozess. Dies lässt sich ändern indem in `/usr/include/bits/typesizes.h` der Wert für `define __FD_SETSIZE` hochgesetzt wird, z.B. auf 2048. Nach einem Neukompilieren von Python kann der ZEO-Server dann auch alle 1080 *file descriptors* verwenden.

Andreas Gabriel hat ein Skript geschrieben, mit dem sich die maximale Anzahl der Verbindungen testen lässt: zeo-check-max-connections.py

Handler Handler sind eine spezifische Form von Subscribern, die nichts bereitstellen und meistens von Events aufgerufen werden.

Beim Aufruf eines *Handlers* wird kein Rückgabewert erwartet. Auch bieten *Handler* keine API an. Daher werden *Handler* meist als Funktion und nicht als Klasse implementiert. Zum Beispiel:

```
>>> import datetime
>>> def documentCreated(event):
...     event.doc.created = datetime.datetime.utcnow()
>>> documentCreated = component.adapter(IDocumentCreated)(documentCreated)
```

Die letzte Zeile markiert den *Handler* als Adapter von `IDocumentCreated`-Events. Nun wird der *Handler* noch registriert mit:

```
>>> component.provideHandler(documentCreated)
```

Schließlich kann die `handle`-Funktion verwendet werden um *Handlers*, die für einen *Event* registriert sind, aufzurufen:

```
>>> component.handle(DocumentCreated(doc))
>>> doc.created.__class__.__name__
'datetime'
```

Siehe auch:

- Handlers

i18n Präparierung des Quellcodes, sodass er ohne weitere Änderung in verschiedene Sprachen übersetzt werden kann. i18n wird durch den ersten und letzten Buchstaben von *Internationalization* und die Anzahl der dazwischenliegenden Zeichen gebildet.

Die Übersetzungsarbeit selbst wird dann l10n genannt.

Integrationstest Ein Test, ob eine Komponente mit anderen Komponenten zusammen läuft. Die meisten Tests, die für Plone-Produkte geschrieben werden, sind Integrationstests da das gesamte Plone-Framework benötigt wird, um

die gewünschten Testergebnisse zu erhalten. Ein Beispiel für einen Integrationstest ist der Test, ob ein Objekt eines neuen Artikeltyps erstellt werden kann nachdem dieses Produkt in einer Plone-Site installiert wurde.

Interface Zope-Interfaces sind Objekte, die das externe Verhalten desjenigen Objekts spezifizieren, das sie bereitstellt. Dies geschieht durch:

- Informelle Dokumentationen in Doc-Strings.
- Attribut-Definitionen
- Invarianten, also Bedingungen für Objekte, die dieses Interface bereitstellen.

Dabei spezifiziert ein Interface die Charakteristiken eines Objekts, sein Verhalten und seine Fähigkeiten.

Interfaces machen Angaben, *was* ein Objekt bereitstellt, nicht *wie* es bereitgestellt wird. Sie beruhen auf dem Design By Contract-Modell.

Während in einigen anderen Programmiersprachen Interfaces ein Bestandteil der Sprache selbst sind, werden in Python mit der ZCA Interfaces als Meta-Klasse implementiert, die ererbt werden kann.

Für eine Komponente wird zunächst dessen Interface erstellt. Interface- Objekte werden üblicherweise mit *Python Class Statements* erstellt, sind jedoch selbst keine Klassen sondern Objekte. Ein Interface-Objekt wird nun als *Subclass* von `zope.interface.Interface` erstellt:

```
from zope.interface import Interface

class IHello(Interface):

    def hello(name):
        """Say hello to somebody"""
```

Durch das Subclassing von `zope.interface.Interface` wird nun das Interface-Objekt `IHello` erstellt:

```
>>> IHello
<InterfaceClass __main__.IHello>
```

Interfaces können auch verwendet werden um ein bestimmtes Objekt zu einem spezifischen Typ gehört. Ein solches Interface ohne Attribute und Methoden wird Marker-Interface genannt. Ein solches Interface kann z.B. so aussehen:

```
>>> from zope.interface import Interface

>>> class ISpecialGreeting(Interface):
...     """A special greeting"""
```

Gelegentlich sind Regeln mit einem oder mehreren Attributen für das Interface einer Komponente erforderlich. Solche Regeln werden Invarianten genannt und können mit `zope.interface.invariant` erstellt werden.

So kann z.B. für ein `person`-Objekt mit den Attributen `name`, `email` und `phone` ein Validator erstellt werden, der überprüft ob entweder `email` und `phone` angegeben wurden.

1. Zunächst wird nun ein aufrufbares Objekt entweder als Funktion oder Instanz erstellt:

   ```
   >>> def contacts_invariant(obj):
   ...
   ...     if not (obj.email or obj.phone):
   ...         raise Exception(
   ...             "At least one contact info is required")
   ```

#. Anschließend wird das Interface des `person`-Objekts mit der `zope.interface.invariant`-Funktion definiert:

```
>>> from zope.interface import Interface
>>> from zope.interface import Attribute
>>> from zope.interface import invariant

>>> class IPerson(Interface):
...
...     name = Attribute("Name")
...     email = Attribute("Email Address")
...     phone = Attribute("Phone Number")
...
...     invariant(contacts_invariant)
```

1. Schließlich kann die `validateInvariants`-Methode verwendet werden:

```
>>> from zope.interface import implements

>>> class Person(object):
...     implements(IPerson)
...
...     name = None
...     email = None
...     phone = None

>>> veit = Person()
>>> veit.email = u"veit@example.org"
>>> IPerson.validateInvariants(veit)
>>> chris = Person()
>>> IPerson.validateInvariants(jill)
Traceback (most recent call last):
...
Exception: At least one contact info is required
```

Dieses `Hello`-Interface kann nun assoziiert werden mit einer konkreten Klasse, in der das Verhalten definiert wird. In unserem Beispiel:

```
class HelloComponent:

    implements(IHello)

    def hello(self, name):
        return "Hello %s!" % name
```

Die neue Klasse `HelloComponent` implementiert das `Hello`-Interface.

