Das Zope-Buch

Behandelt auch Zope 2.5

Original: Amos Latteier und Michel Pelletier

Übersetzung: Lucia Cosima Eiselstein, Egon Frerich, Sascha Gresk, Reinhard Holler, Jörg Jenetzky, Eckhard Jost, Stefan Mahlitz, Rudolf Kautz, Marcus Schopen, Uwe Schuerkamp, Erich Seifert, Stefanie Winkelmann und Gabriele Zöttl

Lektorat: Silke Ebel, Rudolf Kautz und Henriette Krech

Übersetzungskoordination: Erich Seifert

Inhaltsverzeichnis

Einleitung

Willkommen beim *Zope-Buch*. Dieses Buch ist dafür gedacht, Ihnen Zope und seine Verwendungsmöglichkeiten vorzustellen. Zope ist ein Open-Source-Web-Anwendungsserver. Wenn Sie sich für das Verfassen von Web-Seiten, das Programmieren von Web-Skripten, die Verwendung von Datenbanken, das Verwalten von Inhalten oder die Durchführung einer gemeinsamen Web-Entwicklungsaufgabe interessieren, dann sollten Sie dieses Buch lesen.

Warum sollte ich dieses Buch lesen?

Dieses Buch soll sowohl jetzige und neue Zope-Benutzer ansprechen:

- Sie müssen kein Programmierer sein, um dieses Buch zu lesen oder Zope zu verwenden.
- Sie sollten eine Vorstellung davon haben, wie das Web funktioniert; mit Grundkenntnissen über HTML und URLs.
- Sie sollten wissen, was ein Web-Browser und ein Web-Server ist und Sie sollten eine Vorstellung darüber haben, wie diese miteinander kommunizieren.

Der erste Teil des Buchs erklärt Ihnen, wie Sie Zope über seine Web-Verwaltungsschnittstelle, dem Web Managment Interface, benutzen, um dynamischen Inhalt zu verwalten. Die Konzepte in diesen Kapiteln sind grundlegende Zope-Konzepte, die Ihnen zeigen, wie man Zope verwendet, um Inhalt im Netz zu veröffentlichen.

Einige spätere Abschnitte des Buches behandeln weiterführende Themen wie relationale Datenbanken, das Skripting mit verschiedenen Programmiersprachen und XML. Diese Kapitel vermitteln Ihnen keine Grundlagen über relationale Datenbanken, wie sie Programmieren lernen oder über XML, sie zeigen Ihnen lediglich, wie diese Technologien mit Zope zu verwenden sind.

Wie das Buch aufgebaut ist

Im Folgenden sehen Sie den Aufbau dieses Buches sowie eine kurze Zusammenfassung jedes Kapitels.

Teil I: Einführung in Zope

Diese Kapitel helfen dem Leser beim Start und Betrieb von Zope. Sie erfahren etwas über die Grundobjekte in Zope und dessen Vokabular.

Kapitel 1: Einführung in Zope

Kapitel 1 erklärt Ihnen, was Zope ist und für wen es gedacht ist. Es beschreibt in groben Zügen, was Sie mit Zope tun können. Sie lernen auch etwas über die Unterschiede zwischen Zope und anderen Web-Anwendungs-Servern.

Kapitel 2: Zope benutzen

Kapitel 2 behandelt die wichtigsten Zope-Konzepte. Am Ende dieses Kapitels sollten Sie in der Lage sein, Zope zu verwenden, um einfache aber leistungsfähige Web-Anwendungen zu erstellen und zu verwalten.

Kapitel 3: Verwendung grundlegender Zope-Objekte

Kapitel 3 führt *Objekte* ein. Sie sind die wichtigsten Elemente von Zope. Hier erfahren wir, was ein Objekt im Allgemeinen ist und führen dann die Grundobjekte von Zope ein: Ordner (Folders), DTML-Dokumente, DTML-Methoden, Dateien (Files) und Bilder (Images).

Kapitel 4: Dynamische Inhalte mit DTML

Kapitel 4 stellt *DTML* vor, Zopes Tag-basierte Skripting-Sprache. Hier beschreiben wir die Verwendung von DTML für das Erstellen von Vorlagen und für das Skripting und seine Rolle in bezug auf andere Arten, Zope zu programmieren. Wir behandeln die DTML-Syntax und die drei grundlegendsten Befehle *var*, *if* and *in*. Nach dem Lesen dieses Kapitels werden Sie in der Lage sein, dynamische Web-Seiten zu erstellen.

Kapitel 5: Zopes Seitenvorlagen verwenden

Kapitel 5 führt Zopes Seitenvorlagen -- die Zope Page Templates -- ein, ein neues Werkzeug um dynamisches HTML zu erzeugen. Dieses Kapitel zeigt Ihnen, wie man Seitenvorlagen erstellt und bearbeitet. Es führt auch grundlegende Vorlagenanweisungen ein, die es Ihnen ermöglichen dynamischen Inhalt einzufügen.

Kapitel 6: Erstellung einfacher Zope-Anwendungen

Kapitel 6 führt den Leser durch mehrere Alltagsbeispiele, in denen Zope-Anwendungen aufgebaut werden. Es liefert viele Beispiele, die zeigen, wie man Zope-Objekte verwenden kann und wie sie zusammenarbeiten können, um einfache Anwendungen zu erzeugen.

Teil II: Erstellen von Web-Anwendungen mit Zope

Diese Kapitel liefern einen tiefergehenden Einblick in weiterführende Zope-Themen. Sie liefern die Grundlagen, die notwendig sind, um richtige Web-Anwendungen mit Zope aufzubauen.

Kapitel 7: Benutzer und Sicherheit

Kapitel 7 betrachtet, wie Zope Benutzer, Authentifizierung, Berechtigungen und andere sicherheitsbedingte Angelegenheiten behandelt. Sicherheit spielt in Zopes Design eine zentrale Rolle und das sollte sie auch in den Web-Anwendungen, die Sie mit Zope erstellen.

Kapitel 8: Variablen und DTML für Fortgeschrittene

Kapitel 8 wirft einen näheren Blick auf DTML. Es behandelt Sicherheit in DTML und das komplizierte Thema, wie Variablen in DTML gesucht werden. Es deckt auch fortgeschrittene Anwendungen der in Kapitel 3 besprochenen Grundbefehle und der unzähligen Befehle für Spezialzwecke ab. Dieses Kapitel macht Sie zu einem DTML-"Zauberer".

Kapitel 9: Zopes Seitenvorlagen für Fortgeschrittene

Kapitel 9 gibt einen tiefergehenden Einblick in Vorlagen. Dieses Kapitel bringt Ihnen alle Vorlagenanweisungen und Ausdruckstypen bei. Es behandelt auch Makros, die es Ihnen erlauben Darstellungselemente wiederzuverwenden. Am Ende dieses Kapitels werden Sie alles über Vorlagen wissen.

Kapitel 10: Zope-Scripting für Fortgeschrittene

Kapitel 10 handelt vom Zope-Skripting mit Python und Perl. Darin zeigen wir, wie man Geschäftslogik in Zope mit Hilfe von leistungsfähigeren Werkzeugen als DTML schreiben kann. Es erörtert die Vorstellung von *Skripten* (Scripts) in Zope und konzentriert sich dabei auf python- und perl-basieriende Skripten. Dieses Kapitel zeigt Ihnen, wie Sie Ihrer Site leistungsfähiges Skripting hinzufügen können.

Kapitel 11: Inhalt durchsuchen und kategorisieren

Kapitel 11 zeigt Ihnen, wie Objekte mit Zopes eingebauter Suchmaschine, dem *Catalog*, indiziert und durchsucht werden können. Es führt Indizierungskonzepte ein und erörtert verschiedene Vorgehensweisen für das Indizieren und Suchen. Zuletzt erörtert es Metadaten und Suchergebnisse. Dieses Kapitel zeigt Ihnen, wie Sie eine leistungsfähige und einfach zu benutzende Informationsarchitektur aufbauen können.

Kapitel 12: Verbindung mit relationalen Datenbanken

Kapitel 12 beschreibt, wie sich Zope mit externen relationalen Datenbanken verbinden lässt. Es zeigt Ihnen, wie Datenbanken anzubinden und abzufragen sind. Es behandelt auch Funktionen, die Ihnen erlauben relationale Daten so zu behandeln, als ob sie Zope-Objekte wären. Zuletzt erörtert das Kapitel Überlegungen zu Sicherheit und Leistungsfähigkeit.

Teil III: Entwickeln fortgeschrittener Web-Anwendungen mit Zope

Der letzte Teil des Buches behandelt fortgeschrittene Themen. Sie erfahren, wie Sie Ihre Web-Anwendung je nach Ihren Bedürfnissen skalieren und Zope selbst erweitern können.

Kapitel 13: Skalierbarkeit und ZEO

Kapitel 13 behandelt Probleme und Lösungen beim Aufbau und bei der Verwaltung großer Web-Anwendungen und geht dabei auf Themen der Verwaltung und Skalierbarkeit ein. Vor allem wird die "Zope Enterprise Option" (ZEO) im Detail behandelt. Dieses Kapitel zeigt Ihnen die Werkzeuge und Methoden auf, die Sie brauchen, um eine kleine Site zu einer großformatigen Site zu machen und Millionen von Besuchern zu bedienen.

Kapitel 14: Zope erweitern

Kapitel handelt von den Möglichkeiten Zope zu erweitern, indem Sie Ihre eigenen Objektarten erstellen. Es erörtert *ZClasses*, und wie Instanzen von Klassen erzeugt werden. Es beschreibt Schritt für Schritt wie man eine ZClass erstellt und die dabei aufkommenden Sicherheits- und die Designaspekte. Zuletzt zeigt es, wie man Basisklassen in Python für ZClasses erstellt, und beschreibt die grundlegenden Klassen, die mit Zope geliefert werden. Dieses Kapitel zeigt Ihnen, wie Zope auf eine höhere Ebene gebracht werden kann, indem Sie Zope Ihren Bedürnissen entsprechend maßschneidern.

In diesem Buch verwendete Konventionen

Dieses Buch verwendet die folgenden typographischen Konventionen:

Kursivschrift
: Kursivschrift zeigt Variablen an und wird auch verwendet, um neue Begriffe einzuführen.

`Dicktengleiche Schrift`
: Text in dicktengleicher Schrift zeigt Eingabekommandos, Hyperlinks und Code-Listings an.

Zopebuch: Inhaltsverzeichnis

Kapitel 2: Zope verwenden

Dieses Kapitel hilft Ihnen beim Start und Betrieb von Zope. Es führt Sie durch die Installation und die Inbetriebnahme von Zope. Dieses Kapitel behandelt die wichtigsten Zope-Konzepte. Am Ende dieses Kapitels sollten Sie in der Lage sein, Zope zu verwenden, um einfache aber leistungsfähige Web-Anwendungen zu erstellen und zu verwalten.

Zope herunterladen

Die ersten Schritte um Zope verwenden zu können, sind das Herunterladen und Installieren. Zope ist frei verfügbar auf der Zope.org-Web-Site. Die neueste stabile Version ist im Download-Bereich von Zope.org stets verfügbar.

Zope ist zur Zeit als ausführbares Programm für Windows, Linux und Solaris verfügbar. Dies bedeutet, dass Sie es einfach herunterladen und installieren können, ohne Programme kompilieren zu müssen. Für andere Plattformen müssen Sie den Quellcode herunterladen und Zope kompilieren. Zope kann auf fast jedem Unix-ähnlichen Betriebssystem kompiliert und ausgeführt werden. Als eine allgemeine Faustregel gilt: Wenn Python für Ihr Betriebssystem

verfügbar ist und Sie einen C Compiler haben, dann können Sie Zope wahrscheinlich verwenden.

Zope installieren

Sie werden Zope unterschiedlich, jeweils abhängig von Ihrer Plattform installieren. Wenn Sie eine aktuelle Version von Linux verwenden, kann es sein, dass Zope schon installiert ist. Sie können Zope sowohl in Binär- als auch in Quelltext-Form bekommen. Es sind auch mehrere verschiedene binäre Formate verfügbar.

Installieren von Zope unter Windows

Zope für Windows kommt als selbstinstallierende *.exe*-Datei. Um Zope zu installieren, doppelklicken Sie auf das Installationsprogramm. Das Installationsprogramm führt Sie durch den Installationsprozess. Wählen Sie einen Namen für Ihre Zope-Installation aus und ein Verzeichnis, in das diese installiert werden soll. Klicken Sie auf *Next* und erstellen Sie ein neues Zope-Benutzerkonto. Dieses Konto wird *Initial User* (Anfangsbenutzer, Anm. d. Üb.) genannt. Dies erstellt ein Konto, das Sie verwenden können, um sich in Zope zum ersten Mal anzumelden. Wenn Sie wollen können Sie diesen Benutzernamen und das Kennwort später ändern.

Wenn Sie Windows NT oder Windows 2000 verwenden, können Sie Zope als Dienst ausführen. Zope als Dienst auszuführen ist eine gute Idee für einen öffentlichen Server. Wenn Sie Zope nur für den persönlichen Gebrauch brauchen Sie es nicht als Dienst laufen zu lassen. Bedenken Sie, dass, wenn Sie Windows 95, Windows 98 oder Windows ME (Millenium Edition) verwenden, Sie Zope nicht als Dienst ausführen können.

Wenn Sie beschließen, Zope später zu deinstallieren, können Sie dazu das *Unwise.exe*-Programm in Ihrem Zope-Verzeichnis verwenden.

Herunterladen von Linux- und Solaris-Binaries

Laden Sie die Binärdaten für Ihre Plattform herunter und entpacken das tarball-Archiv:

```
$ tar xvfz Zope-2.4.0-linux2-x86.tgz
```

In diesem Beispiel laden Sie die Version 2.4.0 herunter. Dies ist vielleicht nicht die neueste Version von Zope, wenn Sie das hier lesen, vergewissern Sie sich also, dass Sie die neueste *stabile* Version von Zope für Ihre Plattform bekommen.

Dies entpackt Zope in ein neues Verzeichnis. Geben Sie das Zope-Verzeichnis ein und führen Sie das Zope-Installationsskript aus:

```
$ cd Zope-2.4.0-linux2-x86
$ ./install
```

Das Installationsprogramm gibt während der Zope-Installtion Informationen aus. Unter anderem erstellt es ein initiales Benutzerkonto. Sie können den Namen des Anfangsbenutzers und dessen Kennwort später mit dem *zpasswd.py*-Skript wechseln (siehe Kapitel 7, "Benutzer und Sicherheit").

Das Installationsprogramm konfiguriert Zope so, dass es unter Ihrer UNIX-Benutzer-ID läuft. Wenn Zope lieber unter einer anderen Benutzer-ID auszuführen möchten, können Sie die Befehlszeilenoption -u benutzen, um den Benutzer anzugeben unter dessen Kennung Sie Zope laufen lassen wollen. Es gibt viele Bücher darüber mit weiteren Informationen über Benutzer-IDs und der UNIX-Verwaltung im Allgemeinen, die Sie sich ansehen sollten, wenn Sie irgend etwas besonderes machen möchten. Bis hierhin funktioniert alles tadellos, wenn Sie Zope nur mit Ihrem Benutzer ohne zusätzliche Befehlszeilenoptionen installieren.

Bezüglich weiterer Informationen über das Installieren von Zope sehen Sie in den Installationsanweisungen unter *doc/INSTALL.txt* nach und finden Sie mehr über das Installationsprogramm heraus, indem Sie es mit der Hilfsoption -h ausführen:

```
$ ./install -h
```

Zope im *RPM*- und *deb*-Format bekommen

Die Zope Corporation stellt Zope nicht im RPM-Format zur Verfügung, aber andere Leute tun das. Jeff Rush verpackt Zope regelmäßig als RPM. Für weitere Informationen sehen Sie auf seine Web-Seite (http://www.taupro.com/Downloads/Zope/). Zope ist auch im *deb*-Packetformat von Debian-Linux verfügbar. Zope-deb-Packete können Sie auf der Debian-Web-Site finden. Im Allgemeinen finden sich die neuesten Zope-Versionen in der *instabilen* Distribution.

Kompilieren von Zope aus dem Quelltext

Wenn keine Binärdateien für Ihre Plattform verfügbar sind, dann haben Sie vielleicht eine Chance Zope aus dem Quelltext zu kompilieren. Um dies zu tun, installieren Sie Python aus dem Quellcode für Ihre Plattform und vergewissern Sie sich, dass Sie einen C-Compiler haben. Sie können Python von der Python.org-Web-Site bekommen. Obwohl wir versuchen die neueste Python-Version für Zope zu verwenden, ist die aktuellste Python-Version oft neuer als die Version, die wir "offiziell" in Zope unterstützen. Für Information darüber welche Version von Python Sie brauchen um Zope mit, zu kompilieren, sehen Sie in den Release Notes für jede Version auf der Web-Seite nach. Zope 2.4 benötigt Python 2.1. Zope 2.3 und ältere Versionen verwendeten Python 1.5.2.

Laden Sie die Zope-Quelltext-Version herunter und extrahieren Sie das tarball-Archiv:

```
$ tar xvfz Zope-2.4.0-src.tgz
```

Dies entpackt Zope in ein neues Verzeichnis. Geben Sie das Zope-Verzeichnis ein und führen Sie das Zope-Installationsskript aus:

```
$ cd Zope-2.4.0-src
$ python wo_pcgi.py
```

Das Installationsprogramm kompiliert Zope und erstellt Ihre Installation. Das Installationsprogramm gibt während es läuft Informationen aus, einschließlich des Namens des Anfangsbenutzers und dessen Kennwort. Es ist wichtig, diese Informationen *aufzuschreiben*, um sich in Zope anzumelden. Bezüglich weiterer Informationen sehen Sie in den Installationsanweisungen unter *doc/INSTALL.txt* nach. Sie können das Konto des Anfangsbenutzers später mit dem *zpasswd.py*-Skript ändern (siehe Kapitel 7, "Benutzer und Sicherheit").

Zope starten

Abhängig von Ihrer Plattform führen Sie Zope mit verschiedenen Befehlen aus. Egal welche Plattform Sie verwenden, Sie können Zope entweder manuell oder automatisch ausführen. Wenn Sie Zope manuell ausführen, sagen Sie Zope einfach, wann es starten und wann es beenden soll. Wenn Sie Zope automatisch ausführen, wird es gestartet und beendet, wenn Ihr Computer gestartet und beendet wird.

Starten von Zope unter Windows

Das Installationsprogramm erstellt ein Zope-Verzeichnis mit einer Stapeldatei namens *start.bat*. Doppelklicken Sie auf das *start.bat*-Symbol. Dies öffnet ein Fenster, das Protokoll-Information beinhaltet. Mit diesem Fenster finden Sie heraus, auf welchem Port Zope läuft. Sie können sich jetzt in Zope mit einem Web-Browser anmelden.

Wenn Sie Zope als Dienst ausführen, können Sie Zope über den Systemsteuerungspunkt "Dienste" starten und beenden. Zope wird Ereignisse in das Ereignisprotokoll schreiben, sodass Sie das Starten und Beenden Ihres Dienstes nachvollziehen können. Wenn Sie Zope als Dienst ausführen, müssen Sie wissen, auf welchem Port Zope läuft, da Sie keinen direkten Zugriff auf seine detaillierten Protokollinformationen haben werden.

Zope bringt seinen eigenen Web-Server mit. Wenn Sie Zope starten, startet auch der Web-Server. Wenn Sie möchten, können Sie Zope an Ihren vorhandenen Web-Server, wie etwa IIS, anbinden. Dies übersteigt jedoch den Rahmen dieses Buches. Der Zope Administrator's Guide deckt diesen Themenbereich ab.

Starten von Zope unter Unix

Führen Sie das *start*-Skript aus:

```
$ ./start &
```

Zope fängt an zu starten und gibt Protokoll-Informationen auf der Konsole aus. Sie sollten nun Informationen darüber sehen, auf welchem Port Zope läuft. Sie können sich jetzt in Zope mit einem Web-Browser anmelden.

Zope bringt seinen eigenen Web-Server mit. Wenn Sie Zope starten, startet auch der Web-Server. Wenn Sie möchten, können Sie Zope an Ihren vorhandenen Web-Server, wie etwa Apache, anbinden. Dies übersteigt jedoch den Rahmen dieses Buches. Der Zope Administrator's Guide deckt diesen Themenbereich ab.

Das *start*-Skript kann auch bearbeitet werden, um Zope mit vielen verschiedenen Optionen zu starten. Wie man den Zope-Systemstart anpasst, wird ebenfalls im "Administrator's Guide" beschrieben.

Das Anmelden

Um sich in Zope anzumelden brauchen Sie einen Web-Browser. Zopes Benutzerschnittstelle ist komplett in HTML geschrieben. Jeder Browser, der modernes HTML versteht, funktioniert deshalb. Mozilla und jede Version 3.0 oder höher des Microsoft Internet Explorer oder des Netscape Navigator kann verwendet werden.

Um sich in der Management-Oberfläche anzumelden, geben Sie in Ihren Web-Browser den Management-URL von Zope ein. Der Management-URL für Zope ist Zopes Basis-URL mit angehängtem */manage*. Angenommen Sie haben Zope auf Ihrem lokalen Rechner installiert haben, sodass er auf dem Standard-Port 8080 läuft, ist der Management-URL:

```
http://localhost:8080/manage
```

Dieser URL funktioniert normalerweise, es kann jedoch sein, dass sie auf einem anderen Rechner als dem hier gezeigten anmelden müssen. Um herauszufinden, welchen URL Sie genau verwenden müssen, sehen Sie sich die Protokoll-Information an, die Zope ausgibt sobald es startet. Zum Beispiel:

```
      ------
      2000-08-07T23:00:53 INFO(0) ZServer Medusa (V1.18) started at Mon Aug
7 16:00:53 2000
              Hostname: erdnuss
              Port:8080

      ------
      2000-08-07T23:00:53 INFO(0) ZServer FTP server started at Mon Aug  7
16:00:53 2000
              Authorizer:None
              Hostname: erdnuss
              Port: 8021
      ------
      2000-08-07T23:00:53 INFO(0) ZServer Monitor Server (V1.9) started on
port 8099
```

Der erste Protokolleintrag zeigt, dass Zope auf einer Maschine mit dem Namen *erdnuss* läuft und dass der Web-Server auf Port 8080 läuft. Dies bedeutet, dass der Management-URL *http://erdnuss:8080/manage* ist. Später im Buch sehen wir uns die anderen Server an, auf die in der Protokoll-Information verwiesen wird.

Nachdem Sie Zopes Management-URL in Ihren Browser eingeben haben, fordert Ihr Browser Sie dazu auf, für Zope einen Benutzernamen und ein Kennwort anzugeben. Geben Sie den Namen des Anfangsbenutzers und dessen Kennwort ein, die während des Installationsprozesses erstellt wurden. Wenn Sie den Namen und das Kennwort des Anfangsbenutzers nicht kennen, dann fahren Sie Zope durch Schließen seines Fensters herunter. Ändern Sie das Kennwort des Anfangsbenutzers mit dem zpasswd.py-Script und starten Sie Zope neu. Siehe Kapitel 7, "Benutzer und Sicherheit" bezüglich weiterer Information über das Konfigurieren des Anfangsbenutzer-Kontos.

Zope mit dem Management-Interface steuern

Nachdem Sie sich erfolgreich angemeldet haben, sehen Sie eine Web-Seite des Zope-Management-Interfaces, wie sie Abbildung 2-1 zeigt.

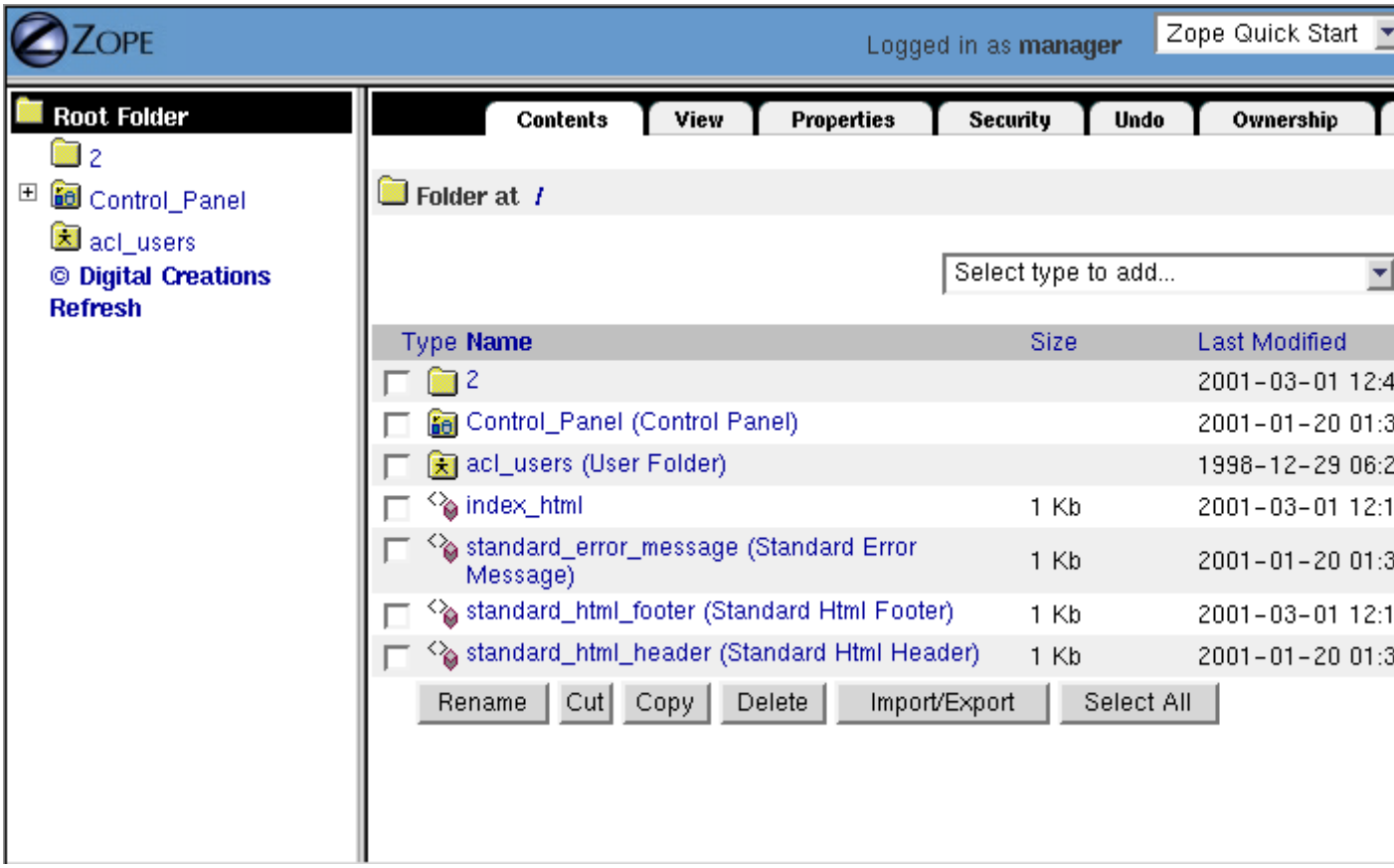


Abbildung 2-1 Das Zope-Management-Interface.

