

Namespace Packages

Martin v. Löwis

Übersicht

- Package: hier "Einheit der Code-Strukturierung in Python; Zusammenfassung von Modulen"
- nicht: "separat installierbare Software"
 - "distribution" (PJE), "project"
- Problem
- PEP 382
- PEP 402

Problem

- Python package == Verzeichnis mit `__init__.py`
- ggf. weitere Module (`foo.py`) und Unter-Packages
- Manche Packages enthalten separat entwickelte und installierbare Module / Unterpackages
 - zope: `zope.interface`, `zope.tal`, `zope.paste`
 - `zope.app.{annotation,file,folder,ftp,...}`
 - `tipfy`, `tipfy.ext.{acl,auth/blobstore,db,...}`

Problem (2)

- separate Installation der Unterpakete erwünscht
- Lösung: Installation einfach in das Toplevel-Verzeichnis (z.B. `zope.app`) hinein
- aber: wem gehört `zope/app/__init__.py`?
- aber: Schreiben in gemeinsames Verzeichnis nicht wünschenswert / erlaubt
- eggs, private Erweiterung von Paketen in `site-packages`, ...

setuptools

- Namespace Packages
- Packages, die *ausschließlich* Unterpakete / Module enthalten
- also: keinen eigenen Code (`__init__.py`)
- Hack: setuptools generiert `__init__.py`, das `__path__` manipuliert

__path__

- Attribut eines Packages (zusätzlich zu `__file__` und `__name__`)
- i.d.R. nur das Package-Verzeichnis selbst
- aber: Suchpfad für Package analog zu `sys.path` "schon immer" unterstützt
 - `pkgutil.extend_path`

setuptools-Probleme

- kein "Standard"-Verfahren
- pkgutil Teil der Standardbibliothek
- aber: namespace-Deklaration in setup.py spezifisch für setuptools
- Hack: jedes Teilprojekt muss `__init__.py` ausliefern; nur das erste wird benutzt
- Debian (u.a.): kollidierende `__init__.py`, wenn Untermodule doch in ein Verzeichnis installiert werden

Terminologie

- "Namespace Packages" m.E. verwirrend
- jedes Packages ist ein Namespace
- m.E. geht es um "partial packages"
- oder "package portion"
- oder "package component"
- ähnlich zu "partial classes" in C#

PEP 382

PEP 382

- Ziel: "echte" Unterstützung von Namespace Packages
- `__init__.py` soll weiterhin als Package-Initialisierung verwendbar bleiben
- Frage: woran erkennt man, dass ein Verzeichnis Teil eines Package ist?
 - es kann nur ein `__init__.py` geben

PEP 382 (2)

- Partial Package deklariert durch .pyp-Datei in Verzeichnis, alternativ zu `__init__.py`
- Name der .pyp-Datei irrelevant, sollte nicht mit anderen .pyp-Dateien kollidieren
- Installation in ein Verzeichnis erzeugt keine kollidierenden Files
- Inhalt der Datei irrelevant

PEP 382 (3)

- Auswirkungen auf import
- Suche sucht gesamten `__path__` des Eltern-Package (oder `sys.path`) ab nach `__init__.py` oder `.pyp`.
- ggf. müssen Finder (PEP 302) angepasst werden, um nach `.pyp`-Files zu suchen.

PEP 382 (4)

- Diskussion (PJE):
 - warum sind .pyp-Files nötig?
 - wie ist das gleiche Problem in Java, Perl, PHP gelöst?
 - geht es nicht auch noch anders?

PEP 402

PEP 402

- Phillip Eby
- Jedes Verzeichnis mit dem Namen des Package gehört potentiell zum Package
- `__path__` wird dynamisch beim ersten Zugriff aufgebaut

PEP 402 (2)

- Fall 1: reguläres Modul wird importiert
 - "import csv" lädt csv.py
 - `__path__` ist nicht gesetzt
- Folgende Imports von Untermodulen erzeugen `__path__`
 - "import csv.excel" sucht nach Verzeichnissen "csv/"
 - berechnet `__path__`
 - lädt "csv/excel.py"
- Entlehnt von Perl

PEP 402 (3)

- Fall 2: es gibt kein Modul für das Package selbst
- "import zope.interface" findet allerlei Verzeichnisse namens zope, merkt sie sich
- kein zope/ __init__.py und kein zope.py wird gefunden
- ein leeres Package wird synthetisiert, und __path__ auf die Liste von Verzeichnissen gesetzt
- Verzeichnisse werden nun nach "interface/ __init__.py", "interface.py", ... durchsucht

PEP 402 (4)

- Diskussion:
 - gute Rückwärtskompatibilität (Suche bricht weiterhin bei erstem `__init__.py` ab, nicht-Package-Verzeichnisse werden ignoriert)
 - offene Fragen: woran erkennt man, dass ein Untermodul gesucht wird ("from zope import interface")?
 - evtl: dynamischer `__path__`-Aufbau für alle Packages und Module

Status

- Eric Smith ist als PEP-Zar eingesetzt
- Implementierung für Python 3.3
definitives Ziel
- offene Punkte:
 - distutils-etc-Unterstützung
 - Rückportierung auf alte Python-Versionen (in distribute / distutils2)

Fragen?