Dabei kann eine solche Klasse auch mehrere Interfaces implementieren. Sollen also Instanzen unserer `HelloComponent` zusätzlich ein `Other`-Interface implementieren, wird einfach eine Sequenz der Interface-Objekte in der `HelloComponent`-Klasse bereitgestellt:

```
class HelloComponent:

    implements(IHello, IOther)
    ...
```

Mit `implementedBy` kann ein Interface gefragt werden, ob eine bestimmte Klasse oder Instanz dieses Interface implementiert. So sollte z.B. die Überprüfung, ob eine Instanz der `HelloComponent`-Klasse `Hello``implementiert den Wert ``true` zurückliefern:

```
IHello.implementedBy(HelloComponent)
```

Interfaces können einfach erweitert werden, so kann z.B. unser `IHello`-Interface um eine Methode `lastGreeted` erweitert werden:

```
class ISmartHello(IHello):
    """A Hello object that remembers who is greeted"""

    def lastGreeted(self):
        """Returns the name of the last person greeted."""
```

`getBases` gibt eine Liste der Interfaces aus, die durch dieses Interface erweitert wurden, z.B.:

```
>>> ISmartHello.getBases()
(<InterfaceClass __main__.IHello>,)
```

`extends` gibt `true` oder *false*`, je nachdem, ob ein Interface ein anderes erweitert oder nicht, z.B.:

```
>>> ISmartHello.extends(IHello)
True
>>> IOther(Interface):
...     pass
>>> ISmartHello.extends(IOther)
False
```

`names` gibt eine Liste der Namen aller Items aus, die durch das Interface beschrieben werden, z.B.:

```
>>> IUser.names()
['getUserName', 'getPassword']
```

`namesAndDescriptions` gibt eine Liste von Tuples `(name, description)` aus, z.B.:

```
>>> IUser.namesAndDescriptions()
[('getUserName', <zope.interface.interface.Method.Method object at 80f38f0>),
('getPassword', <zope.interface.interface.Method.Method object at 80fded8>)]
```

- zope.interface
- Components and Interfaces

jQuery JavaScript-Bibliothek, die die Traversierung und das Event-Handling von HTML-Dokumenten vereinfacht. So lässt sich z.B. in einem Einzeiler angeben, dass alle PDFs in einem neuen Fenster geöffnet werden sollen:

```
jQuery("#content a[ @href $= '.pdf']").attr('target', '_blank');
```

Weitere Informationen zu jQuery erhalten Sie unter:

- http://jquery.com/
- http://docs.jquery.com

Und mit FireQuery gibt es eine Firefox-Extension, die in Firebug integriert ist.

Kinetic Style Sheet In Plone 3 verwendetes AJAX-Framework.

Kupu Kupu ist ein graphischer HTML-Editor, der mit Plone zusammen ausgeliefert wird.

l10n l10n ist die Übersetzung in eine oder mehrere spezifische Sprachen. l10n wird durch den ersten und letzten Buchstaben von *Localization* und die Anzahl der dazwischenliegenden Zeichen gebildet.

Layer Ein Layer ist eine Sammlung von Templates und Skripten. Dabei bildet ein Stapel von Layern einen Skin. Im ZMI können Sie im *Properties*-Reiter des `portal_skins`-Tool die Definition von Skins über Layer sehen und im *Content*-Reiter sehen Sie diese Layer als *Filesystem Directory View* oder *Folder*.

LDAP LDAP beschreibt die Kommunikation zwischen einem sog. LDAP-Client und einem Verzeichnisdienst. EIn solches Verzeichnis kann z.B. ein Adressbuch sein, das Personendaten enthält. Der LDAP-Client kann dann ein E-Mail-Programm sein, das bei der Suche nach einer Adresse eine Anfrage an den LDAP-Server, der diese Adressinformationen bereitstellt, stellt.

LDAP ist spezifiziert in RFC 4511.

Logging Für des Entwickelns stellt Plone einen eigenen Logger bereit: plone_log:

```
from logging import getLogger
log = getLogger('Plone')
log.info('Debug: %s \n%s', summary, text)
```

Wie `plone_log` verwendet werden kann, finden Sie z.B. in setConstrainTypes.cpy:

```
...
plone_log=context.plone_log
constrainTypesMode = context.REQUEST.get('constrainTypesMode', [])
currentPrefer = context.REQUEST.get('currentPrefer', [])
currentAllow = context.REQUEST.get('currentAllow', [])
plone_log( "SET: currentAllow=%s, currentPrefer=%s" % ( currentAllow,␣
↪currentPrefer ) )
...
```

Manager Rolle, die in Zope alle Berechtigungen erhält bis auf *Take Ownership*.

METAL Macro Expansion Tag Attribute Language

METAL kann für das Verarbeitung von Macros für HTML und XML verwendet werden. Sie kann zusammen mit TAL und TALES verwendet werden.

Macros erlauben Definitionen in einer Datei, die von einer oder mehreren anderen Dateien verwendet werden können. Dabei werden macros immer in vollem Umfang verwendet.

`metal:define-macro` Definieren eines Macros als Element und dessen Teilbaum.

`metal:use-macro` Verwenden eines Macros wobei der Ausdruck in Zope immer die Angabe des Pfads ist, der auf ein Macro in einem anderen Template verweist.

`metal:define-slot` Definieren eines Slots, der angepasst werden kann.

Wird ein Macro verwendet, so können dessen Slots ersetzt werden um das Macro anzupassen. Slot-Definitionen liefern dann den Standard-Inhalt für diesen Slot, der verwendet wird, sofern das Macro nicht angepasst wird.