Das Zope Management-Interface erlaubt Ihnen, Zope innerhalb Ihres Web-Browsers zu steuern.

Verwenden des Navigators

Das Zope-Management-Interface ist in drei Rahmen aufgeteilt. Mit dem linken Rahmen navigieren Sie in Zope, wie Sie auch in einem Dateisystem mit einem Dateimanager wie Windows Explorer navigieren würden. Dieser Rahmen wird *Navigator* genannt und wird im

linken Rahmen von Abbildung 2-1 gezeigt. In diesem Rahmen Sie sehen den Stammordner und alle seine Unterordner. Der Stammordner ist in der oberen linken Ecke der Baumstruktur. Der Stammordner ist das "Haupt" von Zope. Alles in Zope befindet sich im Stammordner.

Manche der Ordner haben Pluszeichen links von ihnen. Mit diesen Pluszeichen können Sie die Ordner öffnen, um die darin enthaltenen Unterordner zu sehen.

Über der Ordnerstruktur zeigt Ihnen Zope Anmeldeinformation an. In diesem Bildschirmfoto können Sie sehen, dass Sie gegenwärtig als "manager" angemeldet sind. Wenn Sie sich in Zope anmelden, werden Sie das Anfangsbenutzerkonto verwenden und nur den Namen dieses Kontos statt "manager" sehen.

Um einen Ordner zu verwalten, klicken Sie auf ihn und er wird im rechten Rahmen des Browserfensters erscheinen. Dieser Rahmen heißt *Arbeitsbereich*.

Verwenden des Arbeitsbereichs

Der rechte Rahmen des Management-Interfaces zeigt gerade zu verwaltende Objekt an. Wenn Sie sich das erste Mal in Zope anmelden, ist das aktuelle Objekt der Stammordner, wie im rechten Frame von Abbildung 2-1 gezeigt. Der Arbeitsbereich gibt Ihnen Information über das aktuelle Objekt und erlaubt Ihnen es zu ändern.

Entlang des oberen Bildschirmbereichs befinden sich einige Reiter. Der derzeit ausgewählte Reiter ist in einer helleren Farbe hervorgehoben. Jeder Reiter bringt Sie zu einer anderen *Ansicht* des aktuellen Objekts. In jede Ansicht lässt Sie eine andere Verwaltungsfunktion des Objekts durchführen.

In Abbildung 2-1 sehen Sie die *Contents*-Ansicht des Stammordner-Objekts root.

Im oberen Bereich des Arbeitsbereichs, direkt unterhalb der Reiter, ist eine Beschreibung des Typs und des URLs des aktuellen Objekts. Auf der linken Seite ist ein Symbol, das den Typ des aktuellen Objekts darstellt und rechts daneben ist der URL des Objekts.

In Abbildung 2-1 sagt Ihnen "Folder at /", dass das aktuelle Objekt ein Ordner ist und dass sein URL / ist. Beachten Sie, dass dieser URL der Objekt-URL relativ zu Zopes Basis-URL ist. Wenn also der URL oder Ihre Zope-Site *http://meinesite.beispiel.de:8080* ist, dann wäre der URL von "Folder at /myFolder" *http://meinesite.beispiel.de:8080/myFolder*.

Beim Erkunden der verschiedenen Zope-Objekte werden Sie feststellen, dass Sie die URLs (wie auf dem Management-Bildschirm gezeigt), verwendet werden kann, um in den Objekten zu navigieren.

Wenn Sie zum Beispiel einen Ordner unter */Zoo/Reptilien/Schlangen* verwalten, können Sie zum Ordner unter */Zoo* durch Klicken auf das Wort *Zoo* im URL des Ordners zurückkehren.

Der letzte Rahmen ist der oberste Rahmen in Zope. Dieser enthält eine Auswahlliste, die Sie folgendes wählen lässt:

Preferences (Voreinstellungen, Anm. d. Üb.)
: Hier können Sie Standardvorgaben für Ihre Zope-Sitzung setzen. Sie können sogar den obersten Rahmen ausblenden.

Logout (Abmelden, Anm. d. Üb.)
: Durch anwählen dieses Menüelements melden Sie sich von Zope ab.

Quick Start Links (Schnellstart-Links, Anm. d. Üb.)
: Dies sind schnelle Links zu Dokumentations- und Gemeinde-Ressourcen für Zope.

Benutzer in Zope verstehen

Zope ist ein Mehrbenutzersystem. Sie haben schon gesehen, wie Sie sich in Zope über das Management-Interface mit den Anfangsbenutzernamen und -kennwort anmelden können. Zope unterstützt andere Arten von Benutzern:

Notfallbenutzer (Emergency User)
: Der Notfallbenutzer wird bei Zope selten verwendet. Dieses Konto wird für das Erstellen von anderen Benutzerkonten und das Instand setzen von Dingen verwendet, falls Sie sich zufällig aussperren sollten.

  Der Notfallbenutzer ist sowohl sehr mächtig als auch sehr beschränkt. Er wird von den meisten Sicherheitssteuerungen nicht eingeschränkt, kann jedoch nur eine Art von Objekten erstellen: Benutzer. Der Einsatz des Notfallbenutzers, um Ihr Zope-System im Falle einer versehentlichen Aussperrung zu reparieren wird im Administrator's Guide erörtert.

Manager
: Der Manager ist das Arbeitstier in Zope. Sie müssen sich mit dem Managerkonto anmelden, um die meiste mit dem Aufbau von Zope-Web-Sites verbundene Arbeit zu erledigen. Der Anfangsbenutzer ist ein Manager und Sie können so viele Managerkonten erstellen wie Sie benötigen.

Andere
: Sie können Ihre eigene Art von Benutzern erstellen, die bestimmten Gruppen angehören, oder dafür verantwortlich sind, einen von Ihnen definierten Aufgabenbereich auszuführen. Dies wird ausführlicher in Kapitel 7, "Benutzer und Sicherheit" erklärt, das Zopes Sicherheits- und Benutzerverwaltung erörtert.

Benutzer erstellen

Manager können Zope-Benutzer in einer besonderen Art von Ordner, dem *User Folder*, erstellen.

Symbole der User Folder sehen wie ein Ordner mit einer Person darauf aus. User Folder haben immer den Namen *acl_users*, wie in Abbildung 2-1 gezeigt.

Klicken Sie auf den Ordner acl_users im Stammordner, um ihn zu öffnen. User Folder enthalten Benutzerobjekte. Sie können neue Benutzer erstellen und vorhandene Benutzer bearbeiten. Klicken Sie auf die Schaltfläche *Add*, um einen neuen Benutzer zu erstellen, wie in Abbildung 2-2 gezeigt.

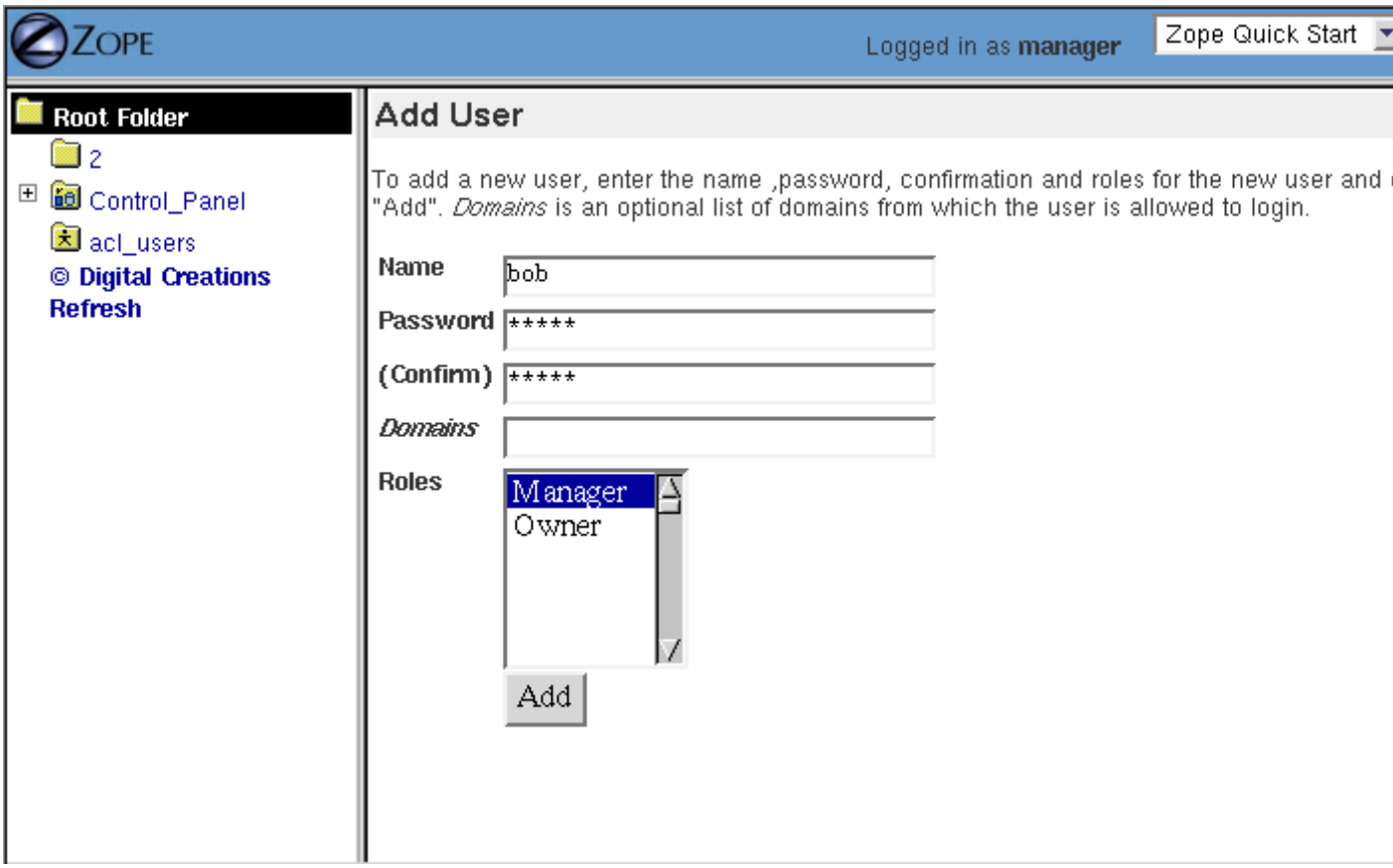


Abbildung 2-2 Hinzufügen eines neuen Benutzers.

Füllen Sie das Formular aus, um einen neuen Benutzer zu erstellen. Geben Sie in das *Name*-Feld Ihren gewünschten Benutzernamen ein. Wählen Sie ein Passwort und geben Sie es in die *Password*- und *(Confirm)*-Felder ein. Lassen Sie das Feld *Domains* frei. Dies ist eine fortgeschrittene Funktion und wird in Kapitel 7, "Benutzer und Sicherheit" erörtert. Wählen Sie die Rolle *Manager* in der *Roles*-Auswahlliste. Klicken Sie dann die Schaltfläche *Add*.

Herzlichen Glückwunsch, Sie haben gerade ein Managerkonto erstellt. Zope zeigt Ihnen dieses neue Managerkonto im User Folder an. Wenn Sie wollen, können Sie den Benutzer später ändern oder löschen.

Anmeldeinformationen ändern

Um Ihre Anmeldeinformationen zu ändern, wählen Sie *Logout* im oberen Rahmen des Management-Interfaces aus. Sie werden aufgefordet, sich erneut anzumelden. Um die Anmeldeinformationen zu ändern, geben Sie einen neuen Benutzernamen und ein neues Passwort ein.

Um sich abzumelden, wählen Sie *Logout* im oberen Rahmen des Management-Interfaces aus und brechen Sie die erneute Anmeldeaufforderung ab. Nun sollten Sie eine Meldung sehen, die Ihnen mitteilt, dass Sie abgemeldet wurden. Wenn Sie versuchen auf das Zope-Management-Interface zuzugreifen nachdem sie abgemeldet wurden, werden Sie erneut zur

Anmeldung aufgefordert. Sie können sich von Zope auch abmelden, indem Sie Ihren Web-Browser beenden.

Objekte erstellen

Das Zope-Management-Interface stellt alles in Metaphern von Objekten und Ordnern dar. Wenn Sie Webanwendungen mit Zope aufbauen, verwenden Sie die meiste Zeit darauf Objekte in Ordnern zu erstellen und zu verwalten. Zum Beispiel erstellen Sie ein Benutzerobjekt in einem User Folder, um ein neues Manager-Konto zu erzeugen.

Kehren Sie zum Stammordner durch Klicken auf den linken oberen Ordner im Navigator-Rahmen zurück.

Um dem aktuellen Ordner ein neues Objekt hinzuzufügen, wählen Sie ein Objekt aus der mit "Select type to add..." betitelten Auswahl aus. Diese Auswahlliste heißt *Produktauswahlliste*.

Um beispielsweise einen Ordner hinzuzufügen, wählen Sie *Folder* in der Produktauswahlliste. An dieser Stelle werden Sie zu einem Erstellformular weitergleitet, das Informationen über den neuen Ordner abfragt, wie in Abbildung 2-3 gezeigt.

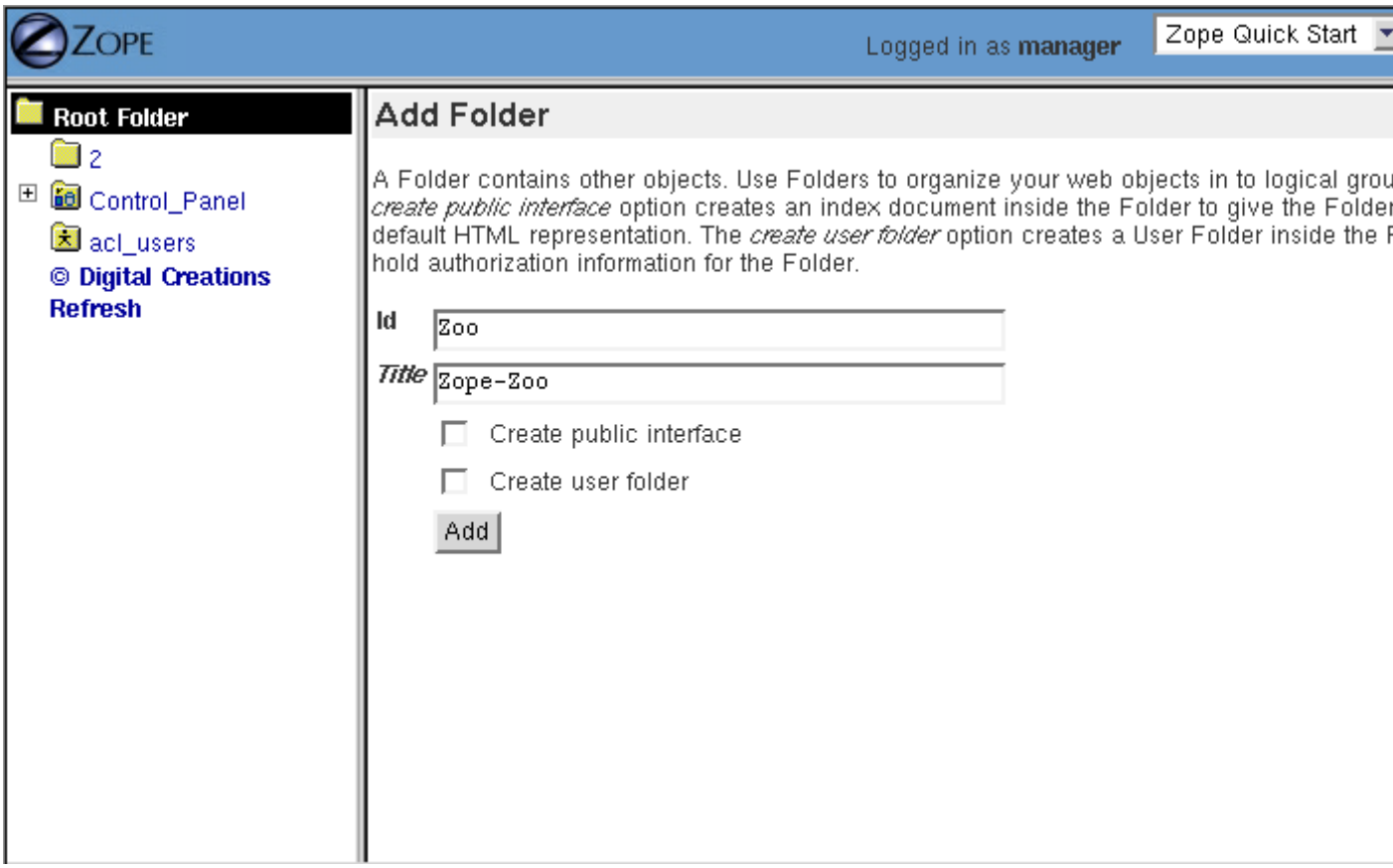


Abbildung 2-3 Formular zum Hinzufügen von Ordnern.

Geben Sie "Zoo" im Feld *Id* und "Zope-Zoo" im Feld *Title* ein. Klicken Sie dann die Schaltfläche *Add*.

Zope erstellt nun einen neuen Ordner im aktuellen Ordner. Sie können dieses überprüfen, indem Sie nachsehen, ob sich jetzt ein neuer Ordner mit dem Namen *Zoo* im Stammordner befindet.

Klicken Sie auf *Zoo*, um ihn zu öffnen. Achten Sie darauf, dass der URL des Ordners auf der ID des Ordners basiert. Wenn Sie wollen, können Sie mehrere Ordner in Ihrem neuen Ordner erstellen. Erstellen Sie zum Beispiel einen Ordner im *Zoo*-Ordner mit der Kennung *Arktis*. Gehen Sie zum *Zoo*-Ordner und wählen Sie *Folder* in der Auswahlliste. Dann geben Sie "Arktis" für die ID des Ordners ein und "arktischer Ausstellungsbereich" für den Titel ein. Klicken Sie jetzt auf die Schaltfläche *Add*. Sie erstellen neue Objekte immer auf dieselbe Weise:

1. Gehen Sie zum Ordner, dem Sie ein neues Objekt hinzufügen wollen.
2. Wählen Sie die Art des Objekts, die Sie hinzufügen wollen, in der Auswahlliste.
3. Füllen Sie das Erstellformular aus und bestätigen Sie es.
4. Zope erstellt ein neues Objekt im aktuellen Ordner.

Beachten Sie, dass jedes Zope-Objekt eine Kennung hat, die Sie im Erstellformular angeben müssen, wenn Sie das Objekt erstellen. Die Kennung bestimmt, wie Zope Objekte benennt. Objekte verwenden auch ihre IDs für ihre URLs.

Kapitel 3 "Verwenden grundlegender Zope-Objekte" behandelt alle grundlegenden Objekte und was diese für Sie tun können.

Objekte bewegen

Die meisten Rechnersysteme erlauben Ihnen, Dateien in Verzeichnissen mit Hilfe von ausschneiden, kopieren und einfügen zu bewegen. Zope hat ein ähnliches System, mit dem Sie Objekte in Ordnern bewegen können, indem Sie sie ausschneidet oder kopiert und sie dann an einem neuen Ort einfügt.

Um mit kopieren und einfügen zu experimentieren, erstellen Sie einen neuen Ordner im Stammordner mit der ID *Baeren*. Wählen Sie *Baeren* danach aus, indem Sie das Kontrollkästchen links neben dem Ordner ankreuzen. Klicken Sie dann auf die Schaltfläche *Cut*. Cut entfernt die ausgewählten Objekte aus dem Ordner und legt sie in die Zwischenablage. Das Objekt verschwindet jedoch *nicht* von seinem Standort, bis es woanders eingefügt wird.

Gehen Sie nun in den *Zoo*-Ordner indem Sie ihn anklicken und gehen Sie danach in den Ordner *Arktis* indem Sie ihn anklicken. Sie hätten auch den Navigator verwenden können, um an dieselben Stelle zu gelangen. Klicken Sie jetzt auf die Schaltfläche *Paste*, um das/die ausgeschnittenen Objekt(e) in den aktuellen Ordner einzufügen. Sie sollten nun sehen, dass der Ordner *Baeren* an seinem neuen Standort erscheint. Sie können überprüfen, ob der Ordner verschoben wurde, indem Sie zum Stammordner gehen und sehen, dass der Ordner *Baeren* sich nicht mehr darin befindet.

Copy arbeitet ähnlich wie Cut. Wenn Sie kopierte Objekte einfügen, werden die Originalobjekte nicht geändert. Wählen Sie das/die Objekt(e) die Sie kopieren wollen aus und

klicken Sie auf die Schaltfläche *Copy*. Navigieren Sie danach zu einem anderen Ordner und klicken auf die Schaltfläche *Paste*.

Sie können Ordner ausschneiden und kopieren, die andere Objekte enthalten und so mit einem einzelnen Copy und Paste viele Objekte auf einmal bewegen. Gehen Sie beispielsweise zum *Zoo*-Ordner und kopieren Sie den Ordner *Arktis*. Fügen Sie es jetzt in den *Zoo*-Ordner ein. Sie haben jetzt zwei Ordner im *Zoo*-Ordner, *Arktis* und *copy_of_Arktis*. Wenn Sie ein Objekt in denselben Ordner einfügen, aus dem Sie es kopiert haben, ändert Zope die Kennung des eingefügten Objekts. Dies ist ein notwendiger Schritt, da Sie keine zwei Objekte mit derselben Kennung in dem selben Ordner haben können.

Um den Ordner *copy_of_Arktis* umzubenennen, wählen Sie den Ordner durch ankreuzen des Kontrollkästchens links vom Ordner aus. Klicken Sie dann auf die Schaltfäche *Rename*. Dies bringt Sie zum Umbenennungs-Formular wie in Abbildung 2-4 gezeigt.

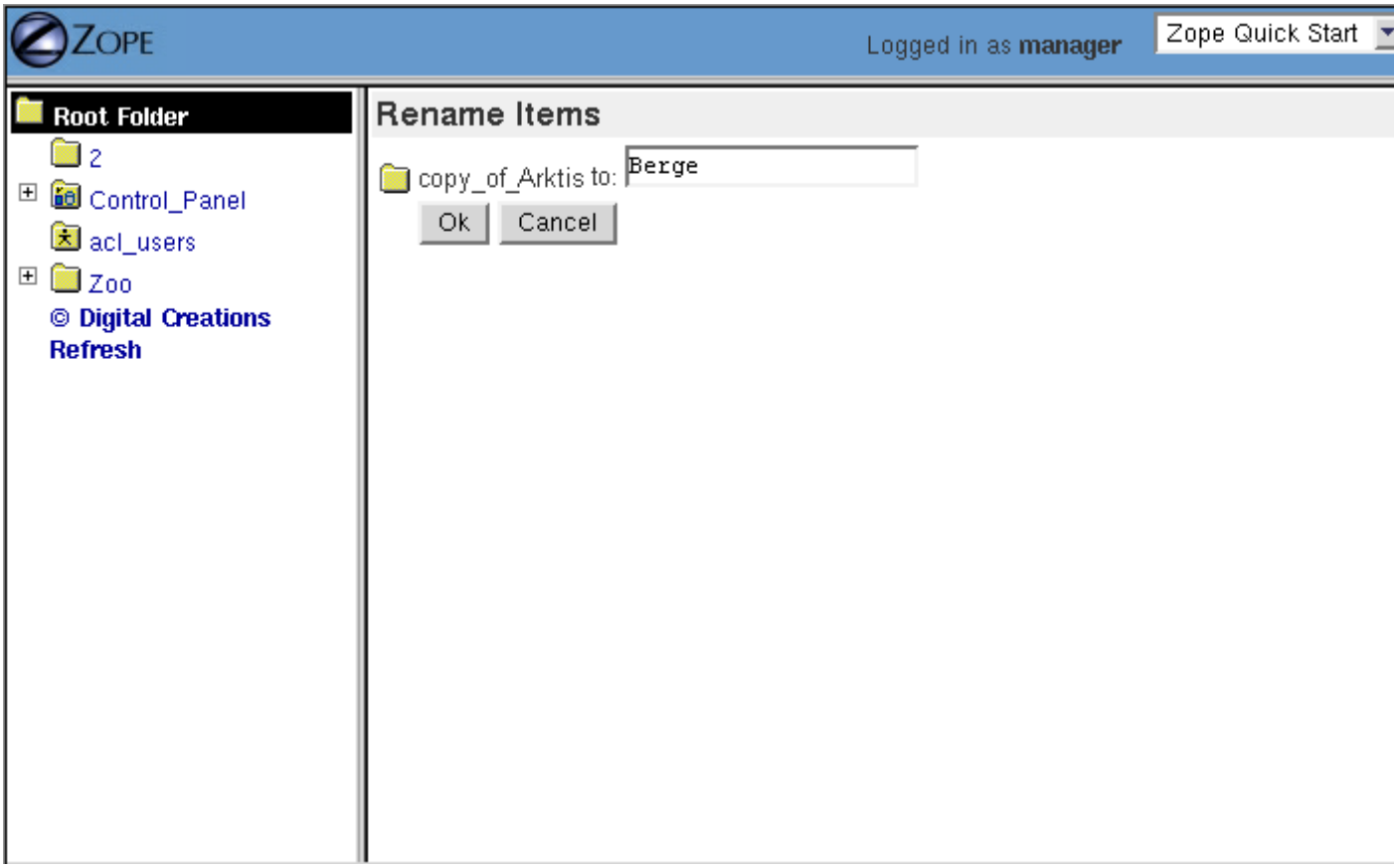


Abbildung 2-4 Umbenennen eines Objekts.