Die Anweisung `metal:define-slot` muss innerhalb von `metal:define-macro` verwendet werden. Darüberhinaus müssen die Slot-Namen innerhalb eines Macros einheitlich sein.

`metal:fill-slot` Anpassen eines Macros indem ein Slot dieses Macros ersetzt wird.

Die Anweisung `metal:fill-slot``muss innerhalb von ``metal:use-macro` verwendet werden. Darüberhinaus müssen die Slot-Namen innerhalb eines Macros einheitlich sein.

If the named slot does not exist within the macro, the slot contents will be silently dropped.

- The Zope2 Book: METAL Overview

Monkey Patch Ein Monkey Patch erlaubt die Änderung des Verhaltens von Zope oder eines Produkts ohne den Original-Code verändern zu müssen.

Ein Monkey Patch lässt sich einfach mit collective.monkeypatcher erstellen. Hierzu tragen wir in die `configure.zcml`-Datei folgendes ein:

```
<configure
    ...
    xmlns:monkey="http://namespaces.plone.org/monkey">
    ...
    <monkey:patch
        description="TinyMCE JSON Folder listing should ignore INavigationRoot"
        class="Products.TinyMCE.adapters.JSONFolderListing.JSONFolderListing"
        original="getListing"
        replacement=".patches.getListing"
        />
```

Nun erstellen wir unseren Pach, indem wir aus `Products.TinyMCE.adapters.JSONFolderListing.JSONFolderListing` die Methode `getListing` in die Datei `patches.py` kopieren und entsprechend anpassen.

Anmerkung: Mit collective.monkeypatcherpanel wird ein Zope2-Control-Panel angelegt, das die mit `collective.monkeypatcher` erstellten Monkey Patches anzeigt.

mr.bob mr.bob ist ein Dateisystem-Template- Renderer. Er ermöglicht, aus einer Vorlage eine Verzeichnisstruktur zu erstellen, die das Erstellen von Python-Paketen deutlich vereinfacht.

Weitere Informationen zur Installation, den Vorlagen und Standardeinstellungen erhalten Sie im Abschnitt Referenzen des Plone- Entwicklerhandbuchs.

Siehe auch:

- mr.bob's documentation
- Git repository

Namensraum Plone verwendet verschachtelte Pakete um Namensräume zu bilden, die durch Pfadnamen eindeutig angesprochen werden können und so Kollisionen mit anderen Paketen vermeiden helfen. Dabei nutzt Plone eine Funktion der Setuptools, womit mehrere getrennte Python Packages ausgeliefert werden können, die einen gemeinsamen Top-level-Namespace teilen, z.B. `plone.theme` und `plone.portlets`.

PAS PAS ist ein Framework zur Authentifizierung in Zope. PAS ist ein Zope-`acl_users`-Ordner, das Plugins verwendet um verschiedene Authentifizierungsschnittstellen bereitzustellen.

Paste Paste ist ein WSGI-Entwicklungs- und Deployment-System, das von Ian Bicking entwickelt wurde.

PDB PDB ist ein interaktiver Debugger, mit dem schrittweise durch den Code gegangen werden kann um Probleme aufzufinden.

Um einen einfachen *Breakpoint* zusetzen, kann folgendes angegeben werden:

```
import pdb; pdb.set_trace()
```

Anschließend sollte Zope neu im Vordergrund gestartet werden mit:

```
$ ./bin/instance fg
```

Anschließend sollte der Code ausgeführt werden, für den der Breakpoint gesetzt wurde. Das Terminal, in dem die Instanz gestartet wurde, sollte dann eine Debug-Session öffnen mit folgender Angabe:

```
-> Pdb().set_trace()
(Pdb)
```

Sie können nun mit `r` (*Return*) den `set_trace()`-Aufruf verlassen und so schrittweise den Code untersuchen.

Wenn ein Fehler in einer Methode auftritt, die häufig ausgeführt wird, ist es jedoch nur lästig, sehr häufig *Return* angeben zu müssen. Daher empfiehlt sich, das sog. *post-mortem*-Idiom zu verwenden:

```
try:
    [YOUR CODE HERE]
except:
    import pdb, sys
    e, m, tb = sys.exc_info()
    pdb.post_mortem(tb)
```

Anschließend sollte die Zope-Instanz wieder im Vordergrund gestartet werden. Nun wird `pdb` nur noch aufgerufen, wenn ein Fehler im Abschnitt `[YOUR CODE HERE]` auftritt.

Um zu gewährleisten, dass derselbe pdb-Breakpoint in einer Deubug-Session nicht mehrfach eine Exception ausgibt, kann die Variable `PDB_ACTIVE` auf `1` gesetzt werden:

```
if not globals().get( 'PDB_ACTIVE', 0 ):
    globals()['PDB_ACTIVE'] = 1
    import pdb; pdb.set_trace()
```

Anmerkung: Entfernen Sie bitte wieder die *debugging hooks* bevor der Code in das Repository eingecheckt wird.

Eine `~/.pdbrc`-Konfigurationsdatei kann verwendet werden um sich einige Shortcuts zum Debuggen zu erstellen, z.B.:

```
# Print a sorted dictionary.
# %1 is the dict
# %2 is the prefix for the names.
alias p_ for k in sorted(%1.keys()): print "%s%-15s= %-80.80s" % ("%2",k,repr(
↪%1[k]))

# Print the member variables of something
alias pi p_ %1.__dict__ %1.

# Print the member variables of self
alias ps pi self

# Print locals
alias pl p_ locals() local:

# Next list and step list
alias nl n;;l
alias sl s;;l
```

Um weitere Hilfsfunktionen in pdb nutzen zu können, lassen sich auch externe Python-Dateien in die `~/.pdbrc`-Datei einbinden – siehe hierzu PdbRcIdea.

Python Documentation: Debugger Commands Verwendung von pdb

Ken Manheimer: Conversing With Zope Ausführliche Anleitung für die Verwendung von pdb mit Zope

Stephen Ferg: Debugging in Python Eine kurze praktische Einführung in `pdb`

Jeremy Jones: Interactive Debugging in Python Eine ausführliche Anleitung mit fortgeschrittenen Beispielen

PLIP Vergleichbar mit Pythons PEPs (Python Enhancement Proposals).

Das Plone-Team strukturiert und organisiert mit PLIPs den Entwicklungsprozess von Plone.

Plone Plone ist ein, auf dem freien Webanwendungsserver `Zope`_ aufbauendes Enterprise-Content-Management-System, das in der Programmiersprache Python geschrieben ist.

Es kann für Intranet- und Extranet-Anwendungen, als Dokumentenmanagementsystem und als Groupware eingesetzt werden. Zahlreiche Erweiterungen ermöglichen den Einsatz für weitere Aufgaben, z.B. im eLearning, Webshop oder Bilddatenbank.

Portlet Portlets sind frei konfigurierbare Ansichten, die sich an beliebigen Stellen der Website hinzufügen lassen.

Folgende Portets werden mit Plone mitgeliefert:

- Calendar portlet
- Classic portlet
- Collection portlet
- Termine
- Login
- Navigation
- Nachrichten
- RSS feed
- Aktuelle Änderungen
- Revisionsliste
- Suche
- Static text portlet

Die Zuweisung kann über folgende Kategorien erfolgen:

Kontextabhängige Portlets `context`

Artikelspezifische Portlets `content_type`

Gruppenportlets `group`

Beachten Sie bitte, dass gruppenspezifische Portlets normalerweise unterhalb von kontextabhängigen Portlets angezeigt werden.

Nutzerportlets `user`

Diese Angabe ist vermutlich nur für die Dashboard-Portlet-Manager sinnvoll.

Portlet Manager Plone wird mit folgenden Portlet Managern ausgeliefert:

`plone.leftcolumn` und `plone.rightcolumn` für die linke und rechte Spalte

`plone.dashboard1` bis `plone.dashboard4` für die vier Spalten des Dashboard.

Python Python ist die Programmiersprache, die von Zope und Plone verwendet wird.

The Python Tutorial http://docs.python.org/tutorial/

Google Python classes http://code.google.com/edu/languages/google-python-class/

Es wird nicht empfohlen, die systemweite Python-Installation zu verwenden da für Plone häufig Python-Pakete benötigt werden, die nicht oder nicht in der gewünschten Version vorliegen.

Wie Python aus den Sourcen installiert werden kann, ist im Kapitel Entwicklungsumgebung beschrieben.

Python Egg Python Eggs sind ein Deploymentformat für Python-Packages. Sie enthalten ein `setup.py`-Skript mit Metainformationen (Lizenz, Abhängigkeiten, etc.) Mit der Python-Bibliothek *Setuptools* können solche Abhängigkeiten automatisch nachgeladen werden, wobei in Eggs spezifische Versionen angegeben werden können.

Python Package Python-Pakete strukturieren den Namensraum von Python-Modulen so, dass sog. *dotted module names* verwendet werden können.

Sehen Sie auch in der Python Dokumentation: Packages

Python Package Index Der Python Package Index PyPI unter `pypi.python.org ist ein Index mit tausenden von Python-Paketen. Setuptools, easy_install und buildout nutzen diesen Index, um Python Eggs automatisch zu installieren.

Er ist momentan noch der Standardhost zum Herunterladen von Paketen. Zukünftig wird pypi.org der Standard-Host werden; momentan ist er jedoch noch nicht voll funktionsfähig.

PYTHONPATH Suchpfad für die Dateien von Modulen eines Python-Interpreters. Das Format entspricht demjenigen von `PATH`. Innerhalb von Python ist der PYTHONPATH mit `sys.path` verfügbar. So kann z.B. beim Aufruf des `bootstrap.py` ein Skript `bin/buildout` erzeugt werden mit folgendem Inhalt:

```
#!/opt/python/2.4.6/bin/python

import sys
sys.path[0:0] = [
  '/opt/plone/3.3/eggs/zc.buildout-1.3.1-py2.4.egg',
  '/opt/plone/3.3/eggs/setuptools-0.6c9-py2.4.egg',
  ]
```

PyUnit Ein Standard-Unit-Testing-Framework für Python.

Repoze Repoze ist eine Sammlung von Technologien um den Webanwendungsserver **`Zope`_** mit WSGI-Anwendungen zu verbinden.

Request Um die Ansicht einer Seite in der Plone-Site zu erhalten, wird ein Request an die Plone-Site gestellt. Diese Anfrage wird in Zope in ein request-Objekt gekapselt, i.a. `REQUEST` genannt (oder `request` in ZPT).

Resource Registries Plone-Infrastruktur, das CSS- und Javascript-Deklarationen in getrennten Dateien erlaubt. Erst für der Auslieferung werden diese Dateien zusammengeschrieben. Auch muss für das Einbinden einer neuen Datei nicht jedesmal in Zope Page Templates geändert werden um die Datei zu importieren oder auf sie zu verweisen. Die *Resource Registries* sind im ZMI zu finden unter `portal_css`, `portal_javascript` und `portal_kss`.

roadrunner roadrunner ist ein Testrunner, der die testgetriebene Entwicklung deutlich beschleunigen kann indem er vorab das Standard-Zope- und Plone-Environment für PloneTestCase läd.

Round-Robin Round-Robin wird bei der Lastverteilung (load balancing) von Varnish verwendet, wobei die Resourcen möglichst gleichmäßig beansprucht werden sollen. Dabei werden die Prozesse in einer Warteschlange verwaltet und der vorderste Prozess erhält eine bestimmte Zeit lang Zugang zu den Resourcen bevor er sich wieder am Ende der Warteschlange einreiht.

Skin Ein Stapel von Layer, die als Suchpfad verwendet werden wenn eine Seite gerendert wird. Skins werden im `portal_skins`-Tool definiert, das auch durch das ZMI erreichbar ist.