Geben Sie die neue Id "Berge" ein und klicken Sie auf *OK*. Zope-IDs können aus Buchstaben, Ziffern, Leerzeichen, Bindestrichen, Unterstrichen und Punkten bestehen und unterscheiden Groß- und Kleinschreibung. Hier sind einige gültige Zope-IDs: *index.html*, *42* und *Schlangengrube*.

Jetzt enthält Ihr Zoo-Ordner einen *Arktis*- und einen *Berge*-Ordner. Jeder dieser Ordner enthält einen *Baeren*-Ordner. Der Grund dafür ist, dass, als wir eine Kopie des *Arktis*-Ordners erzeugt haben, auch der darin enthaltene *Baeren*-Ordner mitkopiert wurde.

Wenn Sie ein Objekt löschen möchten, wählen Sie es aus und klicken Sie dann auf die Schaltfläche *Delete*. Im Gegensatz zu ausgeschnittenen Objekten, werden gelöschte Objekte nicht in die Zwischenablage gelegt und können nicht wieder eingefügt werden. Im nächsten Abschnitt sehen wir, wie wir gelöschte Objekte wieder auffinden können.

In Zope können Sie einige besondere Objekte im Stammordner nicht ausschneiden, löschen oder umbenennen. Zu diesen Objekten gehört das *Control_Panel*, der *standard_html_header*, der *standard_html_footer* und die *standard_error_message*. Diese wichtigen Objekte sind für Zopes Betrieb erforderlich. In einigen Fällen können Sie diese Vorgänge nicht durchführen: Zum Beispiel können Sie kein benutzerdefiniertes Objekt in einen gewöhnlichen Ordner einfügen.

Wenn Sie Probleme mit Ausschneiden und Einfügen haben, vergewissern Sie sich, dass Sie Cookies in Ihrem Browser aktiviert haben. Zope verwendet Cookies, um die Objekte im Auge zu behalten, die Sie ausschneiden und kopieren.

Fehler rückgängig machen

Jede Aktion in Zope, die bewirkt, dass sich Objekte ändern, kann über den *Undo*-Reiter rückgängig gemacht werden. Sie können den Status vor einem gemachten Fehler wiederherstellen indem Sie ihn rückgängig machen.

Wählen Sie den *Zoo*-Ordner aus, den wir vorher erstellt haben und klicken Sie auf *Delete*. Der Ordner verschwindet. Sie können ihn durch Widerrufen der Delete-Aktion wiederherstellen.

Klicken auf den Reiter *Undo*, wie in Abbildung 2-5 dargestellt.

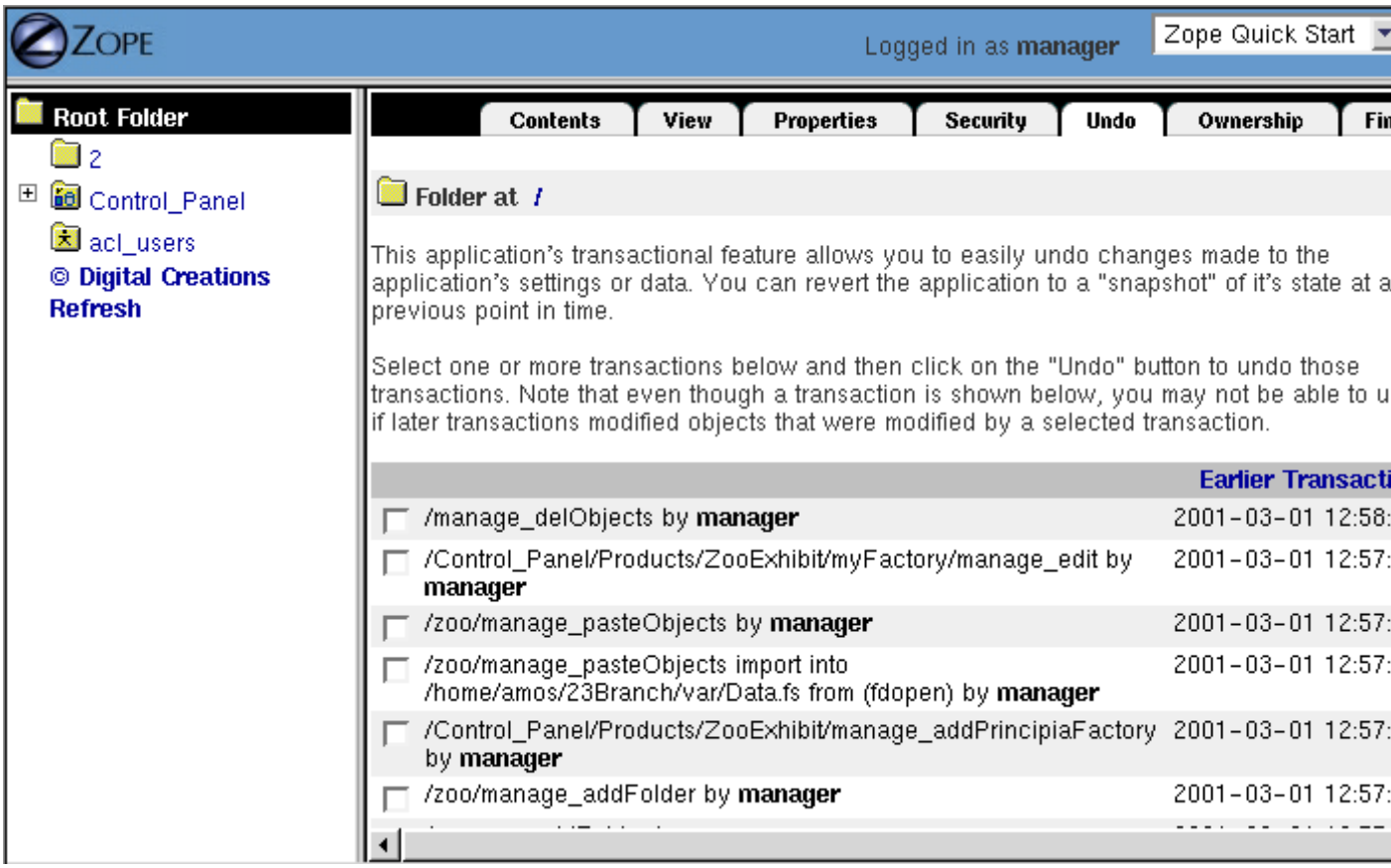


Abbildung 2-5 Die Undo-Ansicht.

Wählen Sie die erste Transaktion */manage_delObjects* aus und klicken Sie auf die Schaltfläche *Undo*.

Dieses Vorgehen veranlasst Zope dazu, die letzte Transaktion zu widerrufen. Sie können sicherstellen, dass die Aufgabe beendet worden ist, indem Sie sich vergewissern, dass der Ordner *Zoo* zurückgekehrt ist.

Undo-Details und -Kniffe

Undo arbeitet mit der Objektdatenbank, die Zope verwendet, um alle Zope Objekte zu speichern. Änderungen an der Objektdatenbank gehen in Transaktionen vor sich. Sie können sich eine Transaktion als jede Änderung vorstellen, die Sie in Zope durchführen, wie einen Ordner zu erstellen oder ein Bündel von Objekten an einer neuen Stelle einzufügen. Jede Transaktion beschreibt alle Änderungen, die geschehen, während die Aktion ausgeführt wird.

Sie können keine Transaktion widerrufen, von der eine spätere Transaktion abhängt. Wenn Sie beispielsweise ein Objekt in einen Ordner einfügen und dann ein Objekt in demselben Ordner löschen, könnten Sie sich fragen, ob Sie ungeachtet dessen das vorherige Einfügen widerrufen können. Beide Transaktionen ändern denselben Ordner, sodass Sie die frühere Transaktion nicht einfach widerrufen können. Die Lösung ist beide Transaktionen zu widerrufen. Sie können mehr als eine Transaktion auf einmal widerrufen, indem Sie mehrere Transaktionen im *Undo*-Reiter ankreuzen und dann auf *Undo* klicken.

Ein anderes Problem, das Sie sich vor Augen halten müssen ist, dass Sie ein Undo nicht rückgängig machen können. Deshalb können Sie, wenn Sie einen Ordner hinzufügen und dann diese spezielle Aktion widerrufen, den neuen Ordner nicht durch Rückgängig machen des Undos zurückbekommen.

Eine letzte Bemerkung zu Undo. Nur Änderungen an in Zope gespeicherten Objekten können rückgängig gemacht werden. Wenn Sie Daten in einem relationalen Datenbankserver wie Oracle oder MySQL integriert haben (wie in Kapitel 12, "Verbindung mit relationalen Datenbanken", erörtert wird), können Sie Änderungen an dort gespeicherten Daten nicht rückgängig machen.

Zope verwalten und überwachen

Die Steuerleiste, das Control Panel, ist ein Objekt im Stammordner, das verschiedene Aspekte von Zopes Betrieb steuert.

Klicken Sie auf das *Control_Panel*-Objekt im Stammordner, wie in Abbildung 2-6 gezeigt.

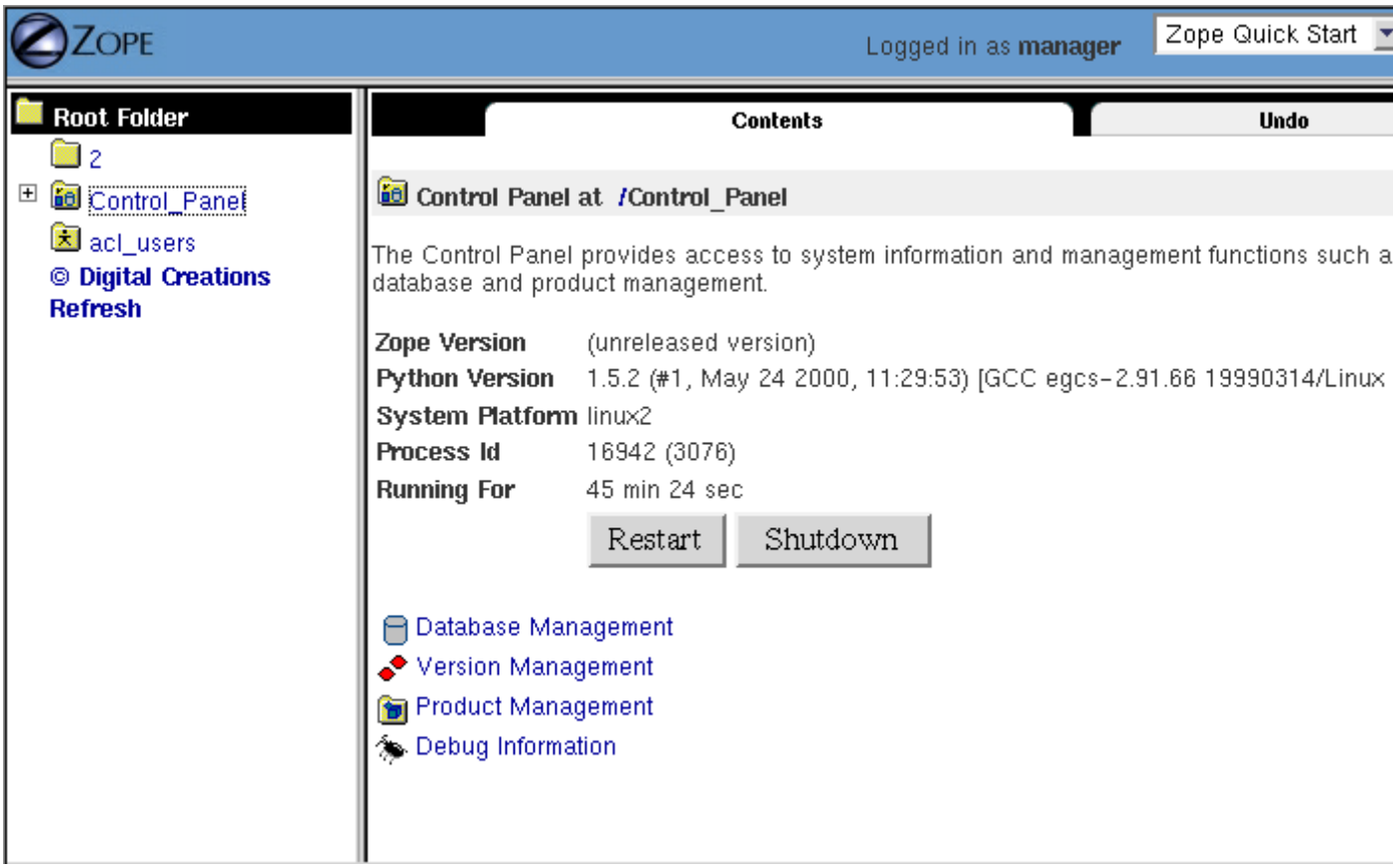


Abbildung 2-6 Die Steuerleiste.

Um Zope herunterzufahren, klicken Sie die Schaltfläche *Shutdown*. Das Herunterfahren von Zope bewirkt, dass der Server aufhört Anfragen zu bearbeiten und völlig aus dem Speicher verschwindet. Sie müssen Zope manuell starten, um mit dessen Betrieb fortzufahren. Fahren Sie Zope nur herunter, wenn Sie mit Ihrer Arbeit fertig sind und Sie Zugang zu dem Server haben, auf dem Zope läuft, sodass Sie Zope später manuell neu starten können.

Wenn Sie Zope unter Unix als Daemon oder als Dienst unter Windows ausführen, können Sie Zope vom Ordner Control Panel aus neu starten. Wenn Sie auf die Schaltfläche *Restart* klicken, fährt Zope herunter und startet danach sofort ein neue Instanz des Zope-Servers. Es kann einige Zeit in Anspruch nehmen bis Zope wieder hochfährt und mit der Bearbeitung von Anfragen beginnen kann.

In der Steruerleiste sehen Sie auch mehrere Links am unteren Bildschirmrand, von denen eins die *Datenbankverwaltung* (Database Management) ist.

Transaktionen werden nicht gelöscht, bis Sie nicht die Zope-Datenbank packen. Dies bedeutet, dass Sie alle Transaktionen außer denen widerrufen können, die durch das Packen der Datenbank entfernt worden sind. Wenn Sie die Datenbank packen wollen, können Sie angeben, welche Transaktionen entfernt werden sollen, sodass Sie zum Beispiel nur Transaktionen entfernen können, sie älter sind als eine Woche.

Verwenden des Hilfesystems

Zope hat eine eingebautes Hilfesystem. Jeder Management-Bildschirm hat eine Help-Schaltfläche in der rechten oberen Ecke. Diese Schaltfläche öffnet ein anderes Browserfenster und bringt Sie zum Zope-Hilfesystem.

Gehen Sie zum Stammordner. Klicken die Help-Schaltfläche und Sie sollten etwas sehen, was aussieht wie in Abbildung 2-7.

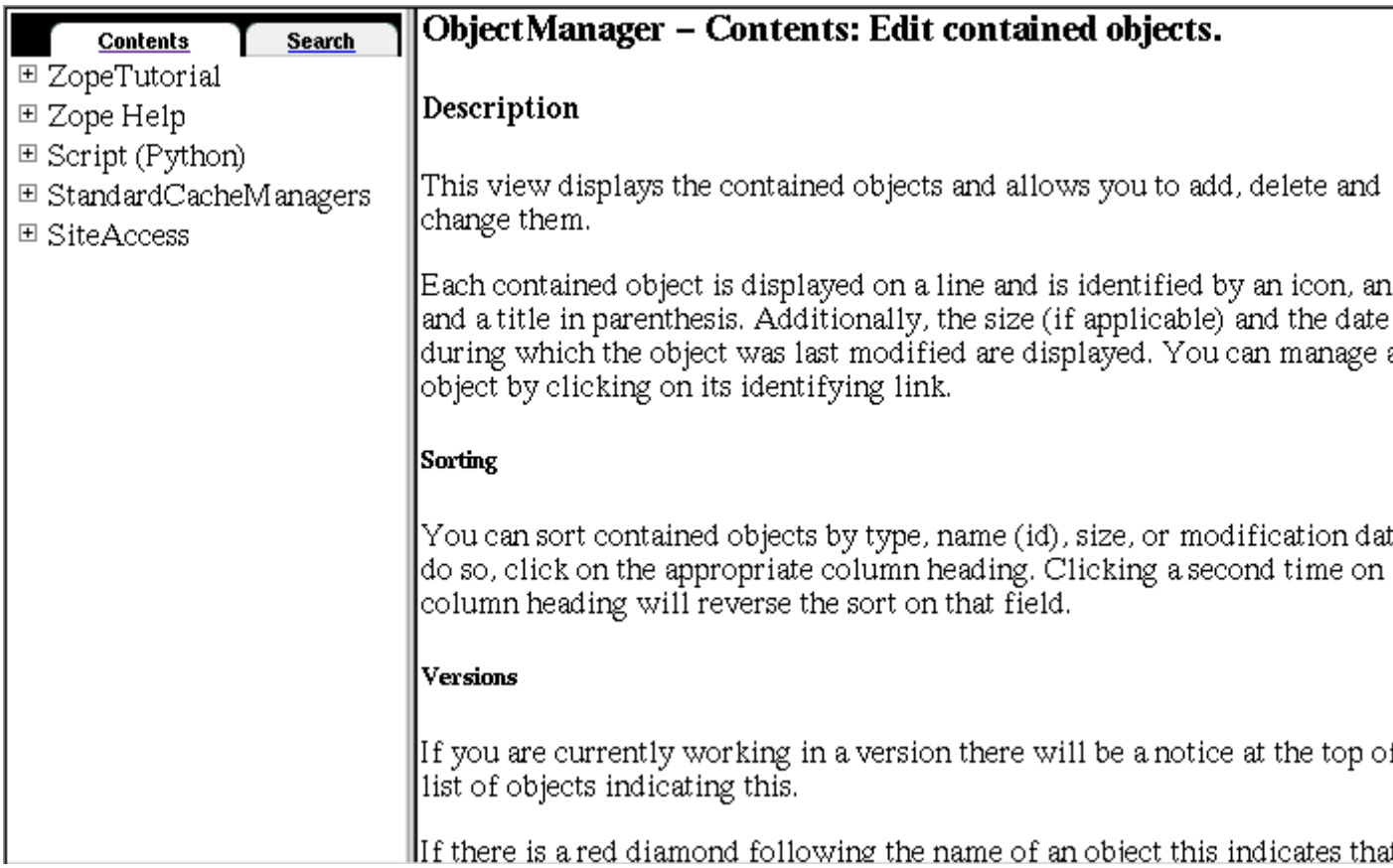


Abbildung 2-7 Das Hilfesystem.

Das Hilfesystem ist, wie das Zope-Management-Interface, in zwei Rahmen aufgeteilt, einen für Navigation und einen für das Anzeigen des aktuellen Themas.

Jedes Mal wenn Sie auf die Hilfe-Schaltfläche des Zope-Management-Bildschirms klicken, zeigt der rechte Rahmen des Hilfesystems das Hilfe-Thema für den aktuellen Management-Bildschirm an. In diesem Fall sehen Sie Information über die Inhaltsübersicht *Contents* eines Ordners.

Durchblättern und Durchsuchen der Hilfe

Normalerweise verwenden Sie das Hilfesystem, um Hilfe zu einem bestimmten Thema zu bekommen. Jedoch können Sie den gesamten Inhalt der Hilfe durchblättern, wenn Sie neugierig sind oder einfach Informationen über andere Dinge finden wollen, außer des Management-Bildschirms, den Sie zur Zeit betrachten.

Das Hilfesystem lässt Sie alle Hilfe-Themen im *Contents*-Reiter des linken Hilferahmens durchblättern, wie in Abbildung 2-7 gezeigt wird. Sie können Hilfe-Themen öffnen und schließen. Um ein Hilfe-Thema im rechten Rahmen zu betrachten, klicken Sie darauf.

Die gesamte Hilfe zu den Zope Management-Bildschirmen befindet sich im Ordner *Zope Help*. In diesem finden Sie viele Hilfe-Themen. Sie finden auch einen Hilfeordner namens *API Reference*. In Ordner befindet sich Hilfe dazu, wie man in Zope programmiert, was in Kapitel 9, "Zope-Scripting für Fortgeschrittene", näher erklärt wird.

Wenn Sie Komponenten von Drittherstellern installieren, beinhalten diese auch Hilfe-Themen, die Sie durchblättern können. Jede installierte Komponente hat ihren eigenen Hilfeordner.

Sie können das Hilfesystem durchsuchen, indem Sie auf den Search-Reiter klicken und eine oder mehrere Suchbedingungen eingeben. Geben Sie beispielsweise "Folder" in das Suchformular ein, um alle Hilfe-Themen zu finden, die Ordner erwähnen.

Mit dem Zope-Tutorium beginnen

Zope bringt ein Tutorium mit. Das Tutorium führt Sie durch alle Grundlagen darüber, wie Sie Zope-Objekte erstellen und verwalten. Um mit dem Tutorium zu beginnen, fügen Sie dem aktuellen Ordner ein Zope Tutorium durch Auswählen von *Zope Tutorial* aus der Produktauswahlliste hinzu. geben Sie ihm eine in diesem Ordner eindeutige ID, wie *tutorium*. Das Tutorium beinhaltet mehrere Beispiele, die Sie je nach Bedarf verändern und kopieren können.

Wenn Sie das Tutorium begonnen haben und es nun beenden wollen, bevor Sie alle Lehrgänge beendet haben, können Sie später zum Tutorium zurückkehren. Gehen Sie einfach zum Hilfesystem und suchen Sie durch Durchblättern des *Zope Tutorial*-Hilfeordners den Lehrgang, mit der Sie gerne fortfahren würden. Es ist nicht nötig das Tutorium erneut zu installieren.

Wenn Sie Probleme mit dem Tutorium haben, vergewissern Sie sich, dass Cookies in Ihrem Browser aktiviert sind. Das Tutorium verwendet Cookies, um im Auge zu behalten wo Ihre Beispielobjekte sind. Auch wenn Sie Javascript in Ihrem Browser aktivieren, stellt das Tutorium sicher, dass das Zope-Management-Interface mit Ihrem Lehrgang synchron bleibt.

Nun, da Zope läuft, ist es Zeit, das System gründlicher zu erkunden. Sie haben gesehen, wie Zope über das Web verwaltet werden kann und Sie haben ein bisschen über die Zope-Objekte erfahren. Im nächsten Kapitel treffen Sie auf viele verschiedene Zope-Objekte und finden heraus, wie man einfache Anwendungen mit ihnen aufzubauen kann.

Zopebuch: Inhaltsverzeichnis

Kapitel 3: Verwendung grundlegendender Zope-Objekte

Wenn Sie eine Webanwendung mit Zope aufbauen, erstellen Sie die Anwendung aus Objekten. Je nach Design sind unterschiedliche Objekte für unterschiedliche Teile Ihrer

Anwendung zuständig. Einige Objekte speichern Ihre Inhaltsdaten, wie Dokumente von Textverarbeitungsprogrammen, Arbeitsblätter (Spread-Sheets) und/oder Bilder. Einige Objekte steuern, wie dynamische Web-Inhalte erzeugt werden, indem sie Eingaben aus einem Web-Formular akzeptieren oder E-Mails senden. Einige Objekte kontrollieren, wie Ihre Inhalte angezeigt oder Ihrem Betrachter *präsentiert* werden, zum Beispiel als Web-Seite oder per E-Mail. Im Allgemeinen übernehmen Zope-Objekte drei Arten von Aufgabenbereichen:

Inhalt
: Zope-Objekte wie Dokumente, Bilder und Dateien beinhalten verschiedene Arten von Text- und binären Daten. Zusätzlich zu Objekten in Zope, die Inhalt enthalten, kann Zope mit extern gespeicherten Inhalten, zum Beispiel in einer relationalen Datenbank, arbeiten.

Logik
: Zope hat Funktionen für das Schreiben von Geschäftslogik. Zope erlaubt Ihnen, Verhalten mit Hilfe von Python, Perl und SQL zu schreiben. "Geschäftslogik" ist jede Art der Programmierung, die keine Darstellung einschließt, sondern es erfordert Aufgaben auszuführen, wie das Ändern von Objekten, das Senden von Nachrichten, das Prüfen von Bedingungen und das Antworten auf Ereignisse.

Darstellung
: Sie können das Look-And-Feel Ihrer Site mit Zope-Objekten steuern, die als Web-Vorlagen arbeiten. Zope besitzt eine Tag-basierte Programmiersprache namens Document Template Markup Language (DTML) um die Darstellung zu steuern.

Das Wort *Objekt* ist ein schwer beladener Ausdruck. Je nach Ihrem Wissenshintergrund kann es eine große Anzahl verschiedener Dinge bedeuten. In diesem Kapitel können Sie sich ein Zope-Objekt als ein Stück Software vorstellen, das Sie mit Hilfe eines Web-Browsers steuern und bearbeiten können.

Zope bringt viele integrierte Objekte mit, die Ihnen helfen, verschiedene Aufgaben durchzuführen. Sie können auch Zope-Objekte (Produkte) von Drittherstellern installieren, um Zopes Leistungsspektrum zu erweitern. Dieses Kapitel erklärt die grundlegendsten Objekte, und wie sie funktionieren. Sie können vollständig funktionierende Zope-Sites mit Hilfe der wenigen Grundobjekte erstellen, die in diesem Kapitel erläutert werden.

Dieses Kapitel ist in etwa gemäß der drei obigen Kategorien Inhalt, Logik und Darstellung strukturiert. Es gibt andere Arten von Objekten in Zope, die nicht eindeutig in einen dieser drei Aufgabenbereiche passen. Diese sind am Ende des Kapitels erklärt.

Zope-Ordner verwenden

Ordner (Folders, Anm. d. Üb.) sind die Bausteine von Zope. Der Zweck eines Ordners ist, andere Objekte zu *enthalten* und Objekte dadurch zu *gliedern*, dass er sie in verschiedene Gruppen unterteilt.

Ordner können alle Arten von Objekten einschließlich anderer Ordner enthalten, so dass Sie Ordner ineinander verschachteln können, um eine Baumstruktur von Ordnern zu erzeugen. Diese Art Anordnung von Ordnern innerhalb von Ordnern verleiht Ihrer Zope-Site *Struktur*. Eine gute Struktur ist sehr wichtig, da fast alle Aspekte von Zope (von Sicherheit über Verhalten zu Darstellung) von der Ordnerstruktur Ihrer Site beeinflusst werden.

Ordnerstrukturen sollten jedem sehr vertraut sein, der schon einmal mit Dateien und Ordnern mit einem Dateimanagerprogramm auf seinem Computer gearbeitet hat, wie dem Microsoft *Windows Explorer* oder irgendeinem der beliebten X-Dateimanager wie dem *xfm*, dem *kfm*, *konqueror*, *Gnome*-Dateimanager u.v.a. Das Zope-Management-Interface versucht, diesen bekannten Programmen so ähnlich wie möglich zu sein, so dass Sie mit der Verwaltung von Zope-Objekten ebenso vertraut sind, wie wenn Sie Dateien über Ihren Computer verwalten würden.

Ordnerinhalte verwalten

In Kapitel 2, "Zope verwenden", haben Sie Objekte erstellt und Objekte verschoben. Kurz gesagt, Sie erstellen Objekte in Ordnern durch das Wählen des Namens des gewünschten Objekts in der Auswahlliste am oberen Rand der Inhaltsübersicht. Dann füllen Sie das Erstellformular aus und schicken es ab. Ein neues Objekt wird dem aktuellen Ordner danach hinzugefügt. Über das verwenden der Schaltflächen *Cut*, *Copy*, *Paste* und *Rename* können Sie Objekte zwischen Ordnern bewegen.

Objekte importieren und exportieren

Sie können Objekte mit Hilfe von *Export* und *Import* von einem Zope-System zu einem anderen bewegen. Sie können alle Arten von Zope-Objekten in eine *Exportdatei* exportieren. Diese Datei kann dann in jedes andere Zope-System importiert werden.

Sie können sich das Exportieren eines Objekt wie das Klonen eines Stücks Ihres Zope-Systems in eine Datei vorstellen, die Sie sich dann von Rechner zu Rechner bewegen lässt. Sie können nun diese Datei nehmen und den Klon auf jeden anderen Zope-Server aufsetzen. Stellen Sie sich vor, Sie haben einige Dokumente in einem Zope-Ordner. Wenn Sie gerade jene Objekte zum Zope-System Ihres Freunds kopieren wollten, könnten Sie den Ordner exportieren und die Exportdatei per E-Mail an Ihren Freund senden, der ihn dann importieren könnte.

Nehmen wir an, Sie haben einen Ordner für Hausaufgaben, die Sie aus dem schulischen Zope-Server exportieren und mit nach Hause nehmen wollen, um daran an Ihrem heimischen Zope-Server weiterzuarbeiten. Sie können einen solchen Ordner Namens "Heimarbeit" in Ihrem Stammordner erstellen. Gehen Sie zum Ordner, der Ihren Ordner *Heimarbeit* enthält. Wählen Sie den Ordner *Heimarbeit* (im Beispiel Abbildung 3-1 homeWork) durch Abhaken des nebenstehenden Kontrollkästchens aus. Klicken Sie dann die Schaltfläche *Import/Export*. Sie sollten sich jetzt in der Ordner-Ansicht Import/Export befinden, wie in Abbildung 3-1 gezeigt.

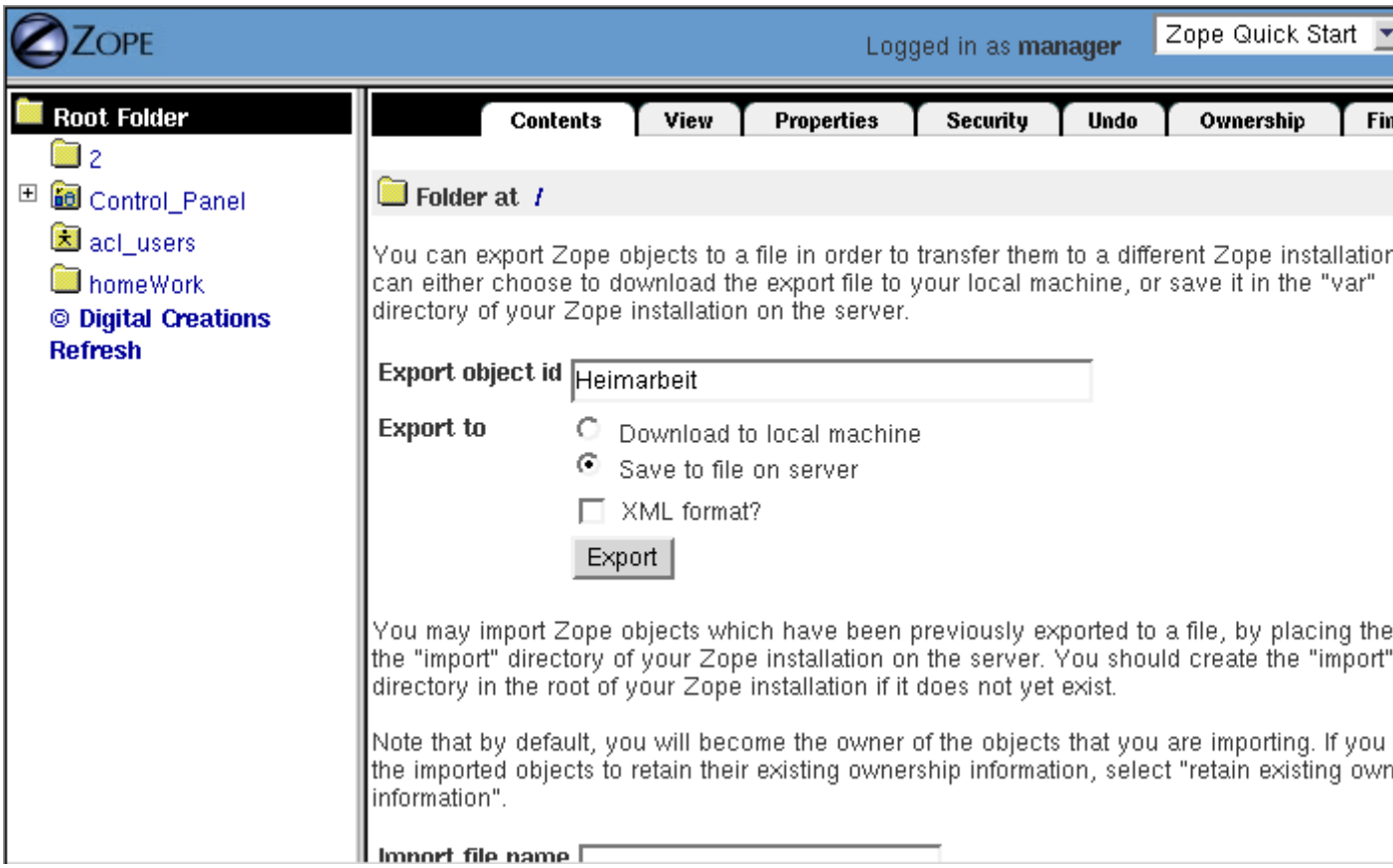


Abbildung 3-1 Die Import-/Export-Ansicht

Es gibt zwei Bereiche in dieser Darstellung. Die obere Hälfte ist der Exportbereich, und die untere Hälfte ist der Importbereich. Um ein Objekt aus dieser Darstellung zu exportieren, geben Sie die ID des Objekts in das erste Formularfeld "Export object id" ein. In unserem Fall füllt Zope dieses Feld schon für uns aus, da wir den Ordner *Heimarbeit* bereits in der letzten Darstellung ausgewählt haben.

Die nächste Formularoption lässt Sie wählen, ob Sie die Exportdatei auf Ihren Computer herunterzuladen oder sie auf dem Server lassen wollen. Wenn Sie "*Download to local machine*" ("Auf lokalen Recher laden") aktivieren und auf die Schaltfläche Export klicken, fordert Sie Ihr Web-Browser dazu auf, die Exportdatei herunterzuladen. Wenn Sie "*Save to file on server*" ("Datei auf Server speichern") wählen, sichert Zope die Datei auf dem selben Rechner, auf dem Zope läuft und Sie müssen die Datei von diesem Standort selbst abholen. Die Exportdatei wird in Zopes *var*-Verzeichnis auf Ihrem Server geschrieben. Standardmäßig haben Exportdateien die Endung *.zexp*.

Im Allgemeinen ist es praktischer, die Exportdatei auf Ihren lokalen Rechner herunterzuladen. Manchmal ist es stattdessen angenehmer, die Datei auf den Server zu sichern, zum Beispiel wenn Sie eine langsame Verbindung haben und die Exportdatei sehr groß ist oder wenn Sie nur versuchen, das exportierte Objekt zu einer anderen Zope-Instanz auf dem selben Rechner zu bringen.

Das letzte Feld im Exportformular ist das "*XML format?*"-Kontrollkästchen. Wenn Sie dieses Kästchen ankreuzen, wird das Objekt im *eXtensible Markup Language* (XML)-Format

exportiert. Ist dieses Kästchen nicht angekreuzt, wird die Datei in Zopes binärem Format exportiert. Das XML-Format ist beim Herunterladen zwar wesentlich größer, aber es ist für M enschen lesbar und kann als XML syntaktisch analysiert werden. Bis jetzt ist das einzige Werkzeug, das dieses Format versteht, Zope selbst, aber in Zukunft wird es vielleicht andere Hilfsprogramme geben, die Zopes XML Format verstehen können. Im Allgemeinen sollten Sie dieses Kästchen unangekreuzt lassen, es sei denn, Sie sind neugierig darauf, wie das XML-Format aussieht und wollen es von Hand prüfen.

Klicken Sie die Schaltfläche Export und sichern Sie Ihre Exportdatei *Heimarbeit.zexp* (Die Dateierweiterung zexp wird automatisch vergeben).

Nehmen wir jetzt an, Sie sind nach Hause gegangen und wollen die Datei in Ihren heimischen Zope-Server importieren. Zuerst müssen Sie die Exportdatei in das *import*-Verzeichnis Ihrer Zope-Installation kopieren. Dieses Verzeichnis finden Sie auf dem Rechner, auf dem Zope installiert ist. Gehen Sie jetzt zur *Import-/Export*-Ansicht des Ordners, in dem Sie den Import ausführen wollen. Geben Sie den Namen der Exportdatei in das Formualrfeld "*Import file name*" ein und klicken Sie auf *Import*, um die Objekte in Zope zu importieren.

Zope bietet Ihnen die Möglichkeit, entweder mit "*Take ownership of imported objects*" das importierte Objekt in Besitz zu nehmen oder mit "*Retain existing ownership information*" vorhandene Besitz-Informationen beizubehalten. Das Thema Besitz wird in Kapitel 7, "Benutzer und Sicherheit" näher erörtert . Lassen Sie zunächst das Kontrollkästchen "*Take ownership of imported objects*" abgehakt.

Nachdem Sie die Datei importiert haben, sollten Sie ein neues Objekt im Zope-Ordner vorfinden, in dem Sie den Import durchgeführt haben.

Zum Üben wiederholen Sie dieselbe Export- und Import-Vorgehensweise. Beachten Sie, dass Sie kein Objekt mit derselben ID in einen Ordner importieren können, in dem bereits ein vorhandenes Objekt mit gleichem Namens liegt. Deshalb müssen Sie sie in einen Ordner importieren, der noch keinen Ordner *Heimarbeit* enthält.

Temporäre Ordner

Temporäre Ordner (Temporary Folders) sind Zope-Ordner die dazu benutzt werden zeitlich begrenzte Objekte zu speichern. Temporäre Ordner verhalten sich fast genauso wie normale Ordner mit drei wesentlichen Unterschieden:

1. Alles was sich in einem Temporäre Ordner befindet, verschwindet sobald man Zope neustartet.
2. Man kann keine Aktionen rückgängig machen, die auf in einem Temporären Ordner enthaltenen Objekte angewandt wurden.
3. Man kann keine Version verwenden, um Objekte in einem Temporäre Ordner zu verändern.

Standardmäßig gibt es einen Temporären Ordner Namens *temp_folder* in Ihrem Stammordner. Sie werden vielleicht bemerken, das sich in *temp_folder* ein Objekt Namens "Session Data Container" befindet. Dies ist ein Objekt, das von Zopes Standardkonfiguration der Sitzungsverwaltung benutzt wird. Für weitere Informationen zum Thema Sitzungen sehen Sie später im Abschnitt "Sitzungen verwenden" in diesem Kapitel nach.

Temporäre Ordner speichern ihren Inhalt eher im Arbeitsspeicher als in der Zope-Datenbank. Dies prädestiniert sie zum Speichern kleiner Objekte in die häufig geschrieben wird, wie Sitzungsdaten. Dennoch ist es eine schlechte Idee Temporäre Ordner zu verwenden, um große Objekte zu speichern, weil Ihrem Rechner dadurch möglicherweise nicht mehr genügend Arbeitsspeicher zur Verfügung steht.

Zope-Seitenvorlagen benutzen

In Zope 2.5,wurde ein neuer mächtiger Objekttyp Namens *Page Templates* ("Seitenvorlagen") eingeführt. Seitenvorlagen ermöglichen es, dynamische Darstellungen für eine Web-Seite zu definieren, indem man eine HTML-Vorlage schreibt. Der HTML-Code der Vorlage wird dadurch dynamisch, dass spezielle XML-Namensraum-Elemente in den HTML-Code eingefügt werden, die das dynamische Verhalten für die Seite definieren.

Seitenvorlagen sind aus mehreren Gründen sehr leistungssfähig:

- Sie sind immer gültiges HTML. Es muss kein ungültiger Code in die Vorlagen eingefügt werden, wie man es bei anderen dynamischen Sprachen machen würde.
- Seitenvorlagen trennen Logik von Darstellung. Da sie absichtlich ausschließlich auf die Darstellung abzielen, erlauben Ihnen Seitenvorlagen nicht sie als Allzweck-Programmiersprache zu verwenden.
- HTML-Designer müssen keine Programmierer sein. Weil Seitenvorlagen aus HTML-Entwürfen heraus entwickelt werden, brauchen Ihre HTML-Designer nichts über Programmierung wissen, um das anfängliche Design einer Seitenvorlage entwickeln zu können.
- Ihre Programmierer müssen keine HTML-Designer sein. Seitenvorlagen haben auch im umgekehrten Fall Vorteile, weil Ihre Programmierer Ihre Vorlagen dynamischen machen können, indem sie nur XML-Tag-Attribute hinzufügen. Sie können mit verschiedenen dynamischen Verhalten experimentieren ohne das ursprüngliche HTML zu zerstören oder neuzuschreiben.

Zope-Seitenvorlagen erstellen

Erzeugen Sie einen Ordner namens *Vertrieb* im Stammordner root. Klicken Sie auf den Ordner Verkäufe und wählen Sie dann *Page Template* aus der Produktauswahlliste aus, über die Sie in Kapitel 2 etwas erfahren haben. Dieses Vorgehen bringt Sie zum Erstellformular für eine Seitenvorlage. Tragen Sie als Id "Vertriebsseite" und als Titel "Vorlage für Vetriebsmitarbeiter" und klicken Sie auf *Add*. Sie haben erfolgreich eine Seitenvorlage erstellt. Jedoch ist deren Inhalt ein Standardblindtext. Fahren Sie also mit dem nächsten Schritt, der Bearbeitung des Inhalts, fort.

Zope-Seitenvorlagen bearbeiten

Der einfachste Weg eine Seitenvorlage zu bearbeiten ist, im Zope-Management-Interface auf ihren Namen oder ihr Symbol zu klicken. Wenn Sie auf eines dieser Elemente klicken, werden Sie zur *Edit*-Ansicht der Seitenvorlage weitergeleitet, die Ihnen einen Texteingabebereich zur Verfügung stellt in dem Sie die Vorlage bearbeiten können. Ersetzen Sie den ursprünglichen Inhalt, der bereits in der Seitenvorlage steht, mit dem folgenden HTML-Code:

```
<html>
  <body>
    <h1>Dies ist meine erste Seitenvorlage!</h1>
  </body>
</html>
```

und klicken Sie auf *Save*. Nun können Sie auf den *View*-Reiter klicken um sich die Seitenvorlage anzusehen. Diese Vorlage im Speziellen macht nichts besonderes und besitzt auch kein dynamisches Verhalten. In späteren Abschnitten dieses Kapitels, werden wir dynamisches Verhalten hinzufügen. In Kapitel 5, werden Sie Seitenvorlagen noch wesentlich ausführlicher verwenden um dynamische Darstellungen zu erstellen.

Zope-Seitenvorlagen hochladen

Angenommen Sie möchten Ihre HTML-Vorlagen lieber nicht in einem Web-Browser bearbeiten oder Sie haben schon vorhandene HTML-Seiten die Sie in Zope bringen möchten. Zope ermöglicht Ihnen Ihre vorhandenen HTML-Dateien hochzuladen und Sie in Seitenvorlagen umzuwandeln.

Wählen Sie *Page Template* aus der Produktauswahlliste. Dies bringt Sie zum Erstellformular, das wir vorher gesehen haben. Das letzte Formularelement im Erstellformular ist die Schaltfläche *Browse*. Klicken Sie auf die Schaltfläche. Ihr Browser öffnet daraufhin ein Dialogfenster zur Dateiauswahl. Wählen Sie die Textdatei auf Ihrem Rechner aus, die Sie in die Seitenvorlage hochladen möchten.

Geben Sie eine *Id* für das neue Dokument an und klicken Sie auf *Add*. Nachdem Sie auf *Add* geklickt haben, gelangen Sie wieder zurück zur Verwaltungsansicht. Hier sehen Sie Ihre neue Seitenvorlage.

Zope-Dokumente verwenden

Dokumente enthalten Text. In Webanwendungen verwendet man Dokumente im Allgemeinen, um Web-Seiten zu erstellen. Man kann Dokumente auch verwenden, um Textdateien, Textteile oder HTML-Code wie Sidebars oder Kopfzeilen zu speichern. Zusätzlich zum Speichern von Text erlaubt ein Dokument Ihnen, den Text über das Netz zu bearbeiten. Zope hat mehrere verschiedene Arten von Dokumenten. Das wichtigste ist DTML-Dokument. DTML steht für *Document Template Markup Language*.

Andere Objekttypen von Drittherstellern (im Allgemeinen "Produkte", engl. Products, genannt) und stehen auf Orten wie Zope.org zur Verfügung. Sie erweitern Ihre Installation dahingehend, dass andere Arten von textlichem und nicht-textlichem Inhalt unterstützt werden.

DTML-Dokumente

DTML-Dokumente verwendet man, um Web-Seiten und Dokumentenabschnitte, wie Sidebars zu erstellen, die Webseiten gemeinsam nutzen. DTML-Dokumente können Scripting-Befehle in DTML (Zopes tag-basierter Scripting-Sprache) enthalten. Die Mischung aus HTML und DTML erzeugt dynamische Web-Seiten.

DTML-Dokumente sind auch dazu geeignet, gemeinsam genutzten Inhalt, wie häufige Dokumentstrukturen, zu erstellen.

DTML-Dokumente erstellen

Klicken Sie auf den Ordner Vertrieb und wählen Sie danach *DTML Document* aus der Produktauswahlliste. Dieses Vorgehen bringt Sie zum Erstellformular für ein DTML-Dokument. Geben Sie die ID "Vetriebsmitarbeiter" und den Titel "Die Dschungel-Vertriebsmitarbeiter" an und klicken Sie auf *Add*. Sie haben erfolgreich ein DTML-Dokument erstellt. Jedoch ist dessen Inhalt ein Standardblindtext. Fahren Sie also mit dem nächsten Schritt, der Bearbeitung des Inhalts, fort.

DTML-Dokumente bearbeiten

Die einfachste und schnellste Art, ein DTML-Dokument zu bearbeiten, ist über das Management-Interface. Um ein Dokument auszuwählen, klicken Sie auf dessen Namen oder sein Symbol, was das in Abbildung 3-2 gezeigte Formular öffnet.

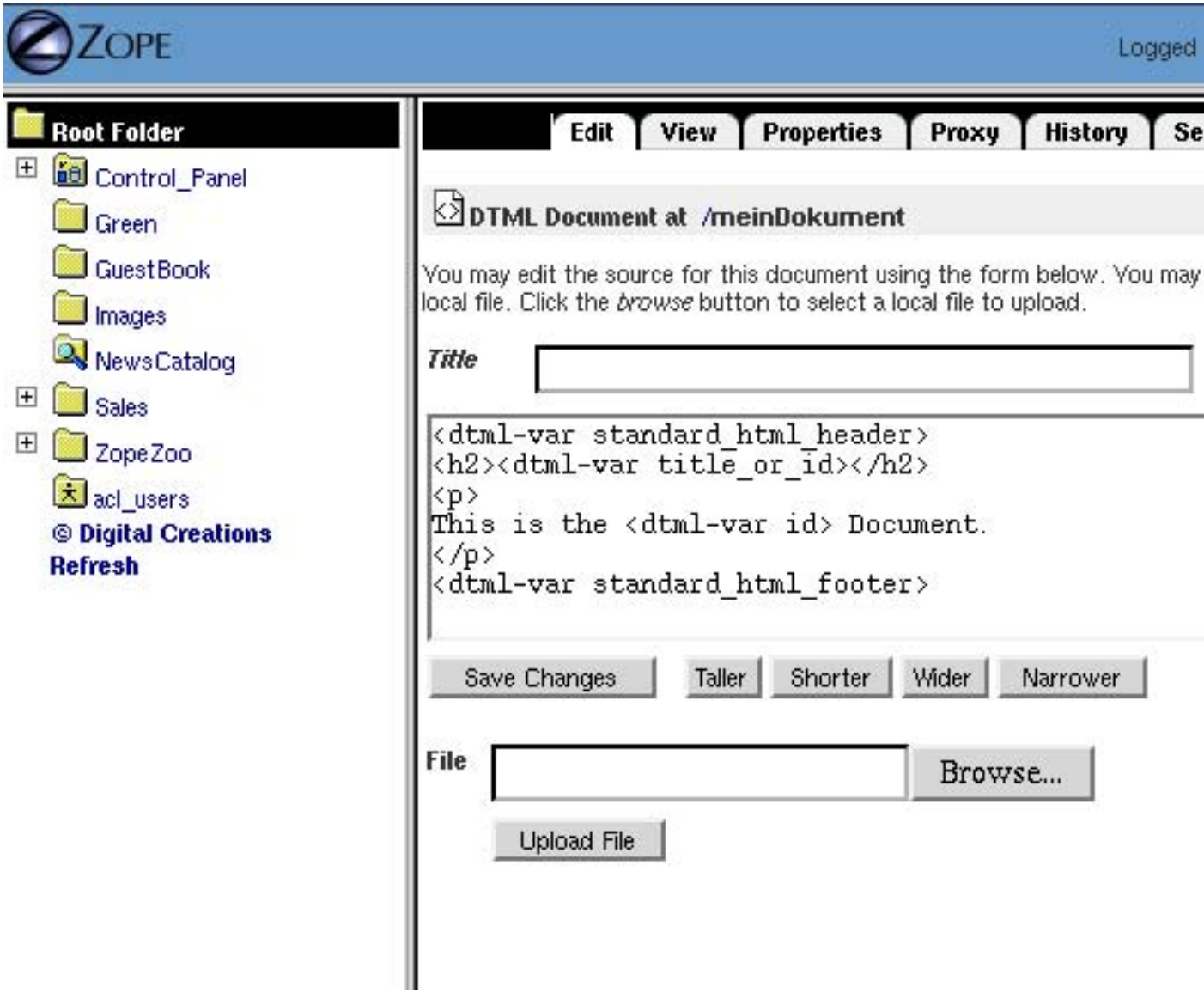


Abbildung 3-2 Bearbeiten eines DTML-Dokuments.

Diese Ansicht zeigt ein Textfeld, in welchem Sie den Inhalt Ihres Dokuments bearbeiten können. Wenn Sie auf die Schaltfläche *Change* klicken, bestätigen Sie alle Änderungen, die Sie im Textbereich vorgenommen haben. Sie können die Größe des Textbereichs mit den Schaltflächen *Taller* (höher), *Shorter* (kürzer), *Wider* (breiter) und *Narrower* (schmaler) kontrollieren. Sie können auch mit dem Eingabefeld *File* und der Schaltfläche *Upload File* eine neue Datei in das Dokument laden.