Subscriber Subscriber sind eine spezifische Form von Adaptern, die Verwendung findet, wenn alle Adapter eines Objekts zu einem Adapter zusammengefasst werden sollen.

Subscriber lassen sich registrieren mit `registerSubscriptionAdapter`:

```
>>> components.registerSubscriptionAdapter(tests.A1_2)
...
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
Registered event:
SubscriptionRegistration(<Components comps>, [I1], IA2, u'', A1_2, u'')
```

Subscriber können bereitgestellt werden mit `provideSubscriptionAdapter`.

```
>>> component.provideSubscriptionAdapter(SingleLineSummary)
>>> component.provideSubscriptionAdapter(AdequateLength)

>>> doc = Document("A\nDocument", "blah")
>>> [adapter.validate()
...  for adapter in component.subscribers([doc], IValidate)
...  if adapter.validate()]
['Summary should only have one line', 'too short']
```

Mit `subscribers` erhalten Sie die Subscriber der jeweiligen Komponente:

```
>>> doc = Document("A\nDocument", "blah")
>>> [adapter.validate()
...  for adapter in component.subscribers([doc], IValidate)
...  if adapter.validate()]
['Summary should only have one line', 'too short']
```

Der Name und die Factory-Methode eines Subscribers sowie die Angabe, ob der Subscriber erforderlich ist, erhalten Sie mit `provided`, `factory` und `required` aus `registeredSubscriptionAdapters`:

```
>>> for registration in sorted(
...     components.registeredSubscriptionAdapters()):
...     print registration.required
...     print registration.provided, registration.name
...     print registration.factory, registration.info
```

Subscriber lassen sich löschen mit `unregisterSubscriptionAdapter`:

```
>>> components.unregisterSubscriptionAdapter(tests.A1_2)
...
Unregistered event:
SubscriptionRegistration(<Components comps>, [I1], IA2, u'', A1_2, '')
True
```

- Subscription Adapters
- Subscribers.

Supervisor Supervisor ist ein Client/Server-System, das die Prozessüberwachung und -kontrolle auf Unix-Betriebssystemen erlaubt.

Dieses Python-Programm erlaubt `start`, `stop` und `restart` anderer Programme auf UNIX-Systemen wobei es auch abgestürzte Prozesse erneut starten kann.

TAL *TAL* ist eine Template-Sprache, die zur Generierung von XML-Dokumenten verwendet werden kann und abstrahiert dabei völlig von der eingesetzten Programmiersprache. Erst mit TALES, die die Syntax der Ausdrücke von *TAL* bescreibt, werden implementierungsabhängige Ausdrücke in Python erlaubt.

Im Folgenden alle *TAL*-Ausdrücke:

`tal:attributes` erlaubt das dynamische Ändern der Attribute eines Elements.

`tal:define` definiert Variablen.

`tal:condition` testet, ob die angegebenen Bedingungen erfüllt werden.

tal:content ersetzt den Inhalt eines Elements.

tal:omit-tag entfernt ein Element.

tal:on-error beschreibt den Umgang bei einem Fehler.

tal:repeat wiederholt ein Element.

Die tal:repeat-Variable hält folgende Informationen:

index Fortlaufende Zahlen, mit Null beginnend

number Fortlaufende Zahlen, mit Eins beginnend

even wahr für mit even-indexierte Wiederholungen (0, 2, 4, ...)

odd wahr für odd-indexierte Wiederholungen (1, 3, 5, ...)

start wahr für die erste Wiederholung

end wahr für die letzte Wiederholung

first wahr für den ersten Eintrag der Wiederholung

last wahr fpr den letzten Eintrag der Wiederholung

length Länge der Sequenz, d.h. die Gesamtzahl der Einträge einer Wiederholung

letter Position des Eintrags als Kleinbuchstagen: `a-z`, `aa-az`, `ba-bz`, ... `za-zz`, `aaa-aaz` etc.

Letter Position des Eintrags als Versalien

roman Position des Eintrags als römische Zahl in Kleinbuchstaben: `i`, `ii`, `iii`, `iv`, `v` etc.

Roman Position des Eintrags als römische Zahl in Versalien.

tal:replace ersetzt den Inhalt eines Elements.

Erhält ein Element mehrere *TAL*-Anweisungen, so werden diese in folgender Reihenfolge ausgeführt:

1. `tal:define`
2. `tal:condition`
3. `tal:repeat`
4. `tal:content` oder `tal:replace`
5. `tal:attributes`
6. `tal:omit-tag`

TALES *TALES* beschreibt die Syntax der Ausdrücke der Template Attribute Language (TAL) und der Macro Expansion Template Attribute Language (METAL).

TALES stellt mehrere Methoden für Ausdrücke zur Verfügung, die in TAL- und METAL-Attributen durch ein Präfix unterschieden werden können.

path: Der Präfix ist optional, d.h., wird kein Präfix angegeben, so wird ein Pfad-Ausdruck erwartet.

Solche Ausdrücke referenzieren Objekte, um deren Methoden oder Attribute aufzurufen.

string:

Präfix, der beliebige Zeichenketten erlaubt und damit z.B. auch aus Variablen generierte Pfadausdrücke mit ``${...}

odd wahr für odd-indexierte Wiederholungen (1, 3, 5, ...)

start wahr für die erste Wiederholung

end wahr für die letzte Wiederholung

first wahr für den ersten Eintrag der Wiederholung

last wahr fpr den letzten Eintrag der Wiederholung

length Länge der Sequenz, d.h. die Gesamtzahl der Einträge einer Wiederholung

letter Position des Eintrags als Kleinbuchstagen: `a-z`, `aa-az`, `ba-bz`, ... `za-zz`, `aaa-aaz` etc.