Löschen Sie den Standardinhalt, der automatisch im aktuellen DTML-Dokument *Vetriebsmitarbeiter* erscheint.

Fügen Sie in das *Vetriebsmitarbeiter*-Dokument folgenden HTML-Inhalt ein:

```
<html>
<body>
<h2>Die Dschungel-Vetriebsmitarbeiter</h2>

<ul>
  <li>Tarzan</li>
  <li>Cheetah</li>
  <li>Jane</li>
</ul>
</body>
</html>
```

Nachdem Sie die Änderungen an Ihrem Dokument abgeschlossen haben, klicken Sie auf die Schaltfläche *Change*. Zope gibt eine Nachricht zurück, die Ihnen mitteilt, dass Ihre Änderungen angewendet wurden. Jetzt können Sie das Dokument durch Klicken des *View*-Reiters ansehen.

Herzlichen Glückwunsch! Sie haben gerade Zope verwendet, um eine HTML-Seite zu erstellen. Sie können die Erstellung/Bearbeitung in einem Vorgang statt in zweien durchführen, indem Sie die Schaltfläche *Add and Edit* auf der Erstellseite verwenden.

Sie können Ihren HTML-Code online bearbeiten und ihn sofort betrachten. Sie können sogar ganze Zope-Sites mit HTML-Dokumenten und Ordnern erstellen. Dieser Vorgang zeigt Ihnen nur einen Teil der Vorteile von Zope, er ist aber eine gute Art, sich mit Zope vertraut zu machen. Sie können auch dynamischen Inhalt in Zope verfassen und können jene, die nur am Design interessiert sind, ihre eigenen HTML-Web-Seiten auf diese Weise bearbeiten lassen.

Eine HTML-Datei hochladen

Angenommen Sie ziehen es vor, Ihre HTML-Dateien nicht in einem Web-Browser zu bearbeiten, oder Sie haben bereits einige vorhandene HTML-Seiten, die Sie gerne in Zope bringen würden. Zope erlaubt Ihnen, Ihre vorhandenen Textdateien hochzuladen und sie in DTML-Dokumente umzuwandeln.

Wählen Sie *DTML Document* aus der Produktauswahlliste. Dies bringt Sie zum Erstellformular für DTML-Dokumente. Das letzte Formularfeld auf dem Erstellformular ist die Schaltfläche *Browse*. Klicken Sie auf diese Schaltfläche. Ihr Browser öffnet daraufhin

einen Dateiauswahldialog. Wählen Sie die Textdatei auf Ihrem Computer aus, die Sie in dieses Dokument laden wollen.

Geben Sie in eine *Id* für das neue Dokument und klicken Sie auf *Add*. Nach dem Klicken auf *Add* werden Sie zurück auf den Management-Seite gebracht. Dort sehen Sie Ihr neues Dokument.

DTML-Dokumente betrachten

Der Hauptverwendungszweck eines DTML-Dokuments ist es, nützlichen Inhalt zu speichern. Der Hauptverwendungszweck dieses Inhalt ist es, betrachtet zu werden. DTML-Dokumente können auf mehrere verschiedene Arten betrachtet werden:

Das Management-Interface
: Vom Management-Interface aus kann man auf den *View*-Reiter eines Dokuments klicken, um den Inhalt des Dokuments zu betrachten.

Direkter Aufruf über das Web
: Dokumente können direkt über das Web aufgerufen werden, indem man mit einem Web-Browser zu ihrem URL-Standort geht.

Aufruf durch ein anderes Objekt
: Andere Objekte, besonders andere DTML-Objekte, können den Inhalt eines Dokuments darstellen.

Über das Web aufrufen

Wie alle Zope-Objekte, basiert der URL eines DTML-Dokuments auf seiner ID-Kennung. Wenn man zum Beispiel ein DTML-Dokument namens *Bob* im Stammordner hat, dann wäre sein URL:

```
http://localhost:8080/Bob
```

Wenn sich *Bob* in einem *Onkel* genannten Unterordner befindet, dann wäre sein URL:

```
http://localhost:8080/Onkel/Bob
```

Es könnte auch andere DTML-Dokumente namens *Rick*, *Danny* und *Louis* im Onkel-Ordner geben. Auf sie greifen Sie durch das Web ebenso zu:

```
http://localhost:8080/Onkel/Rick
http://localhost:8080/Onkel/Danny
http://localhost:8080/Onkel/Louis
```

URLs in Objekte zu übersetzen, ist keine neue Idee, Web-Server wie Apache tun das die ganze Zeit über, sie übersetzen URLs in Dateien und Verzeichnisse auf einem Dateisystem. Zope führt diese einfache Idee weiter. In Zope sind URLs immer einfach zu lesen, weil sie

sich leicht und einfach auf die Art abbilden lassen, wie Objekte in Zope organisiert sind. Darum haben wir Ihnen gesagt, dass die Struktur Ihrer Site der Schlüssel für den Erfolg Ihrer Site ist.

Direkt zum URL eines DTML-Dokument zu gehen, wird *durch das Web aufrufen* genannt. Dies bewirkt, dass der Inhalt des DTML-Dokuments ausgewertet und an Ihren Web-Browser zurückgegeben wird. Im nächsten Kapitel über DTML sehen wir, was es bedeutet, wenn DTML ausgewertet wird, aber zunächst können Sie einfach mit DTML und simplem HTML-Inhalt experimentieren, um eine Vorstellung davon zu bekommen.

Aufruf durch ein anderes Objekt

Beim Verwenden von Zope sind Ihnen wahrscheinlich DTML-Beispiele wie dieses begegnet:

```
<dtml-var standard_html_header>

  <h1>Dies ist ein einfaches HTML-Dokument.</h1>

<dtml-var standard_html_footer>
```

Hier kann man sehen, dass ein DTML-Objekt, *standard_html_header*, von dem Dokument aufgerufen wird. In diesem Fall ist der ausgewertete Inhalt des ersten Dokuments in den Inhalt dieses aufrufenden Dokuments eingefügt worden. Dies ist ein sehr grundlegendes Konzept in Zope und wird überall in diesem Buch verwendet.

Änderungen an Dokumenten verfolgen

Der Undo-Reiter ermöglicht es Ihnen eine oder mehrere Transaktion rückgängig zu machen. Erinnern Sie sich: Eine Transaktion kann eine Gruppe von Aktionen sein, die alle zur selben Zeit getätigt wurden. Wenn ein Dokument in einer Transaktion bearbeitet wurde, die auch das Verschieben eines Objektes beinhaltete, kann es sein, dass Sie zwar die Änderung am Dokument, *nicht* aber das Verschieben der Datei widerrufen wollen. Um das zu tun, können Sie zur *History*-Ansicht des Objekts gehen und sich die vorherigen Zustände des Objekts ansehen, wie in Abbildung 3-4 gezeigt.

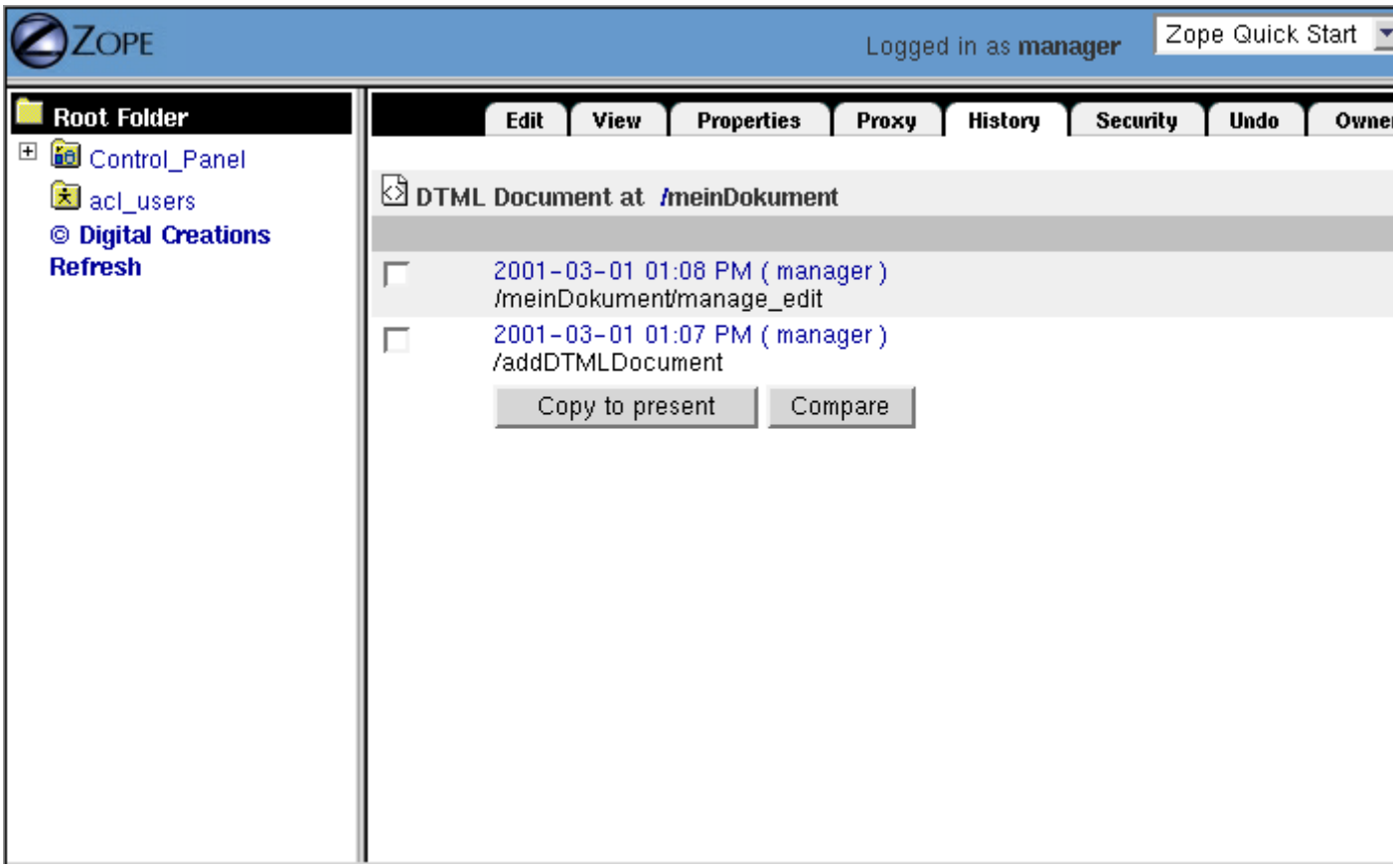


Abbildung 3-4 Die History-Ansicht.

Dokumente unterstützen sogar die Idee der vergleichenden Revisionen, und ermöglichen Ihnen somit, Änderungen an Ihren Objekten zu verfolgen. Zum Beispiel erlauben DTML-Methoden und -Dokumente Ihnen, zwei Überarbeitungen auszuwählen und sie mit einander zu vergleichen. Sie könnten das auch dazu verwenden, um zu sehen, welche Leute Ihr Objekt verändert haben. Lassen Sie uns zum Beispiel sagen, dass Sie ein Dokument hatten, das eine Liste aller der Tiere in einem Zoo enthielt. Wenn eine(r) Ihrer KollegInnen dann beginnt diese Liste zu bearbeiten und sie speichert, können Sie die Protokollvergleichsfunktion verwenden, um die aktuellste "neue" Version der Datei mit der nächsten aktuellsten Version zu vergleichen.

Dieser Vergleich wird im weitverbreiteten *diff*-Format angezeigt. Das Diff zeigt Ihnen die Zeilen an, die dem neuen Dokument hinzugefügt worden sind (über ein Plus), welche Zeile vom alten Dokument entfernt worden ist (über ein Minus), und welche Zeile ersetzt oder verändert worden ist (über ein Ausrufezeichen).

Fernbearbeitung mit FTP, WebDAV und PUT

Zope ermöglicht Ihnen Dokumente direkt in Ihrem Web-Browser zu bearbeiten, obwohl dies nicht die einzige Art ist, wie Dokumente in Zope bearbeitet werden können. Für einfache Dokumente ist das Bearbeiten über das Web eine praktische Methode. Aber für große, komplexe Dokumente oder Dokumente, die eine spezielle Formatierung haben, ist es nützlich, in der Lage zu sein, den Editor zu verwenden, mit Sie am vertrautesten sind.

DTML-Dokumente können per FTP, WebDAV und das HTTP-PUT-Protokoll bearbeitet werden. Viele HTML- und Texteditoren unterstützen diese Protokolle für das Bearbeiten von Dokumenten über entfernte Server. Jedes dieser Protokolle hat Vor- und Nachteile:

FTP
: FTP ist das "File Transfer Protocol" ("Dateitransferprotokoll"). FTP wird verwendet, um eine Datei von einem Rechner auf einen anderen zu übertragen. Viele Texteditoren unterstützen FTP, so dass es sehr hilfreich ist.

WebDAV
: WebDAV ist ein neues Internetprotokoll basierend auf dem, dem Web zugrunde liegenden Protokoll, HTTP. DAV steht für "Distributed Authoring and Versioning" ("Verteilte Bearbeitung und Versionsverwaltung"). Weil DAV neu ist, wird es wahrscheinlich nicht von so vielen Texteditoren unterstützt wie FTP.

PUT
: Das HTTP-Protokoll unterstützt einen einfachen Weg, Inhalte auf einen Server hochzuladen, genannt PUT. PUT wird von vielen HTML-Editoren wie dem Netscape Composer unterstützt.

Mit einer dieser Methoden können Sie Ihren Inhalt mit einer Vielfalt an Hilfsprogrammen bearbeiten. In den nächsten paar Abschnitten werden wir Ihnen einige einfache Hilfsprogramme zeigen, die FTP verwenden, um Zope zu bearbeiten.

Dokumente und Dateien mit WS_FTP hochladen

FTP ist ein beliebter FTP-Client für Windows, den Sie verwenden können, um Dokumente und Dateien in Zope über das FTP-Protokoll hochzuladen. WS_FTP kann von der Ipswitch-Homepage heruntergeladen werden.

Es gibt andere beliebte Windows-FTP-Clients und viele Web-Browser, wie Netscape und der Microsoft Internet Explorer besitzen FTP-Clients. Dieser Abschnitt gilt auch für andere FTP-Clients.

In Kapitel 2, "Zope verwenden", bestimmten Sie den URL Ihres Zope-Systems mit Hilfe des Startprotokolls. Um herauszufinden, wie man den FTP-Server Ihres Zopes anspricht, folgen wir einer ähnlichen Vorgehensweise:

```
        ------
        2000-08-07T23:00:53 INFO(0) ZServer Medusa (V1.18) started at Mon
Aug  7
  16:00:53 2000
                Hostname: erdnuss
                Port:8080

        ------
        2000-08-07T23:00:53 INFO(0) ZServer FTP server started at Mon Aug
7
  16:00:53 2000
                Authorizer:None
                Hostname: erdnuss
                Port: 8021
        ------
```

```
        2000-08-07T23:00:53 INFO(0) ZServer Monitor Server (V1.9) started
on port
  8099
```

Das Systemstartprotokoll sagt, dass der Zope-FTP-Server auf Port 8021 auf dem Rechner mit dem Namen *erdnuss* läuft. Wenn Sie WS_FTP starten, müssen Sie den Rechnernamen und Port-Informationen haben, so dass Sie sich an Zope mittels FTP anmelden können. Nach dem Eingeben des Rechnernamen und des Ports Ihres Zope-Servers klicken Sie auf die Schaltfläche Connect. WS_FTP bittet Sie jetzt um einen Benutzernamen und Kennwort. Geben Sie Ihren Verwaltungs-Benutzernamen und -Kennwort für das Zope-Management-Interface ein.

Wenn Sie Ihren Benutzernamen und Ihr Kennwort korrekt eingeben haben, zeigt Ihnen WS_FTP, wie Ihre Zope-Site in FTP aussieht. Es gibt Ordner und Dokumente, die genau denen entsprechen, die Sie in Ihrem Zope-Stammordner im Web sehen, wie in Abbildung 3-3 gezeigt.

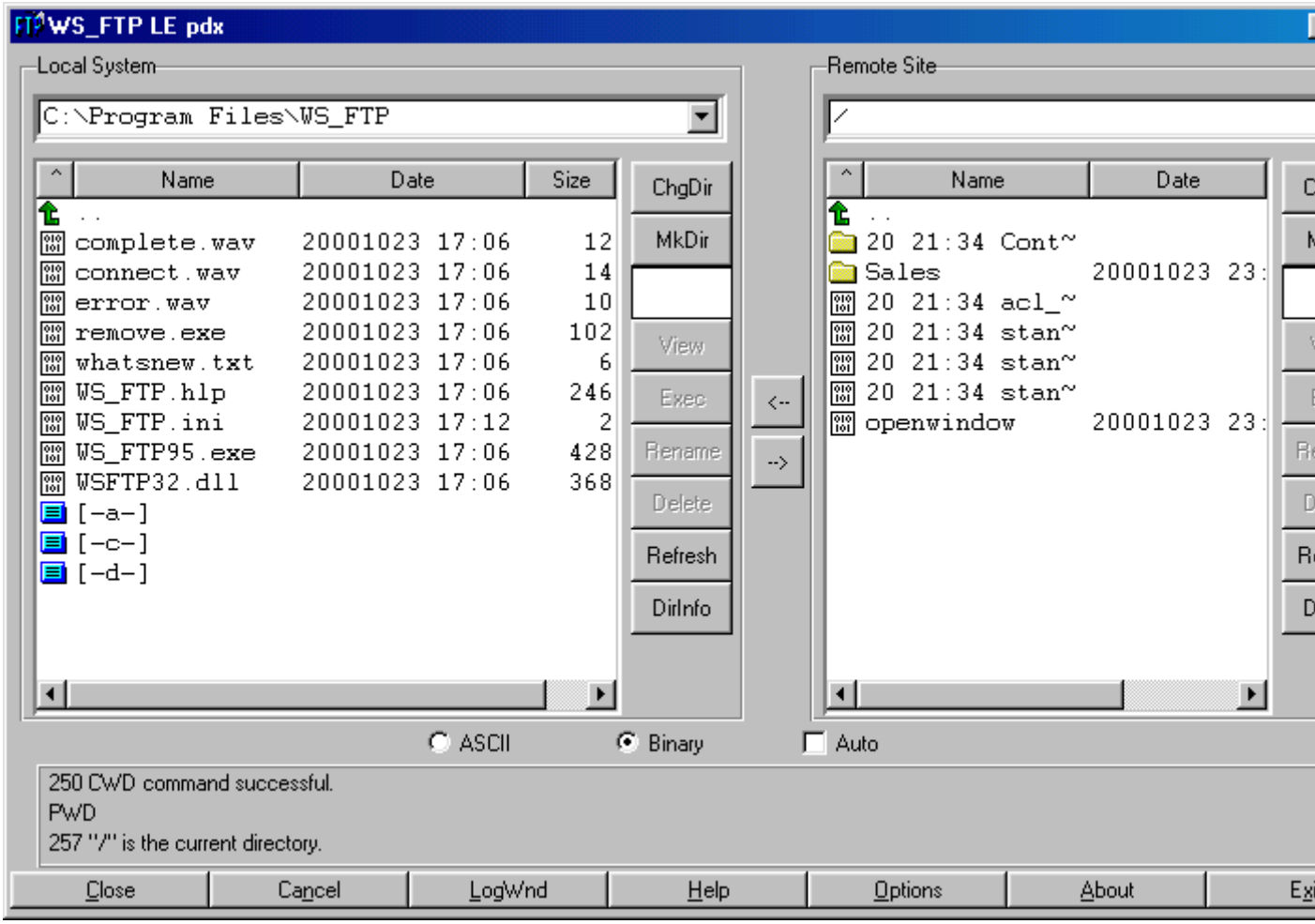


Abbildung 3-3 Bearbeiten von Zope per FTP.

Dateien zu und von Zope zu übertragen, ist eine sehr leichte Aufgabe mit WS_FTP. Auf der linken Seite des WS_FTP-Fensters ist eine Dateiauswahlliste, die die Dateien auf Ihrem

lokalen Rechner darstellt. Die Dateiauswahlliste auf der rechten Seite des WS_FTP-Fensters stellt die Objekte in Ihrem Zope-System dar. Dateien von Ihrem Computer auf Zope oder wieder zurück zu übertragen, ist leicht. Wählen Sie die Datei aus die Sie übertragen wollen und klicken Sie entweder auf den linken Pfeil (Herunterladen) oder auf den rechten Pfeil (Hochladen). WS_FTP hat viele Funktionen und Benutzereinstellungen, die Sie benutzen können, um die Objektverwaltung auf entfernten Rechnern mit Zope sehr einfach zu gestallten.

Zope-Objekte mit Emacs bearbeiten

Emacs ist ein sehr beliebter Texteditor. In der Tat ist Emacs mehr als nur ein Texteditor, er ist eine ganze Kultur. Emacs gibt es in zwei Varianten, GNU Emacs und XEmacs. Beide dieser Varianten von Emacs können direkt über FTP arbeiten, um Zope-Dokumente und andere Textinhalte zu verändern.

Emacs behandelt jedes entferne FTP-System wie jedes andere lokale Dateisystem, was die Fernverwaltung von Zope-Inhalten zu einem einfachen Vorgang werden lässt. Deshalb müssen Sie Emacs nicht verlassen, um Zope zu verwenden.

Emacs liefert einen größeren Umfang an Textbearbeitungsmöglichkeiten, als die meisten Web-Browser-Textfelder. Emacs kann verwendet werden, um Dokumente direkt zu bearbeiten und Objekte per FTP zu verändern, Emacs ist deshalb eine nette Zope-Entwicklungsumgebung.

Wenn Sie Zope starten, lässt Zope standardmäßig einen FTP-Server laufen, genau wie es einen HTTP-Server laufen lässt. Sie können zwar angeben, auf welchem Port der FTP-Server laufen soll wenn Sie Zope starten, standardmäßig ist dieser Port jedoch 8021.

Um sich in Zope anzumelden, starten Sie Emacs. Die Datei die Sie aufrufen, um eine FTP-Verbindung herzustellen, hängt davon ab, welchen Texteditor Sie benutzen: XEmacs oder Emacs:

Xemacs
: Um eine entfernte Datei in XEmacs zu öffnen, rufen Sie eine Datei folgendermaßen auf:

```
/user@server#port:/
```

Dies öffnet eine Verbindung zum Ordner / des FTP-Servers der auf dem Server *server* und Port *port* läuft.

Emacs
: Um eine entfernte Datei in Emacs zu öffnen, rufen Sie eine Datei folgendermaßen auf:

```
/user@server port:/
```

Das Leerzeichen wird eingefügt, indem man die Strg-Taste und die Taste Q gedrückt hält und danach die Leertaste drückt "Strg-Q ".

Bei einer typischen Zope-Installation ist der Dateiname mit dem in XEmacs eine FTP-Sitzung mit Zope zu öffnen */user@localhost#8021:/*.

Emacs fragt Sie nach einem Kennwort für die Anmeldung unter Zopes FTP-Server.

Wenn Sie den Ordner / eines FTP-Server in Zope betrachten, listet Emacs den Inhalt des Stammordners auf:

```
        drwxrwx---   1 Zope     Zope            0 Dec 30  1998
Control_Panel
        drwxrwx---   1 Zope     Zope            0 Dec 30  1998 QuickStart
        drwxrwx---   1 Zope     Zope            0 Dec 30  1998 Vetrieb
        -rw-rw----   1 Zope     Zope         1024 May  3  1999 index_html
        -rw-rw----   1 Zope     Zope         1381 May  3  1999
  standard_error_message
        -rw-rw----   1 Zope     Zope           55 Dec 30  1998
  standard_html_footer
        -rw-rw----   1 Zope     Zope           81 Dec 30  1998
  standard_html_header
```

Sie können alle diese "Dateien" (die in Wirklichkeit Zope-Objekte sind) öffnen, indem Sie sie auf die übliche Emacs-Weise auswählen. Mit Emacs zu arbeiten ist sehr nützlich, aber für den größten Teil ist Emacs ein sehr komplexes Programm, das den meisten Leuten nicht gerade sehr zugänglich ist. Die meisten Macintosh-Nutzer wären zum Beispiel mit einem Hilfsprogramm wie Emacs nicht sehr vertraut. Es gibt viele "einfachere" Editoren, die benutzt werden können, die auch FTP und WebDAV verwenden. Tatsächlich ist WebDAV dafür entwickelt worden, mit Hilfsprogrammen wie Adobes GoLive und Macromedias Dreamweaver verwendet zu werden.

DTML-Dokumente mit WebDAV bearbeiten

WebDAV ist ein, verglichen mit HTTP oder FTP, neueres Internetprotokoll, so dass es weniger Clients gibt, die es unterstützen. Es herrscht jedoch viel Bewegung in der WebDAV-Bewegung, und immer mehr Clients werden entwickelt. Für weitere Informationen darüber, welche Programme das WebDAV-Protokoll unterstützen, sehen Sie auf der WebDAV-Homepage nach.

WebDAV ist eine Erweiterung des HTTP-Protokolls, das viele Funktionen für das gleichzeitige Verfassen und Bearbeiten von Inhalten durch mehre Benutzer bereitstellt. WebDAV bietet Funktionen wie die Sperrung von Dateien, Versionsverwaltung und das Versehen von Dokumenten oder Objekten mit Eigenschaften. Weil die Ziele von WebDAV des Über-Das-Netz-Bearbeitens mit einigen der Ziele von Zope übereinstimmen, unterstützt Zope das WebDAV-Protokoll schon seit einiger Zeit.