Letter Position des Eintrags als Versalien

roman Position des Eintrags als römische Zahl in Kleinbuchstaben: `i`, `ii`, `iii`, `iv`, `v` etc.

Roman Position des Eintrags als römische Zahl in Versalien.

tal:replace ersetzt den Inhalt eines Elements.

Erhält ein Element mehrere *TAL*-Anweisungen, so werden diese in folgender Reihenfolge ausgeführt:

1. `tal:define`
2. `tal:condition`
3. `tal:repeat`
4. `tal:content` oder `tal:replace`
5. `tal:attributes`
6. `tal:omit-tag`

TALES *TALES* beschreibt die Syntax der Ausdrücke der Template Attribute Language (TAL) und der Macro Expansion Template Attribute Language (METAL).

TALES stellt mehrere Methoden für Ausdrücke zur Verfügung, die in TAL- und METAL-Attributen durch ein Präfix unterschieden werden können.

path: Der Präfix ist optional, d.h., wird kein Präfix angegeben, so wird ein Pfad-Ausdruck erwartet.

Solche Ausdrücke referenzieren Objekte, um deren Methoden oder Attribute aufzurufen.

string: Präfix, der beliebige Zeichenketten erlaubt und damit z.B. auch aus Variablen generierte Pfadausdrücke mit `${...}`.

Logische Negation not: Präfix, der den folgenden Ausdruck auswertete und seine logische Negation zurückgibt.

Python python: Präfix, der den Wert des folgenden Python-Skripts ausgibt.

Ein Zugriff dieser Python-Skripte auf sicherheitsrelevante Objekte wird jedoch unterbunden.

Unterdrückung des Quotings structure Ein vorangestelltes `structure` unterdrückt das HTML-Quoting. Damit kann z.B. ein komplettes HTML-Element erzeugt werden.

nothing Einzelner Wert, der von TAL verwendet wird um einen *Nicht-Werte* anzugeben, z.B. `void`, `None`, `Nil`, `NULL`.

default Einzelnder Wert, der in TAL spezifiziert, dass existierender Text nicht ersetzt werden `options` Im Template zulässige Keyword-Argumente

repeat Schleifenvariablen, s.a. The Zope2 Book: Repeat an element

attrs Ein Dictionary, das die zulässigen Werte des aktuellen Tags enthält.

CONTEXTS Liste der Standardnamen. Dies kann verwendet werden um auf eine eingebaute Variable zuzugreifen, die von einer lokalen oder globalen Variable desselben Namens verborgen wird.

- TALES-Spezifikation, Version 1.3
- The Zope2 Book: TALES Overview

Test, funktionaler Test vom Standpunkt eines Endnutzers. Üblicherweise wird ein Use Case oder eine User Story getestet. Ein Beispiel für einen funktionalen Test ist, ob eine bestimmte Nachricht angezeigt wird nachdem ein Formular ohne erforderliche Daten abgeschickt wurde.

Test-Suite Eine Sammlung von Testfällen, die zusammen durchlaufen werden.

Testfall Eine Sammlung von Tests.

Traceback Ein Python Traceback ist eine detaillierte Fehlermeldung, die ausgegeben wird wenn ein Fehler beim Ausführen von Python-Coder auftritt. Um sich einen solchen Traceback anzuschauen, können Sie entweder in die *event log*-Datei in `var/log/instance` schauen oder im ZMI Ihrer Plone-Site in `error_log`. Ein Traceback beginnt mit `Traceback (innermost last):` oder `Traceback (most recent call last):`. Meist ist die bedeutendste Information am Ende eines Tracebacks angegeben.

Unit-Test Ein Test für kleine Code-Einheiten, z.B. das Setzen und Erhalten von Attributen einer Klasse.

Utility *Utilities* sind Komponenten mit einem Interface und die mit einem Interface und einem Namen aufgerufen werden können.

Solche *Utilities* können erstellt werden mit:

```
>>> from zope import interface
>>> class IGreeter(interface.Interface):
...     def greet():
...         "say hello"
>>> class Greeter:
...     interface.implements(IGreeter)
...
...     def __init__(self, other="world"):
...         self.other = other
...
...     def greet(self):
...         print "Hello", self.other
```

queryUtility oder getUtility fragen das Utility nach ihrem Interface:

```
>>> component.queryUtility(IGreeter, 'christian').greet()
Hello chris
>>> component.getUtility(IGreeter, 'christian').greet()
Hello chris
```

`queryUtility` und `getUtility` unterscheiden sich jedoch in ihrer Fehlerbehandlung:

```
>>> component.queryUtility(IGreeter, 'veit')
>>> component.getUtility(IGreeter, 'veit')
...
Traceback (most recent call last):
...
ComponentLookupError: (<InterfaceClass ...IGreeter>, 'veit')
```

provideUtility registriert eine Instanz einer Utility-Klasse, z.B.:

```
>>> from zope import component
>>> greet = Greeter('chris')
>>> component.provideUtility(greet, IGreeter, 'christian')
```

View Ein View ist eine bestimmte Ansicht eines Objektes.

Genauer betrachtet ist ein View eine Funktion zur Berechnung der Darstellung eines Objekts.

Viewlet Ansicht zusätzlicher Informationen, die nicht der Inhalt eines Objekts sind. Dabei werden Viewlets meist durch Viewlet Manager verwaltet. Viewlets und Viewlet Manager ermöglichen die Erstellung von *pluggable user interfaces*.

Viewlet Manager Viewlet Manager verwalten die für sie registrierten Viewlets.

virtualenv virtualenv erlaubt die Erstellung einer virtuellen Python-Umgebung. Damit lassen sich andere Abhängigkeiten, Versionen und Berechtigungen verwenden als in einer systemweiten Installation.

```
$ easy_install-2.7 virtualenv
$ virtualenv my_virtualenv
```

bin Das Verzeichnis enthält die Skripte zum Aktivieren und Deaktivieren des virtualenv, außerdem `easy_install`, `pip` und `python` (Dabei ist `python` eine Kopie desjenigen Python, mit dem das `virtualenv` erstellt wurde.

include Das Verzeichnis enthält nur einen Symlink zum `include`-Verzeichnis derjenigen Python-Installation, aus dem das `virtualenv` erstelt wurde.

lib Das Verzeichnis enthält einen Symlink zum `include`-Verzeichnis derjenigen Python-Installation, aus dem das `virtualenv` erstellt wurde.

Ab Python 2.6 kann ein Nutzer auch einfach seine Python-Umgebung erstellen mit:

```
$ pip install --user foo
```

Siehe auch:

- Compare & Contrast with Alternatives
- PEP 370: Per user site-packages directory
- PEP 370-Documentation

WebDAV WebDAV steht für *Web-based Distributed Authoring and Versioning* und ist eine Erweiterung des Protokolls `HTTP/1.1`, die in RFC 2518 spezifiziert ist. Sie erlaubt, ganze Verzeichnisse zu übertragen. Zudem können Ressourcen bei der Bearbeitung gesperrt werden um ein konkurrierendes Schreiben zu verhindern.

Wie Zope als WebDAV-Server eingerichtet werden kann, ist in WebDAV-Server beschrieben.

Eine Übersicht über diverse WebDAV-Clients finden Sie im Plone-Nutzerhandbuch.

- Dexterity WebDAV notes

Workflow Workflows sind eine einfache Möglichkeit, Geschäftsprozesse abzubilden. Folgende Probleme lassen sich hiermit lösen:

- Artikel können unterschiedliche Zustände (Stadien) annehmen
- Artikel können je nach Stadium unterschiedliche Berechtigungen haben
- Benutzer können bestimmte Ereignisse beim Ändern eines Status auslösen
- Es lassen sich Abnahmen und Übergaben damit realisieren

Home: http://pypi.python.org/pypi/Products.DCWorkflow

WSGI Python-Standard-Interface zwischen Webanendungen mit dem Ziel, die Portabilität von Webanwendungen zu fördern.

WSGI ist in PEP 333 definiert.

XML-RPC XML-RPC ist eine Definition zum Aufruf von Methoden und Funktionen durch entfernte Systeme.

Zope unterstützt XML-RPC für jedes traversierbare Objekt, z.B.:

```
target = 'http://localhost:8080/plone'
path = xmlrpclib.ServerProxy(target).getPhysicalPath()
```

Eine einfache Möglichkeit, einen Nutzer für XML-RPC zu authentifizieren, ist das Einbinden von *HTTP Basic Auth* in eine URL:

```
target = 'http://admin:secret@localhost:8080/plone'
path = xmlrpclib.ServerProxy(target).getPhysicalPath()
```

XML-RPC kann Objekte nicht zuverlässig an andere Aufrufe übergeben. Um an das entfernte Objekt zu gelangen, kann `ZPublisher.Client.Object` verwendet werden.

Sehen Sie auch Zope2.utilities.load_site

wsapi4plone.core stellt zusätzliche Methoden für Zopes XML-PRC-Api zur Verfügung. This is an add-on product exposes more methods available through Zope's XML-RPC api.

Im Folgenden ein Beispiel, wie aus einem `pictures`-Ordner ein Bild mit der ID `portrait.jpg` in einen Ordner `portraits` geladen und in `veit.jpg` umbenannt wird:

```
import os
from xmlrpclib import ServerProxy
from xmlrpclib import Binary

client = ServerProxy("http://admin:secret@localhost:8080/plone")
data = open(os.path.join('pictures', 'portrait.jpg')).read()
myimage = {'portraits/veit.jpg': [{'title': 'a Portrait of Veit', 'image
↪':Binary(data)},'Image']}
output = client.get_object(client.post_object(myimage))
```

Anmerkung: transmogrify.ploneremote nutzt XML-RPC um Inhalte in eine Plone-Site zu importieren.

- XML-RPC How To

XPath XPath ist eine vom W3C entwickelte Abfragesprache, um Teile eines XML-Dokumentes zu adressieren. Auf ihr basieren weitere Standards wie `XSLT`, `XPointer` und `XQuery`.

Ein XPath-Ausdruck setzt sich zusammen aus:

- einem oder mehreren Lokalisierungsschritten,

  sie werden mit dem Zeichen `/` getrennt.

- optional gefolgt von einem oder mehreren Prädikaten.

Lokalisierungsschritte bestehen aus einer Achse und einem Knotentest, die *Achse::Knotentest* geschrieben werden.

Hier die gebräuchlichsten Achsen:

`child` Direkt untergeordneter Knoten `./`

`parent` Direkt übergeordneter Elternknoten `./..`

`self` Der Kontextknoten selbst, der für zusätzliche Bedingungen ausgewählt wird `.`

descendant Untergeordnete Knoten `.//`

attribute Attributknoten `@`

Knotentests schränken die Elementauswahl einer Achse ein:

Elementname Beispiel: `.//Foo` wählt alle Elemente des untergeordneten Knotens mit dem Namen *Foo*.

`*` Auswahl aller Elemente eines Knotens

text(), comment() und processing-instruction() Auswahl von Knoten eines bestimmten Typs

- XML Path Language (XPath) 2.0

ZCatalog ZCatalog ist die Zope Suchmaschine, die die Kategorisierung von und die Suche nach allen Zope-Pbjekten erlaubt. Dabei wird auch die Suche in externen Daten, die z.B. in einer relationalen Datenbank liegen, unterstützt. Darüber hinaus kann der ZCatalog auch zur Erstellung von Sammlungen von Objekten verwendet werden.

Volltextsuche und die gleichzeitige Suche in mehreren Indexen sowie das Gewichten der Felder in den Suchergebnissen werden unterstützt.

Weitere Informationen über den ZCatalog erhalten Sie im **`Zope Book`_**

ZCML XML-Dialekt, der die verschiedenen Zope-Komponenten verbindet.