Das WebDAV-Protokoll entwickelt sich schnell, und immer wieder werden neue Funktionen hinzugefügt. Sie können jeden WebDAV-Client verwenden, um Ihre DTML-Dokumente durch einfaches Verweisen des Clients auf den URL Ihres Dokuments zu bearbeiten. Die meisten Clients veranlasst dies jedoch, zu versuchen, das *Ergebnis* der Dokument-Darstellung zu bearbeiten, nicht den *Quelltext*. Für Dokumente, die Zopes Vorlagensprache DTML verwenden, um dynamischen Inhalt wiederzugeben, kann dies ein Problem sein.

Solange Clients nicht den neusten WebDAV-Standard unterstützen und den Unterschied zwischen dem Quelltext eines Dokuments und seines Darstellungsergebnisses verstehen, bietet Zope einen speziellen HTTP-Server an, den Sie mit einer Befehlszeilenoption `-W` aktivieren können. Dieser Server läuft auf einem anderen Port als Ihr normaler HTTP-Server und gibt anderen, speziellen Quelltextinhalt für WebDAV-Anforderungen zurück, die diesen Port ansprechen. Dies ist ein fortgeschrittenes Merkmal und wird tiefgehender im Documentation-Bereich von Zope.org erklärt.

Zope-Dateien verwenden

Zope-Dateien (Files) enthalten Rohdaten, genau wie die Dateien auf Ihren Computer. Viele Informationen wie Programme, Audio-Informationen, Videos und Dokumente werden im Internet und um die Welt als Dateien transportiert. Sie können Dateien verwenden, um jede Art von Information zu speichern, die Zope nicht ausdrücklich unterstützt, wie Flash-Dateien, Applets, Tarball-Archive usw.

Dateien gehen nicht davon aus, dass ihr Inhalt in einem speziellen Format vorliegt, textuell oder andersgeartet. Dateien sind gut dafür geeignet, jede Art von *binärem Inhalt* zu speichern, der auf irgendeine Art nur rohe Computerinformation ist. Dateien sind auch dafür geeignet um textuellen Inhalt zu speichern, der kein DTML-Scripting benötigt.

Jedes Datei-Objekt hat einen bestimmten *Inhaltstyp* (Content Type), der eine Standard-Internet-MIME-Kennzeichnung für einen Dateityp ist. Wenn Sie eine Datei in Zope hochladen, versucht Zope, den Inhaltstyp aufgrund des Namens der Datei zu erraten, aber Zope rät nicht immer richtig.

Dateien hochladen

Wie auch DTML-Dokumente und -Methoden erlauben Dateien Ihnen, eine Datei von Ihrem Computer hochzuladen, wenn Sie ein neues Objekt erstellen. Klicken Sie auf die Schaltfläche *Browse*, um eine Datei von Ihrem lokalen Computer beim Erstellen eines neuen Zope-Files zu wählen. Versuchen Sie, eine Datei, etwa eine Word-Datei (.doc) oder eine Date im "Portable Document Format" (.pdf) zu wählen. Beachten Sie, dass, wenn Sie eine Datei mit Ihrem Browser hochladen, Sie eventuell den Dateityp nach dem Sie suchen im Hochlade-Dialog Ihres Browsers angeben müssen. Nach Sie eine Datei ausgewählt haben, die hochgeladen werden soll, klicken Sie *Add*. Je nach der Größe der Datei, die Sie hochladen wollen, kann es einige Minuten dauern, bis Zope die Datei hinzufügt.

Nach dem Hinzufügen der Datei klicken Sie auf die neue Datei und betrachten Sie seine *Edit*-Ansicht. Hier sehen Sie, dass Zope den Inhaltstyp erraten hat, wie in Abbildung 3-5 gezeigt.

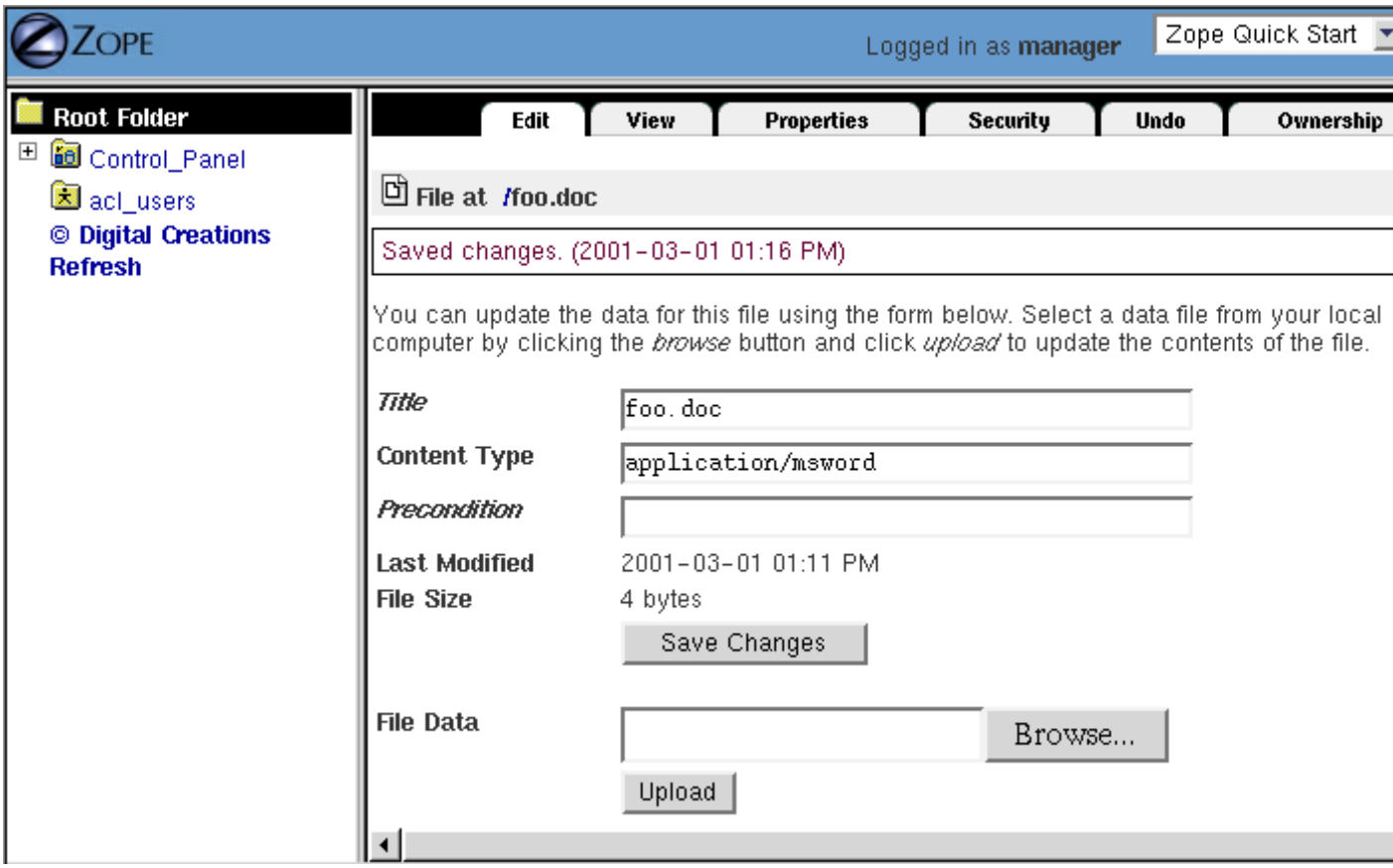


Abbildung 3-5 Die Eigenschaft "Content Type"

Wenn Sie ein Word-Dokument hinzufügen, ist der Inhaltstyp *application/msword*. Wenn Sie eine PDF-Datei hinzufügen, ist der Inhaltstyp *application/pdf*. Wenn Zope den Dateityp nicht erkennt, wählt es den Standardwert, die allgemeine Inhaltsart *application/octet-stream*.

Sie können den Inhalt eines vorhandenen Files über das Öffnen der Ansicht *Upload* ändern. Hier können Sie den Inhalt der Datei durch eine neue Datei ersetzen. Wenn Sie die Felder Id und Title in diesem Formular nicht ausfüllen und Sie eine Datei hochladen, wird Zope den Dateinamen als Kennung und Titel des Objekts verwenden.

Dateien bearbeiten

Falls Ihre Datei Text enthält und kleiner als 64kB ist, können Sie sie mit dem Management-Interface bearbeiten. Text-Dateien sind Dateien deren Inhaltstyp mit *text/* beginnt, wie *text/html* oder *text/plain*. Manchmal ist es bequemer Textdateien über das Management-Interface zu verändern. Nichtsdestotrotz können Sie Dateien auch lokal bearbeiten und Sie dann in Zope hochladen.

Dateien betrachten

Sie können eine Datei durch Öffnen des Reiters *View* im Management-Interface betrachten. Sie können eine Datei auch durch aufrufen seines URLs ansehen. In Wirklichkeit ist der Reiter *View* nur eine Methode vom Zope-Management-Interface aus zum URL einer Datei zu

kommen. Wenn Sie zum Beispiel eine Datei namens *mitarbeiterVertrag.pdf* in Ihrem Zope-Stammordner haben, dann können Sie diese Datei in Ihrem Web-Browser durch aufrufen des URL *http://localhost:8080/mitarbeiterVertrag.pdf* betrachten. Je nach Art der Datei kann Ihr Web-Browser die Datei anzeigen oder sie herunterladen.

Zope-Bilder verwenden

Bilder (Images) stellen Grafiken wie GIF-, JPEG- und PNG-Dateien dar. In Zope sind Bilder ähnlich wie Datei-Objekte, beinhalten aber zusätzliche Funktionen für das Verwalten grafischen Inhalts.

Bildobjekte haben dasselbe Management-Interface wie Dateiobjekte. Alles aus dem vorherigen Abschnitt, wie man File-Objekte verwendet gilt auch für Images. Bildobjekte zeigen Ihnen jedoch eine Vorschau des Bildes wenn Sie sie hochladen.

Viewing Images with HTML

Die häufigste Verwendung für Bilder in Zope ist das einbinden von Bildern in Web-Seiten. Um ein Bild auf eine Web-Seite zu bringen, müssen Sie das HTML-*IMG*-TAG verwenden. Angenommen, Sie haben ein Bild namens *logo* in Ihrem Stammordner, das eine Abbildung Ihres Firmenlogos enthält.

Dieses Bild in HTML-Code zu verwenden, ist ein einfacher Vorgang: Sie können auf es mit einem *IMG*-Tag verweisen, wie Sie es auch tun würden, um jede Art von Bildern in eine Web-Seite einzubinden:

```
<dtml-var standard_html_header>

  <img src="logo">

  <h1>Willkommen!</h1>

<dtml-var standard_html_footer>
```

In diesem Beispiel verweisen Sie auf das Bild logo durch Erstellen eines HTML-*IMG*-TAGs. Es ist aber normalerweise nicht notwendig, Ihre eigenen *IMG*-Tags zu erstellen, um Bilder anzuzeigen. Bildobjekte wissen, wie sie ihre eigenen HTML-Tags generieren müssen. Wenn Sie ein Bildobjekt in DTML einfügen, generiert es ein *IMG*-Tag für sich selbst.

Jetzt wollen wir, dass dieses Logo auf jeder Seite in der linken oberen Ecke erscheint, also erstellen wir einen Verweis darauf in der Methode *standard_html_header*:

```
<html>
  <body>
    <dtml-var logo>
```

Betrachten Sie jetzt den Stammordner indem Sie auf dessen *View*-Reiter klicken. Wenn Sie den Quelltext der Web-Seite ansehen, die Zope erstellt, können Sie erkennen, dass der `var`-DTML-Code für Sie in ein HTML-*IMG*-TAG umgewandelt wurde:

```
<html>
  <body>
    <img src="logo" width="50"  height="30">
```

Den DTML-Befehl *var* zu verwenden, um Bilder darszustellen, macht die Dinge einfacher, da Zope die Werte der height- und width-Attribute des *IMG*-Tags für Sie automatisch herausfindet. Wenn es Ihnen nicht gefällt, wie Zope ein *IMG*-Tag erzeugt, kann es angepasst werden. Konsultieren Sie Anhang B für weitere Informationen über das Image-Objekt und wie es das *IMG*-Tag steuern kann.

Es gibt eine Reihe von Zope-Objekttypen von Drittherstellern (allgemein "Produkte" genannt) zum Speichern und Betrachten von Bildinhalt im Abschnitt "visual" von Zope.org.

Bilder über das Netz betrachten

Bilder können direkt durch das Besuchen ihres URL in Ihrem Web-Browser betrachtet werden. Lassen Sie uns beispielsweise annehmen, dass Sie Ihr Firmenlogo direkt sehen wollen. Das Logo existiert als bildobjekt in Ihrem Stammordner. Es heißt *logo*. Sie können es leicht betrachten, indem Sie direkt zu seinem URL *http://localhost:8080/logo* gehen.

Da Zope-Bilder genauso wie auf einem normalen Web-Server gespeicherte Bilder funktionieren, können Sie auf Ihre Zope-Bilder von anderen Web-Servern zugreifen. Einmal angenommen Sie haben ein Zope-Bild dessen URL *http://imageserver:8080/Voegel/Sittich.jpg* lautet, so können Sie dieses Bild in jede Web-Seite auf jedem Web-Server mit Hilfe des absoluten URLs des Bildes einbinden:

```
<html>

<h1>Remote Image</h1>

<img src="http://imageserver:8080/Voegel/Sittich.jpg">

</html>
```

Dieses Beispiel zeigt, wie Sie Zope-Daten außerhalb von Zope verwenden können, indem Sie Standard-Internet-Protokolle verwenden. Später, in Kapitel 10 "Zope-Scripting für Fortgeschrittene", sehen wir, wie Sie die meisten Zope-Objekte Dienste der Außenwelt anbieten können.

Objekteigenschaften verwenden

Eigenschaften (Properties) sind eine Möglichkeit Informationen mit Objekten in Zope zu verknüpfen. Viele Zope-Objekte, einschließlich Ordner und Dokumente, unterstützen Eigenschaften. Eigenschaften können ein Objekt kennzeichnen, um seinen Inhalt zu identifizieren, (viele Zope-Inhalts-Objekte haben eine Inhaltstyp-Eigenschaft namens Content-Type). Eine andere Verwendung von Eigenschaften ist das Bereitstellen von Metadaten für ein Objekt wie seinen Autor, seinen Titel, seinen Status, usw.

Eigenschaften können komplexer als Zeichenfolgen sein. Sie können auch Nummern, Listen oder andere Datenstrukturen sein. Alle Eigenschaften werden über die Ansicht *Properties* verwaltet. Klicken auf den Reiter *Properties* eines Ordners und Sie kommen zur Ansicht für die Eigenschaftenverwaltung (wie in Abbildung 3-6 gezeigt).

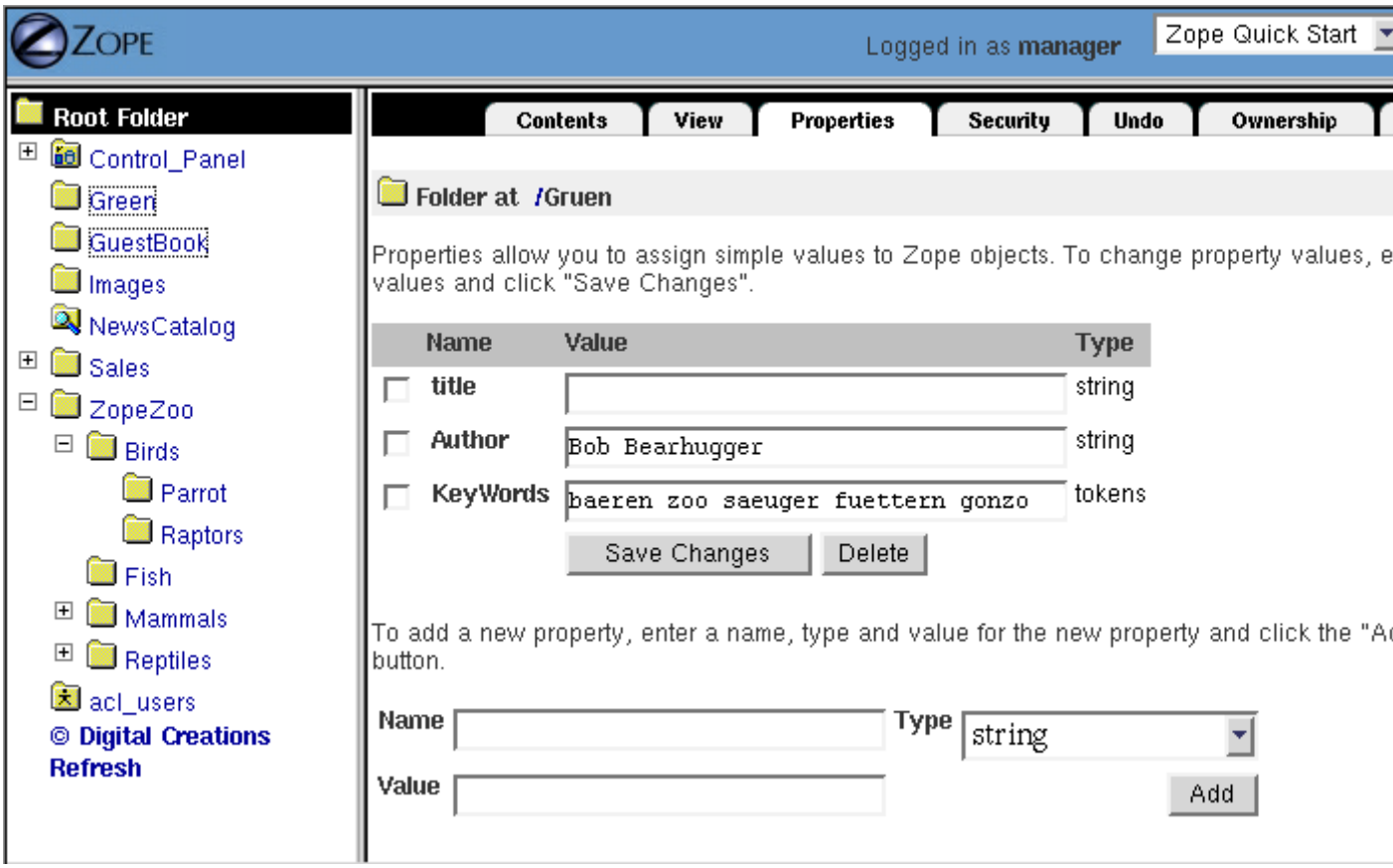


Abbildung 3-6: Die Ansicht zum Verwalten von Eigenschaften

Eine Eigenschaft besteht aus einem Namen, einem Wert und einem Typ. Der Typ einer Eigenschaft definiert, welche Werte sie haben kann.

In Abbildung 3-6 können Sie sehen, dass der Ordner drei Eigenschaften hat: *title*, *Author* und *KeyWords*. Die Eigenschaften *title* und *Author* *string*-Eigenschaften sind (Zeichenfolgen), während die Eigenschaft *KeyWords* vom Typ *tokens* ist. Eine *tokens*-Eigenschaft entspricht einer Folge von Wörtern.

Zope unterstützt eine Vielzahl von Eigenschaftstypen. Jeder Typ ist für eine bestimmte Aufgabe geeignet. Diese Liste gibt einen kurzen Überblick über die Arten von Eigenschaften, die Sie im Management-Interface erstellen können:

string (Zeichenfolge)
: Ein String ist eine Folge von Zeichen in beliebiger Länge. Strings sind die am grundlegendsten und nützlichsten Eigenschaften in Zope..

int (Ganzzahl)

Eine Int-Eigenschaft ist eine Ganzzahl, das jede positive oder negative Zahl annehmen kann, die keine Nachkommastellen hat ist. Ein Int hat garantiert eine Genauigkeit von mindestens 32 Bits.

long (lange Ganzzahl)
: Ein Long ist wie eine Ganzzahl, das keine Bereichseinschränkung hat.

float (Fließkommazahl)
: Ein Float speichert eine Fließkomma- oder Dezimalzahl. Für monetäre (geldliche) Werte wird zum Beispiel oft Float verwendet.

lines (Zeilen)
: Eine Lines-Eigenschaft ist eine Folge von Zeichenfolgen.

tokens
: Eine Token-Eigenschaft ist eine Liste von durch Leerzeichen getrennten Wörtern.

text
: Eine Text-Eigenschaft ist in etwa wie eine String-Eigenschaft, außer dass Zope die Zeilenumbruch-Zeichen anpasst (verschiedene Browser verwenden verschiedene Zeilenumbruchskonventionen).

selection (Auswahl)
: Ein Selection-Eigenschaft ist etwas spezielles. Sie wird verwendet, um ein HTML-Eingabefeld für eine Auswahlliste auszugeben.

multiple selection (Mehrfachauswahl)
: Eine Multiple-Selection-Eigenschaft ist etwas besonderes. Sie wird verwendet, um ein HTML-Eingabefeld für eine Mehrfachauswahlliste auszugeben.

Eigenschaften sind sehr nützliche Hilfsmittel für das Kennzeichnen Ihrer Zope-Objekte mit kleinen Einheiten von Daten oder Information. In Verbindung mit Methoden und Skripten machen Eigenschaften das Erweitern einfacher Objekte, wie Ordner, zu einer sehr leistungsfähigen Methode.

Logik mit Skripten programmieren

In traditioneller Programmiererssprache ist ein *Skript* ein in einer Programmiersprache geschriebenes kurzes Stück Quelltext. Seit Version 2.3 bietet Zope zwei Arten von Skriptobjekten: eine, mit der Sie Skripte in Python schreiben können, und eine, mit der Sie Skripte in Perl schreiben können.

Sowohl Python als auch Perl sind sehr beliebte und leistungsfähige Programmiersprachen. Sowohl Python als auch Perl haben gemeinsame, ähnliche Merkmale: Sie bieten beide eine leistungsfähige, rasche Entwicklung, eine einfache Syntax, viele Add-On-Bibliotheken, eine starke Unterstützung durch ihre Benutzergemeinde und ein reichhaltiges Angebot an kostenloser Online-Dokumentation. Beide Sprachen sind auch Open-Source.

Weil Skripte so leistungsfähig und flexibel sind, sind ihre Anwendungsmöglichkeiten endlos. Skripte werden in erster Linie verwendet, um so genannte *Geschäftslogik* zu schreiben. Geschäftslogik ist etwas anderes als Darstellungslogik. Darstellungslogik wird normalerweise in einer Präsentationssprache wie DTML geschrieben, und ihr Zweck ist es Informationen einem Benutzer zu präsentieren. Geschäftslogik wird normalerweise in eine Scripting-Sprache geschrieben, und ihr Zweck ist es Informationen zu verändern, die aus Inhaltsquellen (wie Dokumenten oder Datenbanken) stammt, oder andere Objekte zu manipulieren. Darstellungslogik baut oft *auf* der Geschäftslogik auf.

Ein einfaches Beispiel für die Verwendung von Skripten ist die Erstellung eines Online-Webformulars, um Ihren Benutzern zu helfen, den Zinseszinsbetrag ihrer Schulden zu berechnen. Diese Art von Berechnung schließt folgende Prozedur ein:

1. Sie brauchen folgende Informationen: Ihren aktuellen Saldo (oder Ihre Schulden) - genannt "kapital", den Jahreszinssatz als Dezimalzahl (beispielsweise 0,095) - genannt "zinssatz", die Zahl der Zeitpunkte in einem Jahr zu denen verzinst wird (normalerweise monatlich) - genannt "perioden" und die Anzahl an Jahren ab jetzt, die Sie berechnen wollen - genannt "jahre".
2. Dividieren Sie Ihren "zinssatz" durch "perioden" (normalerweise 12). Wir nennen dieses Ergebnis "i".
3. Nehmen Sie "perioden" und multiplizieren Sie es mit "jahre". Wir nennen dieses Ergebnis "n".
4. Potenzieren Sie (1 + "i") mit dem Exponenten "n".
5. Multiplizieren Sie das Ergebnis mit Ihrem "kapital". Dies ist der neue Saldo (oder die neuen Schulden).

Für dieses Beispiel brauchen Sie zwei Seitenvorlagen, *zinsFormular* und *zinsDarstellung* genannt, um die Informationen vom Benutzer abzufragen beziehungsweise sie darzustellen. Sie brauchen auch ein *berechneZineseszins* genanntes Python-basiertes Skript, das die tatsächliche Berechnung durchführt. Der erste Schritt ist, ein Web-Formular in *zinsFormular* zu erstellen, das "kapital", "zinssatz", "perioden" und "jahre" Ihrer Benutzer abfragt. Hier ist eine beispielhafte Seitenvorlage *zinsFormular*:

```
<html>
  <body>

  <form action="zinsDarstellung" method="POST">
  <p>Bitte geben Sie die folgende Informationen ein:</p>

  Ihr aktueller Saldo (oder Schulden): <input
name="kapital:float"><br>
  Ihr Jahreszinssatz: <input name="zinssatz:float"><br>
  Verzinsungsperioden pro Jahr: <input name="perioden:int"><br>
  Anzahl der Jahre: <input name="jahre:int"><br>
  <input type="submit" value=" Berechnen "><br>
  </form>

  </body>
</html>
```

Dieses Formular sammelt Informationen und ruft die Methode *zinsDarstellung* auf. Erstellen Sie jetzt ein *berechneZineseszins* genanntes Python-basiertes Skript, das die vier Parameter "kapital", "zinssatz", "perioden" und "jahre" aufnimmt, mit folgendem Python-Code:

```
## Script (Python) "berechneZineseszins"
## Parameter = kapital, zinssatz, perioden, jahre
##
"""
Zinseszinsrechnung
```

```
    """
    i = zinssatz / perioden
    n = perioden * jahre
    return ((1 + i) ** n) * kapital
```

Geben Sie die Parameter in das Feld *Parameters List* und den Code in das Haupttextfeld ein. Die Kommentare die am Anfang des Codes stehen sind beim Bearbeiten über das Web nicht nötig. (Sie sind jedoch hilfsreich beim Bearbeiten von Skripten über FTP.)