Von Zope wird initial die in `site-definition` angegebene Datei abgearbeitet, meist `$INSTANCE/etc/site.zcml`. Diese bindet dann über `<includes>`-Tags alle weiteren ZCML-Konfigurationsdateien ein:

```
<!-- Load the meta -->
<include files="package-includes/*-meta.zcml" />
<five:loadProducts file="meta.zcml"/>

<!-- Load the configuration -->
<include files="package-includes/*-configure.zcml" />
<five:loadProducts />

<!-- Load the configuration overrides-->
<includeOverrides files="package-includes/*-overrides.zcml" />
<five:loadProductsOverrides />

<securityPolicy
    component="Products.Five.security.FiveSecurityPolicy" />
```

***-meta.zcml** Diese Dateien gewährleisten, dass die ZCML-Anweisungen der eingebundenen Pakete bei der Abarbeitung der ZCML-Anweisungen vollständig zur Verfügung stehen.

***-configure.zcml** Hiermit werden ZCML-Dateien innerhalb der installierten Pakete abgearbeitet.

***-overrides.zcml** Damit können Konfigurationen von Paketen überschreiben werden.

securitypolicy.zcml Hiermit wird die Security-Policy festgelegt

Das Starten der Instanz bricht ab, wenn in einer `zcml`-Datei `include package` angegeben wird, dieses Paket jedoch nicht installiert ist. Um dies zu vermeiden, kann `include` an die Bedingung geknüpft werden, dass das Paket installiert ist, z.B.:

```
<include
    zcml:condition="installed zope.app.zcmlfiles"
    package="zope.app.zcmlfiles"
    />
<include
    zcml:condition="not-installed zope.app.zcmlfiles"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    package="zope.app"
    />
<include zcml:condition="installed some.package"
    package="some.package" />
<include zcml:condition="not-installed some.package"
    package=".otherpackage" />
```

Es können auch bestimmte Funktionen als Bedingung genannt werden. Diese Bedingungen können mit `have` oder deren Abwesenheit mit `not-have` angegeben werden, z.B.:

```
<include
    package="Products.CMFCore" file="permissions.zcml"
    xmlns:zcml="http://namespaces.zope.org/zcml"
    zcml:condition="have plone-41" />
<configure
    zcml:condition="not-have plone-4">
   <!-- only when the Plone 4 feature has not been provided -->
</configure>
<configure
    zcml:condition="not-have plone-5">
   <!-- only when the Plone 5 feature has not been provided -->
</configure>
```

- Zope Toolkit ZCML Documentation

ZEO Mittels ZEO greift eine Zope-Instanz nicht unmittelbar auf einen Datenspeicher zu sondern per TCP/IP auf einen sog. ZEO-Server. Durch die Verwendung mehrerer Zope-Instanzen, die auf denselben ZEO-Server zugreifen, lässt sich die Last besser verteilen.

Weitere Informationen zu ZEO erhalten Sie im **`Zope Book`_**

ZMI Erlaubt die Verwaltung des Zope-Servers durch das Web.

Das *Zope Management Interface* lässt sich für die meisten vom Zope-Server ausgelieferten Objekte anzeigen, indem an die URL `manage_workspace` angehängt wird.

ZODB Die ZODB bietet eine einfache Persistenz für Python-Objekte. Sie wird von Zope verwendet um Inhalte, Skripte und Konfigurationen zu speichern.

Das ZMI ist ein Web-Interface zum Verwalten der Inhalte der ZODB.

Zum Weiterlesen:

- Welcome to the ZODB Book
- ZODB tutorial
- ZODB/ZEO programming guide
- ZODB articles

Zope **`Zope`_** ist ein objektorientierter, in der Programmiersprache Python geschriebener, freier Webanwendungsserver.

Zope Component Architecture *ZCA* ist ein Python-Framework zur einfachen Erstellung eines komponentenbasierten Designs. Dabei sind Komponenten wiederverwendbare Objekte mit einem Interface, das beschreibt, wie dieses Objekt angesprochen werden kann. *ZCA* vereinfacht die Erstellung von zwei Basiskomponenten:

Adapter Komponenten, die aus anderen Komponenten erstellt werden um sie einem bestimmten Interface zur Verfügung zu stellen. Dabei sind Subscribers und Handlers zwei spezielle Typen von Adaptern.

Utilities Komponenten, die ein Interface anbieten und von einem Interface und einem Namen aufgerufen werden.

Zur *ZCA* gehören im wesentlichen drei Pakete:

zope.interface wird verwendet um die Interfaces einer Komponente zu definieren.

zope.event bietet ein einfaches Event-System, siehe Event.

zope.component erleichtert die Erstellung, Registrierung und Retrieval der Komponenten.

Die Installation beider Paktete kann einfach erfolgen mit easy_install:

```
$ easy_install zope.component
```

ZopeSkel ZopeSkel ist eine Sammlung von Vorlagen, mit denen sich schnell Zope-Projekte erstellen lassen.

Um solche Projekte zu erstellen verwendet ZopeSkel intern die Paste Script-Bibliothek.

Weitere Informationen zur Installation, den Vorlagen und Standardeinstellungen erhalten Sie in ZopeSkel..

ZPT *Zope Page Templates* ist eine Template-Sprache um XML-konforme Dokumente zu erstellen. Sie ist nahezu vollständig in TAL, TALES und METAL beschrieben.

Darüberhinaus hat *ZPT* jedoch einige zusätzliche Funktionen: Wird als Content-Type `text/html` angegeben, müssen die Namensräume für TAL und METAL nicht angegeben werden. Auch werden HTML-Dokumente mit dem non-XML-Parser analysiert, der nachlässiger mit fehlerhaftem Markup umgeht.

- ZPT-specific Behaviors

Stichwortverzeichnis

S

T

U

V

W

X

Z