Dies gibt den Saldo oder die Schulden zurück, der über Zeitraum "jahre" verzinst wurde. Erstellen Sie danach eine Seitenvorlage namens *zinsDarstellung*, die *berechneZineseszins* aufruft und das Ergebnis zurückgibt:

```
    <html>
      <body>

      <p>Ihr Gesamtsaldo (oder Ihre Gesamtschulden) einschließlich
Zinseszins über
      <span tal:content="request/jahre">2</span> Jahre beträgt:</p>
      <p><b><span tal:content="python:
here.berechneZineseszins(request.kapital,

request.zinssatz,

request.perioden,
                                                          request.jahre)"
>1,00</span> &euro;</b></p>

      </body>
    </html>
```

Rufen Sie zuerst die Seitenvorlage *zinsFormular* ab. Geben Sie jetzt einige Information über Ihren Saldo oder Ihre Schulden ein und klicken Sie auf *Berechnen*. Dies bewirkt, dass *zinsFormular* die gesammelten Informationen an *zinsDarstellung* schickt, das wiederum das Python-basierte Skript *berechneZineseszins* aufruft. Die Darstellungs-Methode verwendet den vom Skript zurückgegebenen Wert bei der entstehenden Darstellung.

Wie bereits früher erwähnt, sind die Möglichkeiten, Skripte zu verwenden, fast endlos. Dieses Beispiel gibt Ihnen jedoch einen guten Eindruck vom allgemeinsten Anwendungsmuster für *Darstellungs*-Objekte Informationen abzufragen und anzuzeigen und das Verwenden von *Geschäftslogik*-Objekten um Berechnungen durchzuführen.

Methoden verwenden

Methoden sind Objekte in Zope, die speziellen ausführbaren Inhalt enthalten. Das Wort "Methode" ist eigentlich die falschen Bezeichnung und seine Verwendung in Zope nimmt langsam ab zugunsten gebräuchlicherer Ausdrücke wie *Skript* und *Vorlage*.

Zope bietet zwei Arten von Methoden: DTML-Methoden und SQL-Methoden. DTML-Methoden werden verwendet, um *Darstellungsvorlagen* zu definieren, die auf Inhaltsobjekte wie DTML-Dokumente und Dateien angewendet werden können. Eine sehr häufige und

beliebte Art, DTML-Methoden zu verwenden, ist das Definieren von Darstellungsvorlagen separat von Ihrem Inhalt.

SQL-Methoden werden verwendet, um Datenbankabfragen zu speichern, die Sie überall in Ihrer Webanwendung wieder verwenden können. SQL-Methoden werden in Kapitel 12 "Verbindung mit relationalen Datenbanken" erklärt, wo ein Beispiel für das Erstellen einer Webanwendung mit Hilfe einer relationalen Datenbank, gegeben wird.

Alle die verschiedenen Objekte in Zope können durch das Anwenden von Methoden auf diese Objekte verändert werden. Beispielsweise besitzen Ordner-Objekte eine `objectValues`-Methode die die im Ordner enthaltenen Objekte zurückgibt. DTML-Methoden können verwendet werden, um einfache Skripte zu schreiben, die diese Zope-API-Methoden aufrufen. Diese Methoden sind im Hilfesystem unter "API Documentation" dokumentiert.

Vor Zope 2.3 waren DTML-Methoden die einzige Art, Skripte mit Ihrem Web-Browser in Zope zu schreiben. Während DTML für sehr einfache Skripte und die Darstellung von Information mittels Vorlagen brauchbar ist, hatte dieser Ansatz viele Probleme, weil DTML nicht so flexibel wie andere Programmiersprachen ist.

Zope 2.3 führte zwei neue Arten von *Skript*-Objekten basierend auf zwei sehr beliebten Programmiersprachen, Python (in dem Zope geschrieben ist) und Perl ein. Sie sollten Python und Perl-basierte Skripte für ko mplexere Skripte verwenden, statt eine DTML-Methode zu schreiben. Während Sie durch die ältere Zope-Dokumentationen, Mailinglisten-Archive und andere Ressourcen auf Zope.org blättern, können Sie viele Verweise auf sehr komplexe DTML-Skripte finden, die vor den Python und Perl-basierten Skripte verwendet wurden. Im Allgemeinen sollten komplexe Skripte entweder in Python oder Perl geschrieben werden. Python und Perl-basierte Skripte werden später in diesem Kapitel beschrieben und in Kapitel 10 "Zope-Scripting für Fortgeschrittene" werden viele Beispiele für ihren Einsatz gegeben.

Ein einfaches Beispiel dafür, wie man DTML-Methoden verwendet, ist das Erstellen einer DTML-Methode Namens *objektListe* im Stammordner:

```
<dtml-var standard_html_header>

<ul>
  <dtml-in objectValues>
    <li><dtml-var getId></li>
  </dtml-in>
</ul>

<dtml-var standard_html_footer>
```

Wenn Sie diese Methode betrachten, ruft sie die *objectValues*-Methode im Stammordner auf und zeigt Ihnen eine einfache HTML-Liste aller der Objekte im Stammordner an, wie in Abbildung 3-7 gezeigt wird.

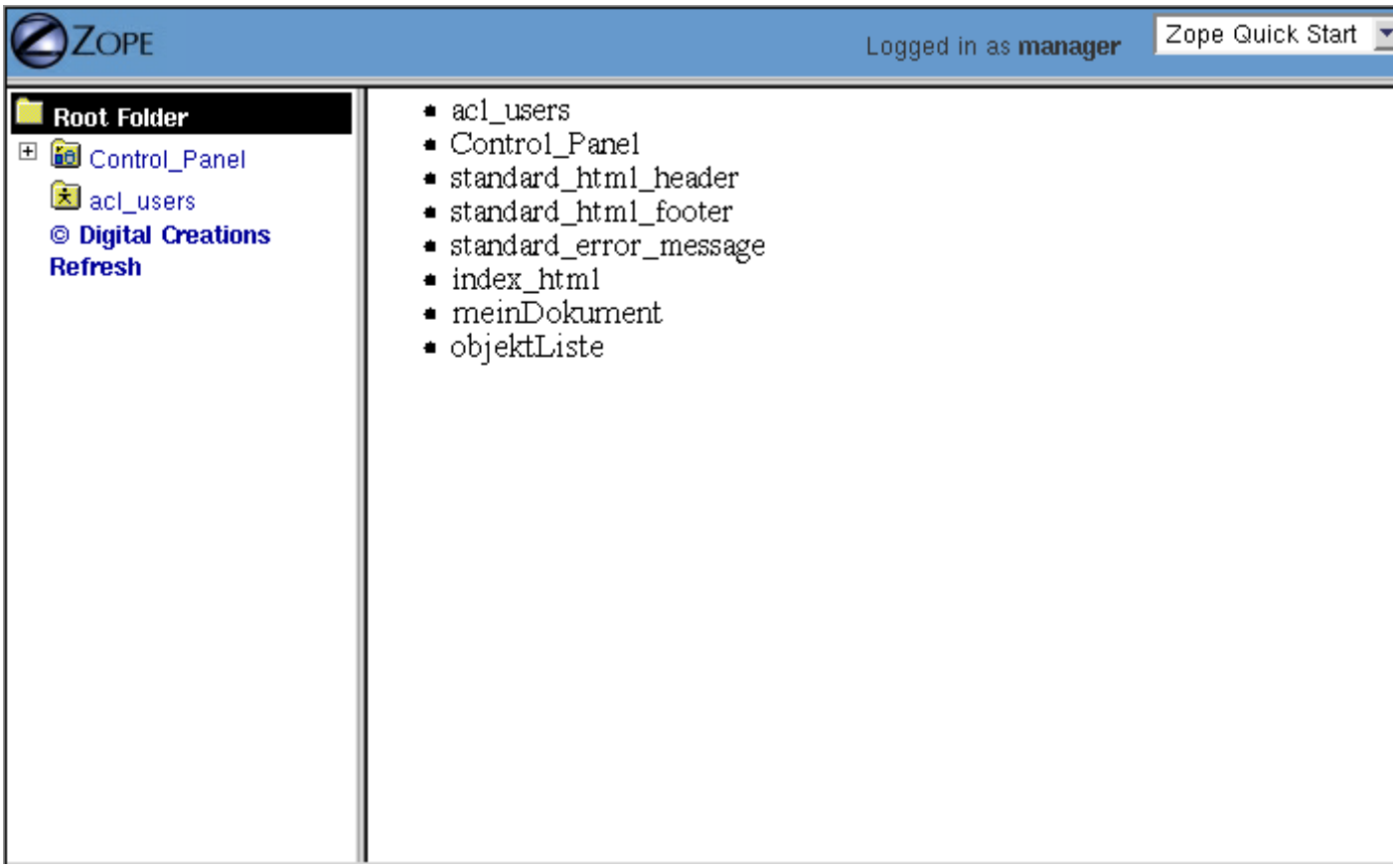


Abbildung 3-7 Ergebnisse der DTML-Methode objektListe

Alle Ordner implementieren die *objectValues*-Methode. Die *objectValues*-Methode ist Teil einer Schnittstelle namens *ObjectManager*, die alle Ordner implementieren.

Zusätzlich zur Anwendung von API-Methoden auf Objekte können DTML-Methoden auch auf eine bestimmten Art dazu verwendet werden, jedes Zope-Objekt zu erweitern. Dies wird im nächsten Kapitel detaillierter erklärt. Effektiv erlaubt Ihnen dies, das Zope-API zu erweitern, indem Sie einfach DTML-Methoden erzeugen.

Sie sahen gerade die *objektListe*-Methode, die sich im Stammordner befindet und eine einfache Liste des Stammordnerinhalts ausgibt. Weil die Methode im Stammordner ist, ist sie jetzt durch andere Objekte in oder unterhalb des Stammordners verwendbar. Diese Methode erweitert das Zope-API für diese Objekte, da es ihnen eine andere aufrufbare Methode liefert.

Um dies zu demonstrieren, lassen Sie uns einen *Primaten* genannten Unterordner erstellen und fügen Sie drei Dokumente, *Affen*, *Menschenaffen*, *Menschen* hinzu. Sie können die *objektListe*-Methode des *Primaten*-Ordners durch das Besuchen des URLs *Primaten/objektListe* aufrufen. Wie Sie sehen könne, unterscheidet sich die Auswirkung des Aufrufs der *objektListe*-Methode des *Primaten*-Ordners von der Auswirkung, es im Stammordner aufzurufen. Die *objektListe*-Methode ist im Stammordner definiert, aber wir verwenden sie hier, um den Inhalt des Primaten-Ordners anzuzeigen. Dieser Mechanismus, Objekte wieder zu verwenden, heißt *Akquisition* und wird in Kapitel 4 "Dynamische Inhalte mit DTML" näher erklärt.

DTML-Methoden dienen hauptsächlich als Darstellungsvorlagen. DTML-Methoden können als Vorlagen dienen, die wiederverwendbare Inhaltselemente zu dynamischen Web-Seiten zusammenfügen. Die Vorlagenfunktionalität von DTML-Methoden wird im nächsten Kapitel im Detail er& ouml;rtert.

Vergleich zwischen DTML-Dokumenten und -Methoden

DTML-Methoden haben dieselbe Benutzerschnittstelle wie DTML-Dokumente, was Anfänger ein bisschen verwirren kann. Alle Vorgehensweisen, die Sie im letzten Kapitel für das Hinzufügen, das Bearbeiten, das Betrachten und das Hochladen von DTML-Dokumenten lernten, sind für DTML-Methoden identisch.

Eine Quelle für häufige Missverständnisse für Zope-Anfänger ist die Frage, wann man ein DTML-Dokument und wann eine DTML-Methode verwenden soll. Oberflächlich betrachtet scheinen diese zwei Möglichkeiten identisch. Beide enthalten DTML und anderen Inhalt, beide führen DTML-Code aus, und beide haben eine ähnliche Benutzerschnittstelle und ein ähnliches API. Was ist also der Unterschied?

DTML-Dokumente sind dazu gedacht, Dokument-ähnliche Inhalt zu speichern. Zum Beispiel könnten die verschiedenen Kapitel eines Buchs in einem DTML-Dokument gespeichert werden. Eine allgemeine Regel ist: wenn Ihr Inhalt hauptsächlich Dokument-orientiert sind und Sie ihn auf Ihrer Site darstellen wollen, dann sollte er in ein DTML-Dokument kommen.

DTML-Methoden sind dazu gedacht, andere Objekte zu verändern und darzustellen. DTML-Methoden speichern normalerweise nicht viel Inhalt, es sei denn, der Inhalt soll sich ändern oder anderen Inhalt manipulieren.

Machen Sie sich keine Sorgen, wenn Sie Ihnen die Unterschiede zwischen DTML-Dokument und -Methode immer noch unklar sind. Sogar die erfahrensten Zope-Programmierer müssen etwas überlegen, bevor sie entscheiden, welche Art von Objekt sie verwenden. In Kapitel 8 "Variablen und DTML für Fortgeschrittene" werden Sie etwas über die technischen Unterschiede zwischen DTML-Dokumenten und DTML-Methoden lernen (Sie suchen Variablen auf unterschiedliche Weise, weil Sie unterschiedliche "Client"-Objekte besitzen). Hier sind einige allgemeine Regeln, um Ihnen zu helfen, sich zwischen DTML-Dokumenten und -Methoden zu entscheiden:

- Wenn es Inhalt ist, verwenden Sie ein DTML-Dokument oder eine Datei wenn Sie keinerlei DTML-Scripting benötigen.
- Wenn es einfache Logik ist, verwenden Sie eine DTML-Methode.
- Wenn es von anderen Objekten dargestellt werden soll, verwenden Sie ein DTML-Dokument.
- Wenn es andere Objekte darstellen soll, verwenden Sie eine DTML-Methode.
- Wenn es komplexes Verhalten ist, verwenden Sie ein Python- oder Perl-basiertes Skript.

Wie Sie gesehen haben, sind DTML-Methoden ein hilfreiches Werkzeug für Darstellung und einfache Programmierung, aber vielleicht werden Sie die Leistungsfähigkeit einer Programmiersprache mit vollem Funktionsumfang wollen und hier kommen schließlich die *Skripte* ins Spiel.

Sitzungen verwenden

Sitzungen (Sessions) erlauben es, die Übersicht über Site-Beuscher zu behalten. Web-Browser benutzen ein Protokoll namens HTTP um Daten mit einem Server wie Zope auszutauschen. HTTP bietet einem Server keine Möglichkeit die Übersicht über die Anfragen eines Benutzers zu behalten. Jede Anfrage wird völlig unabhängig von anderen betrachtet.

Sitzungen überwinden diese Einschränkung von HTTP. Der Begriff "Sitzung" bezeichnet eine Reihe von zusammengehörenden HTTP-Anfragen, die von demselben Client während eines bestimmten Zeitraums kommen. Zopes Sitzungssystem benützt Cookies und/oder HTTP-Formular-Elemente "im Hintergrund", um die Sitzungen eines Benutzer zu verfolgen. Mit Zopes Sitzungssystem können Sie das manuelle Verwalten von Benutzersitzungen vermeiden.

Sie können Sitzungen dazu verwenden, anonyme Benutzer zu verfolgen, wie auch diejenigen die Konten zur Zope-Anmeldung besitzen.

Daten die mit einer Sitzung verknüpt sind nennt man "Sitzungsdaten" (Session Data). Sitzungsdaten sind nur für die Dauer eines Site-Besuchs gültig, die über eine einstellbare Zeitschwelle festgelegt wird, in der ein Benutzer nicht aktiv ist. Sitzungsdaten werden dazu verwendet Informationen über den Besuch eines Benutzers zu speichern, wie die Artikel die ein Benutzer in einen "Einkaufskorb" legt oder welche Seiten ein Benutzer während seines Besuchs auf Ihrer Site aufgerufen hat.

Es ist wichtig, dass man sich bewusst darüber wird, dass das Speichern von sensiblen Informationen in Sitzungsdaten potenziell unsicher ist, wenn die Verbindung zwischen Browsern und Zope nicht in irgend einer Weise verschlüsselt ist. Speichern Sie keine sensiblen Daten wie Telefonnummern, Adressen, Kontonummern, Kreditkartennummern oder andere persönliche Informationen über Ihre Site-Besucher, solange Sie die Verbindung zwischen Zope und den Site-Besuchern nicht mit SSL abgesichert haben.

Sitzungs-Konfiguration

Zope-Versionen nach 2.5 bieten standardmäßig eine vorgefertigte Sitzungsumgebung, das bereits startfähig konfiguriert ist. Es besteht also kein Anlass diese Objekte zu ändern, solange Sie nicht neugierig sind oder die Sitzungseinstellungen verändern möchten. Benutzen Sie das Hilfesystem um Informationen darüber zu erhalten, wie Sie die Einstellungen ändern können.

Zope verwendet mehrere verschiedene Arten von Objeken um Sitzungsdaten zu verwalten. Hier folgen kurze Beschreibungen für ihre Verwendungszwecke.

Browser ID Manager (Browserkennungs-Verwaltung)
: Dieses Objekt verwaltet die Art und Weise, wie die Browser der Benutzer von Anfrage zu Anfrage identifiziert werden und erlaubt es Ihnen einzustellen, ob dies mit Hilfe von Cookies oder Formularvarialben oder einer Kombination aus beiden geschehen soll. Die standardmäßige Sitzungskonfiguration stellt eine Browserkennungs-Verwaltung als Objekt namens `/browser_id_manager` bereit.

Transient Object Container (Behälter für flüchtige Objekte)
: Dieses Objekt speichert Sitzungsdaten. Es erlaubt Ihnen festzulegen, wie lange Sitzungsdaten erhalten bleiben bevor sie ungültig werden. Die standardmäßige Sitzungskonfiguration stellt einen Behälter für flüchtige Objekte Namens

`/temp_folder/session_data` bereit. Die Sitzungsdaten-Objekte im Standardbehälter für flüchtige Objekte `session_data` gehen bei jedem Neustart von Zope verloren.

Session Data Manager (Sitzungsdaten-Verwaltung)
: Dieses Objekt verbindet die Browserkennungs- und Sitzungsdaten-Informationen. Wird ein Ordner der Sitzungsdaten enthält durchlaufen, wird das REQUEST-Objekt mit SESSION bestückt, einem Sitzungsdaten-Objekt. Die standardmäßige Sitzungskonfiguration stellt eine Sitzungsdaten-Verwaltung Namens `/session_data_manager`.

Sitzungsdaten verwenden

Sie werden auf Sitzungsdaten typischerweise über das Attribut `SESSION` im REQUEST-Objekt zugreifen.

Hier ist ein Beispiel, wie man durch Verwendung eines python-basierten Skripts mit einer Sitzung arbeitet:

```
        ## Script (Python) "letzterAufruf"
        sek_pro_tag=24*60*60
        sitzung=context.REQUEST.SESSION
        if sitzung.has_key('letzter Aufruf'):
            # Das Skript wurde bereits aufgerufen, da 'letzter Aufruf'
            # vorher schon einmal in der Sitzung gesetzt wurde.
            vorher=sitzung['letzter Aufruf']
            jetzt=context.ZopeTime()
            sitzung['letzter Aufruf']=jetzt # letzten Aufruf auf jetztige
Zeit setzen
            return 'Sekunden seit letztem Aufruf %.2f' % ((jetzt - vorher)
* sek_pro_tag)
        # Das Skript wurde vorher noch nie Aufgerufen, weil 'letzter Aufruf
        # in den Sitzungsdaten nicht existiert.
        sitzung['letzter Aufruf']=context.ZopeTime()
        return 'Dies ist Ihr erster Aufruf'
```

Rufen Sie dieses Skript ab und laden Sie es mehrmals neu. Es weiß darüber Bescheid, wann Sie das Skript zuletzt abgerufen haben und berechnet die Zeit zwischen den Aufrufen. Wenn Sie Ihren Browser beenden und das Skript wieder aufrufen, hat es Sie vergessen. Wenn Sie jedoch einfach ein paar andere Seiten besuchen und wieder zurückkehren, weiß es den Zeitpunkt Ihres letzten Aufrufs immer noch.

Dieses Beispiel zeigt die grundlegenden Funktionen für das Arbeiten mit Sitzungsdaten: Sitzungsdaten-Objekte verhalten sich genau wie Python-Dictionaries. Sie werden fast immer Sitzungsdaten verwenden, die aus normalen Python-Listen, Dictionaries, Zeichenketten und Zahlen bestehen. Das einzig knifflige an Sitzungen ist, dass wenn Sie mit veränderbaren Sitzungsdaten (wie zum Beispiel Dictionaries oder Listen) arbeiten, Sie die Sitzungsdaten durch erneutes Zuweisen speichern müssen. Hier ist ein Beispiel:

```
        ## Script (Python) "sitzungsBeispiel"
        sitzung=context.REQUEST.SESSION
        # l ist eine Liste
        l=sitzung['meineListe']
        l.append('Spam')
```

```
# Wenn Sie hier auhören, werden Ihre Änderungen an der Liste
# nicht gespeichert. Sie müssen die Sitzungsdaten speichern indem
# Sie sie der Sitzung erneut zuweisen.
sitzung['meineListe']=l
```

Weitere Informationen über Persistenz und veränderbare Daten finden Sie im *Zope Developer's Guide*.

Sie können Sitzung auch in Seitenvorlagen und DTML-Dokumenten verwenden. Hier beipielsweise ein Code-Stück aus einer Vorlage, das die Lieblingsfarbe des Benutzers darstellt (wie sie in der Sitzung gespeichert ist):

```
<p tal:content="request/SESSION/lieblingsfarbe">Blau</p>
```

Hier das gleiche, nur in DTML:

```
<dtml-with SESSION mapping>
  <p><dtml-var lieblingsfarbe></p>
</dtml-with>
```

Sitzungen haben eine Fülle von zusätzlichen Konfigurationsparametern und Anwendungsfällen. Für weitere Informationen über die Anwendungsprogrammierschnittstelle von Sitzungen, sehen Sie im Zope-Hilfesystem nach. Für ein weiteres Beispiel für die Verwendung von Sitzungen, sehen Sie sich das "Shopping Cart"-Beispiel an, das zusammen mit Zope 2.5 und höheren Version geliefert wird (im Examples-Ordner).

Versionen verwenden

Versionsobjekte helfen, die Arbeit mehrerer Leute an denselben Objekten zu koordinieren. Während Sie ein Dokument bearbeiten, kann jemand anderes zur selben Zeit ein anderes Dokument bearbeiten. In einer großen Zope-Site können Hunderte oder sogar Tausende von Menschen Zope gleichzeitig verwenden. Den Großteil der Zeit funktioniert dies gut, aber es können Probleme auftreten. Zum Beispiel könnten zwei Leute zu derselben Zeit dasselbe Dokument bearbeiten. Wenn die erste Person ihre Änderungen abschließt, werden sie in Zope gesichert. Wenn die zweite Person ihr Änderungen abschließt überschreibt sie die Änderungen der ersten Person. Sie können dieses Problem immer durch und Verwenden der *Undo*- und *History*-Funktionen umgehen, aber es kann immer noch ein Problem sein. Um dieses Problem zu beheben, hat Zope *Version*-Objekte.

Ein anderes Problem auf das Sie stoßen können, ist, dass Sie zwar einige Änderungen vornehmen möchten, aber Sie möchten sie nicht öffentlich machen bis Sie fertig sind. Nehmen wir beispielsweise an, dass Sie die Menüstruktur Ihrer Site ändern wollen. Sie wollen nicht an diesen Änderungen arbeiten, während Leute Ihre Site verwenden, weil es das Navigationssystem zeitweilig zerstören kann, während Sie daran arbeiten.

Versionen sind eine Möglichkeit, nicht-öffentliche Änderungen in Zope vorzunehmen. Sie können Änderungen an vielen verschiedenen Dokumenten vornehmen, ohne dass andere

Leute sie sehen. Wenn Sie entscheiden, dass Sie fertig sind, können Sie Ihre Änderungen öffentlich machen oder sie verwerfen. Sie können in einer Version arbeiten solange Sie wollen. Zum Beispiel kann es Sie eine Woche kosten, Ihrem neues Menüsystem den letzten Schliff zu geben. Sobald Sie fertig sind, können Sie all Ihre Änderungen auf einmal zu veröffentlichen, indem Sie die Version freig eben.

Erstellen Sie eine Version durch das Wählen von "Version" in der Produktauswahlliste. Sie sollten zu einem Erstellformular kommen. Geben Sie Ihrer Version die Id *MeineAenderungen* und klicken Sie auf die Schaltfläche *Add*. Jetzt haben Sie eine Version erstellt, verwenden sie jedoch noch nicht. Um Ihren Version zu verwenden klicken Sie auf sie. Sie sollten zur Ansicht *Join/Leave* Ihrer Version kommen, wie in Abbildung 3-8 gezeigt.

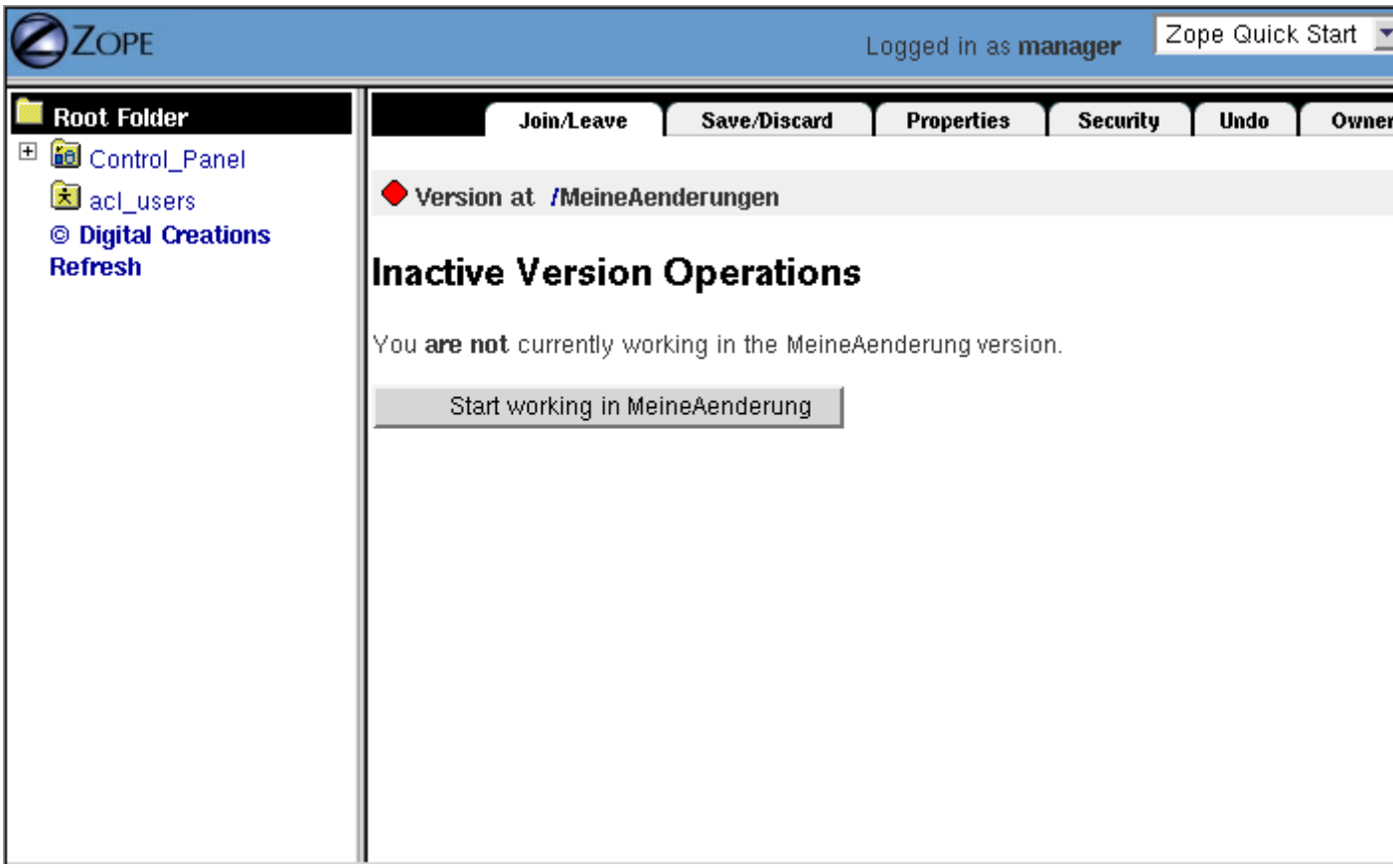


Abbildung 3-8 Anmelden an einer Version

Die Version teilt Ihnen mit, dass Sie sie gegenwärtig nicht verwenden. Klicken Sie auf die Schaltfläche *Start Working in MeineAenderungen*. Jetzt sollte Zope Ihnen sagen, dass Sie in einer Version arbeiten. Kehren Sie jetzt zum Stammordner zurück. Achten Sie darauf, dass Sie überall wohin Sie gehen eine kleine Nachricht am oberen Fensterrand erscheint, die sagt *You are currently working in version /MeineAenderungen*. Diese Nachricht lässt Sie wissen, dass Änderungen, die Sie an dieser Stelle vornehmen, nicht öffentlich sein werden, aber in Ihrer Version gespeichert werden. Erstellen Sie zum Beispiel ein DTML-Dokument namens *neu*. Wie Sie merken, hat es eine kleine rote Raute nach seiner Kennung. Bearbeiten Sie jetzt Ihre *standard_html_header*-Methode. Fügen Sie ihr eine Zeile hinzu wie etwa:

```
<HTML>
  <HEAD>
    <TITLE><dtml-var title_or_id></TITLE>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
  <H1>In einer Version geändert</H1>
```

Jedes Objekt, das Sie erstellen oder bearbeiten, während Sie in einer Version arbeiten, wird mit einer roten Raute markiert werden Kehren Sie jetzt zu Ihrer Version zurück und klicken Sie auf die Schaltfläche *Quit working in MeineAenderungen*. Versuchen Sie jetzt zum Dokument *neu* zurückzukehren. Achten Sie darauf, dass das Dokument, das Sie während Sie in Ihrer Version waren erstellt haben, jetzt verschwunden ist. Andere Änderungen, die Sie in der Version vorgenommen haben, sind auch verschwunden. Wie Sie merken, hat Ihre *standard_html_header*-Methode jetzt eine kleine rote Raute und ein nachfolgendes Sperrsymbol. Dies zeigt, dass dieses Objekt in einer Version geändert worden ist. Wird Objekt in einer Version zu ändern, wird es gesperrt es, sodass es niemand anderes mehr ändern kann, bis Sie die Änderungen die Sie in Ihrer Version vorgenommen haben bestätigen oder verwerfen. Die Sperrung stellt sicher, dass Ihre Versionsänderungen keine Änderungen überschreiben, die andere Leute vornehmen, während Sie in einer Version arbeiten. Wenn Sie sich zum Beispiel vergewissern wollen, dass nur Sie zu einer bestimmten Zeit an einem Objekt arbeiten, so können Sie es in einer Version ändern. Die Sperrung schützt Sie sich nicht nur vor unerwarteten Änderungen, sie macht auch die Sache auch unbequem, wenn Sie etwas bearbeiten wollen, das gerade von jemand anderem gesperrt ist. Es ist sinnvoll, die Verwendung von Versionen einzuschränken, um das Auszusperren anderer Leute beim Vornehmen von Änderungen an Objekten zu vermeiden.

Kehren Sie jetzt zu Ihrer Version dadurch zurück, indem Sie den auf sie und dann auf die Schaltfläche *Start working in MeineAenderungen* klicken. Achten Sie darauf, dass alles in den Zustand zurückkehrt, wie es war, bevor Sie die Version verließen. Lassen Sie uns an dieser Stelle Ihre Änderungen dauerhaft machen. Gehen Sie zur Ansi cht *Save/Discard*, wie in Abbildung 3-9 gezeigt.

Abbildung 3-9 Versionsänderungen freigeben.

Geben Sie einen Kommentar wie *Dies ist ein Test* in das Eingabefeld Comment ein und klicken Sie auf die Schaltfläche *Save*. Ihre Änderungen sind jetzt öffentlich und alle Objekte, die Sie in Ihrer Version geändert haben, sind jetzt nicht mehr gesperrt. Achten Sie darauf, dass Sie immer noch in Ihrer Version arbeiten. Gehen Sie zur Ansicht *Join/Leave* und klicken Sie auf die Schaltfläche *Quit Working in MeineAenderungen*. Stellen Sie jetzt sicher, dass das Dokument, das Sie in Ihrer Version erstellt haben, sichtbar ist. Ihre Änderung am *standard_html_header* sollte auch sichtbar sein. Wie überall in Zope, können Sie diese Änderungen rückgängig machen, wenn Sie wollen. Gehen Sie zur Ansicht *Undo*. Wie Sie bemerken, haben Sie statt vieler Transaktionen - einer für jede Änderung - nur eine Transaktion für alle, die Änderungen haben, die Sie in Ihrer Version vorgenommen haben. Wenn Sie die Transaktion widerrufen, werden alle in der Version vorgenommenen Änderungen rückgängig gemacht.

Versionen sind ein leistungsfähiges Werkzeug für die Zusammenarbeit in Teams. Sie müssen keine Live- und Test-Server laufen lassen, da Versionen es Ihnen erlauben Experimente durchzuführen, sie zu beurteilen und sie dann zu öffentlich machen, wenn Sie entscheiden, dass alles in Ordnung ist. Sie sind nicht darauf beschränkt nur in einer Version zu arbeiten. Viele Leute können in derselben Version arbeiten. Auf diese Weise können Sie über Version zusammenarbeiten, während sie die Änderungen vor der allgemeinen Öffentlichkeit verborgen halten.

Versionen und ZCatalog

Versionen arbeiten nicht gut mit ZCatalog zusammen. Der Grund dafür ist, dass Versionen Objekte sperren sobald diese in einer Version bearbeitet werden, um Änderungen außerhalb einer Version zu verhindern. Dies funktioniert gut wenn Änderungen isoliert sind.

ZCatalog verknüpft auf bestimmte Weise die Veränderungen von ungleichen Objekten. Dies passiert, weil das katalogisieren eines Objekts notwendigerweise den Catalog ändern muss. Objekte die sich selbst katalogisieren wenn sie verändert werden geben ihre Änderungen an den Catalog weiter. Wenn solch ein Objekt in einer Version geändert wird, wird auch der Catalog in der Version geändert, wodurch der Catalog gesperrt wird. Durch diese Eigenschaft lassen sich Catalog und Versionen schlecht miteinander verwenden. In der Regel sollten Versionen nicht in Anwendungen eingesetzt werden die einen Catalog verwenden.

Pufferung zur Leistungssteigerung

Ein *Cache* ist ein temporärer Ort, um Informationen zu speichern, auf die häufig zugriffen wird. Der Grund für das Verwenden eines Caches ist die Geschwindigkeit. Jede Art von dynamischem Inhalt wie eine DTML-Seite oder ein Python-Skript muss jedes Mal ausgewertet werden, wenn es aufgerufen wird. Für einfache Seiten oder schnelle Skripte ist dies normalerweise kein Problem. Bei sehr komplexen DTML-Seiten oder Skripten, die viele Berechnungen ausführen oder entfernte Server aufrufen kann der Zugriff auf diese Seite oder dieses Skript mehr als nur ein triviales Maß an Zeit beanspruchen. Sowohl DTML als auch Python können so komplex werden, besonders wenn Sie viele Schleifen verwenden (wie das `in`-Tag oder die `for`-Schleife in Python), oder wenn Sie viele Skripte aufrufen, die wiederum viele Skripte aufrufen usw. Verarbeitungen, die viel Zeit benötigen, werden *rechenintensiv* genannt.

Durch Speichern des Ergebnisses nach einem einzelnen Aufruf einer rechenintensiven Seite oder eines rechenintensiven Skripts kann ein Cache durch eine spätere Wiederverwendung dieses Aufrufs die Geschwindigkeit Ihrer Site stark erhöhen. Die aller erste Person die diese Seite aufruft erhält die übliche lange Antwortzeit, aber sobald der Wert der Berechnung im Cache gespeichert ist, werden alle folgenden Benutzer die die Seite betrachten werden eine sehr kurze Antwortzeit bemerken weil sie die *gepufferte Kopie* des Ergebnisses bekommen und nicht wieder die selbe rechenintensive Verarbeitung durchlaufen, die der erste Benutzer durchlief.

Um Ihnen eine Vorstellung davon zu geben, wie Caches Ihre Site-Geschwindigkeit verbessern können, stellen Sie sich vor Sie erstellen *www.zopezoo.org* und die aller erste Seite Ihrer Site ist sehr komplex. Lassen Sie uns annehmen, dass diese Seite komplexe Kopf- und Fußzeilen hat, mehrere verschiedene Datenbanktabellen abfrägt und mehrere spezielle Skripte auf komplexe Weise aufruft, die die Ergebnisse der Datenbankabfragen auf komplexe Weise verarbeiten. Jedes Mal, wenn ein Benutzer zu *www.zopezoo.org* kommt, muss Zope diese sehr aufwendige Seite wiedergeben. Lassen Sie uns für Demonstrationszwecke annehmen, dass diese Berechnung dieser komplexen Seite eine halbe Sekunde oder 500 Millisekunden in Anspruch nimmt.

Angenommen es dauert eine halbe Sekunde, um diese fiktionale komplexe Hauptseite wiederzugeben, dann kann Ihr Rechner nur 120 Anfragen pro Minute bearbeiten. In Wirklichkeit wäre diese Zahl wahrscheinlich noch niedriger, weil Zope zusätzlich noch andere Dinge zu tun hat als diese Hauptseite bereitzustellen. Stellen Sie sich jetzt vor, dass Sie diese so einrichten, dass sie gepuffert wird. Da nichts von der rechenintensiven Verarbeitung geleistet werden muss, um die zwischengespeicherte Kopie der Seite anzuzeigen, könnten

noch viel mehr Benutzer die Hauptseite betrachten. Wenn es beispielsweise 10 Millisekunden dauert, um eine zwischengespeicherte Seite anzuzeigen, dann wird diese Seite Ihren Webseite-Besuchern *50 mal schneller* zur Verfügung gestellt. Die tatsächliche Leistung des Caches und von Zope hängt stark von Ihrem Computer und Ihrer Anwendung ab, aber dieses Beispiel gibt Ihnen eine Vorstellung davon, wie Pufferung Ihre Web-Site ziemlich stark beschleunigen kann. Es gibt jedoch einige Nachteile beim Puffern:

Cache-Lebensdauer
: Wenn Seiten lange zwischengespeichert werden, könnten sie die aktuellste Information über Ihre Site nicht wiedergeben. Wenn Sie Informationen haben, die sich sehr schnell ändern, kann das Puffern Ihren Benutzern die neue Information vorenthalten, weil die zwischengespeicherte Kopie die alten Informationen enthält. Wie lange ein Ergebnis zwischengespeichert bleibt wird *Cache-Lebensdauer* der Information genannt.

Persönliche Informationen
: Viele Web-Seiten können für einen speziellen Benutzer personalisiert werden. Diese Informationen zwischenzuspeichern und sie einem anderen Benutzer zu zeigen, wäre aufgrund von Vertraulichkeitsaspekten offensichtlich schlecht und auch weil der andere Benutzer keine Information über *sich* sondern über jemand anderen bekommen würde. Deshalb wird personalisierte Information fast nie gepuffert.

Zope erlaubt Ihnen, diese Probleme durch das Einrichten einer *Cache-Richtlinie* zu umgehen. Die Cache-Richtlinie erlaubt Ihnen, zu kontrollieren, wie Inhalte zwischengespeichert werden. Cache-Richtlinien werden von *Cache-Manager*-Objekten verwaltet.

Einen Cache-Manager hinzufügen

Cache-Manager können genau wie jedes andere Zope-Objekt hinzugefügt werden. Gegenwärtig bietet Zope zwei Arten von Cache-Managern:

HTTP Accelerated Cache Manager
: Ein HTTP Accelerated Cache Manager erlaubt Ihnen, einen HTTP-Cache-Server zu steuern, der extern zu Zope läuft, zum Beispiel Squid. HTTP Accelerated Cache Manager machen das Puffern nicht selbst, aber sie setzen spezielle HTTP-Header, die einem externen Cache-Server sagen, was zwischenzuspeichern ist. Einen externen Cache-Server wie Squid einzurichten geht über den Rahmen dieses Buchs hinaus. Für nähere Details sehen Sie auf der Squid-Site nach.

(RAM) Cache Manager
: Ein RAM Cache Manager ist ein Zope-Cache-Manager, der den Inhalt von Objekten in Ihrem Computerspeicher puffert. Dies macht ihn sehr schnell, bewirkt aber auch, dass Zope mehr Speicher Ihres Computers verbraucht. Ein RAM Cache Manager erfordert um zu funktionieren keine externen Ressourcen wie ein Squid-Server.

Erstellen Sie für die Zwecke dieses Beispiels einen RAM Cache Manager Namens *CacheManager* im Stammordner. Dies wird das Cache-Manager-Objekt für Ihre ganze Site sein.

Jetzt können Sie auf *CacheManager* klicken und seinen Konfigurationsbildschirm betrachten. Es gibt mehrere Elemente auf diesem Bildschirm:

Title
: Der Titel des Cache-Managers. Dieser ist optional.

REQUEST-Variablen
: Diese Information wird verwendet, um die zwischengespeicherte Kopie einer Seite zu speichern. Dies ist ein fortgeschrittenes Merkmal, zunächst können Sie es nur auf "AUTHENTICATED_USER" gestellt lassen.

Threshold Entries (Schwellwert für Einträge)
: Die Anzahl von Objekten, die der Cache-Manager gleichzeitig puffern wird.

Cleanup Interval (Aufräumintervall)
: Die Lebensdauer von gepufferten Ergebnissen.

Lassen Sie zunächst alle diese Einträge wie sie sind, es sind gute, vernünftige Standardwerte. Das ist bereits alles, was man tun muss, um einen Cache-Manager einzurichten!

Es gibt ein Paar weitere Ansichten in einem Cache-Manager, die Sie vielleicht nützlich finden werden. Das erste ist die Ansicht *Statistics*. Diese Ansicht zeigt Ihnen die Anzahl von Treffern ("Hits") und Fehlschlägen ("Misses") im Cache an, um Ihnen zu zeigen, wie wirkungsvoll Ihre Pufferung ist.

Es gibt auch eine Ansicht namens *Associate*, die Ihnen erlaubt, einen bestimmten Typ oder Typen von Zope-Objekten mit einem speziellen Cache-Manager zu verknüpfen. Zum Beispiel kann es sein, dass Sie möchten, dass Ihr Cache-Manager nur DTML-Dokumente puffert. Sie können diese Einstellungen in der Ansicht *Associate* ändern.

An dieser Stelle wird noch nichts zwischengespeichert, Sie haben nur einen Cache-Manager erstellt. Der nächste Abschnitt erklärt, wie Sie den Inhalt tatsächlicher Dokumente puffern können.

Ein Dokument puffern

Ein Dokument zu puffern ist sehr leicht. Bevor Sie ein Dokument puffern können müssen Sie zuerst einen Cache-Manager haben, wie den, den Sie im vorherigen Abschnitt erstellt haben.

Um ein Dokument zu puffern, erstellen Sie ein neues DTML-Dokument-Objekt im Stammordner namens *Wetter*. Dieses Objekt enthält eine Wetterinformation. Lassen Sie uns zum Beispiel sagen, es enthält:

```
<dtml-var standard_html_header>

  <p>Gestern hat es geregnet.</p>

<dtml-var standard_html_footer>
```

Klicken Sie jetzt auf das *Wetter*-DTML-Dokument und auf seine *Cache*-Ansicht. Diese Ansicht lässt Sie dieses Dokument mit einem Cache-Manager verknüpfen. Wenn Sie sich im Listenfeld nach unten bewegen, sehen Sie den Cache-Manager *CacheManager*, den Sie im vorherigen Abschnitt erstellten. Wählen Sie diesen als Cache-Manager für *Wetter* aus.

Jedes Mal wenn jetzt irgend jemand das *Wetter*-Dokument besucht, bekommt er stattdessen die gepufferte Kopie. Für ein so triviales Dokument wie unser *Wetter*-Beispiel hat dies nicht viel Nutzen. Aber stellen Sie sich einmal vor, *Wetter* würde einige Datenbankabfragen enthalten. Zum Beispiel:

```
        <dtml-var standard_html_header>

          <p>Das gestrige Wetter war <dtml-var gesternAbrage></p>

          <p>Die aktuelle Temperatur beträgt <dtml-var
aktuelleTempAbfrage></p>

        <dtml-var standard_html_footer>
```

Lassen Sie uns annehmen, dass *gesternAbrage* und *aktuelleTempAbfrage* SQL-Methoden sind, die eine Datenbank für die gestrige Vorhersage beziehungsweise die aktuelle Temperatur abfragen (für weitere Informationen über SQL-Methoden sehen Sie in Kapitel 12 "Verbindung an relationale Datenbanken" nach). Lassen Sie uns auch annehmen, dass sich die Information in der Datenbank nur einmal jede Stunde ändert.

Jetzt würde das *Wetter*-Dokument ohne zwischenzuspeichern die Datenbank jedes Mal abfragen, wenn es abgerufen würde. Wenn das *Wetter*-Dokument Hunderte Male in einer Stunde abgerufen werden würde, dann würden all jene Hunderte von Abfragen immer dieselbe Information enthalten.

Wenn Sie festlegen, dass das Dokument jedoch gepuffert werden soll, dann führt das Dokument die Abfrage nur durch, wenn der Cache erlischt. Die Standardcachezeit ist 300 Sekunden (5 Minuten), was Ihnen 91 % Ihrer Datenbankabfragen erspart, wenn Sie das Dokument puffern lassen, weil sie nur noch ein Zwölftel so oft abfragen. Es gibt einen Nachteil dieser Methode: Es existiert eine Chance, dass es sein kann, dass die Daten fünf Minuten veraltet sind, aber dies ist normalerweise ein akzeptabler Kompromiss.

Für weitere Informationen über das Puffern und das Verwenden der fortgeschritteneren Cache-Optionen sehen sie im Zope Administrator's Guide nach.

Virtual-Hosting-Objekte

Zope bietet drei Objekte, die Ihnen helfen, virtuelle Hosts einzurichten: *SiteRoot*, *Set Access Rule* und *Virtual Host Monster*. Virtuelles Hosting ist ein Weg viele Web-Sites mit einem Zope Server zu betreuen. Virtuelles Hosting ist eine fortgeschrittene Verwaltungsfunktion, die über den Rahmen dieses Buchs hinausgeht. Sehen im Zope Administrator's Guide für weitere Informationen über das virtuelle Hosting nach.

Mit MailHost Mails verschicken

Zope kommt zusammen mit einem Objekt, das dazu verwendet wird E-Mails abzuschicken, normalerweise in Verbindung mit dem DTML-Tag `sendmail`, das in Kapitel 8 "Variablen und DTML für Fortgeschrittene" weiter beschrieben wird.

MailHosts können sowohl von Python als auch von DTML aus dazu benutzt werden, eine E-Mail-Nachricht über das Internet zu verschicken. Sie sind nützlich als *Gateways* zur Welt. Jedes MailHost-Objekt ist mit einem Mail-Server verknüpft. Beipsielsweise kann man MailHost-Objekt mit `ihrmail.ihredomain.com` verknüpfen, der ihr SMTP-Mail-Server für ausgehende Nachricht wäre. Sobald Sie einen Server mit einem MailHost-Objekt verknüpfen, verwendet das MailHost-Objekt immer diesen Server um Mails zu senden.

Um ein MailHost-Objekt zu erstellen, wählen Sie MailHost aus der Produktauswahlliste. Wie Sie sehen können ist die Standard-Id "MailHost" und der Standard-SMTP-Server und der Port sind "localhost" und "25". Stellen Sie sicher, dass entweder auf Ihrem eigenen Rechner (localhost) ein Mail-Server läuft oder ändern Sie "localhost" in den Namen Ihres SMTP-Servers für ausgehende Nachrichten.

Jetzt können Sie das neue MailHost-Objekt von einem DTML-`sendmail`-Tag aus verwenden. Dies wird in Kapitel 8 "Variablen und DTML für Fortgeschrittene" ausführlich beschrieben. Die API für MailHost-Objekte erlaubt Ihnen Mails von Python-Skripten aus zu verschicken. Sehen Sie im Online-Hilfesystem für weitere Informationen nach.

Zopebuch: Inhaltsverzeichnis

Kapitel 4: Dynamische Inhalte mit DTML

DTML (Document Template Markup Language) ist Zopes Tag-basierte Präsentations- und Scripting-Sprache. DTML dynamisiert das Generieren, Kontrollieren und Formatieren von Inhalten. DTML wird allgemein benutzt, um modulare und dynamische Web-Schnittstellen für ihre Applikationen herzustellen.

DTML ist eine serverbasierte Scripting-Sprache wie SSI, PHP, ASP und JSP. Das bedeutet, daß DTML-Kommandos durch Zope auf dem Server ausgeführt werden und das Ergebnis der Ausführung zu ihrem Web-Browser gesendet wird. Demgegenüber werden clientseitige Scripting-Sprachen wie JavaScript nicht vom Server ausgeführt, sondern zum Browser gesendet und auch dort ausgeführt.

Sie können DTML-Scripting mit zweien der Zope-Objekte verwenden, nämlich *DTML-Dokumente* und *DTML-Skripte*.

Für wen ist DTML?

DTML wurde für Leute geschaffen, die mit HTML und grundlegendem Web-Skripting vertraut sind, nicht für Anwendungsprogrammierer. Wenn Sie mit Zope programmieren wollen, sollten Sie DTML tatsächlich nicht benutzen. In Kapitel 9 (Fortgeschrittenes Zope-Skripting) beschäftigen wir uns mit fortgeschrittenem Programmieren unter Python und Perl.

DTML ist für die Präsentation und sollte von Web-Designern verwaltet werden. Zope hilft Ihnen dabei, Präsentation und Logik getrennt zu halten, indem es verschiedene Objekte für Präsentation (DTML) und Logik (Python, Perl und andere) bereithält. Sie werden eine Menge Vorteile darin finden, ihre Präsentation in DTML zu halten und ihre Logik in anderen Typen von Zope-Objekten. Einige dieser Vorteile beinhalten:

- Logik und Präsentation getrennt zu halten, macht es leicht, jede Komponente zu verändern, ohne die andere zu stören.
- Sie werden oft verschiedene Leute damit beschäftigen, Logik und Präsentation zu warten. Mit der Benutzung verschiedener Objekte für diese Aufgaben machen Sie es leichter für Leute, zusammenzuarbeiten, ohne sich gegenseitig zu stören.
- Es ist einfacher, bestehende Präsentationen und logische Komnponenten zu nutzen, wenn sie nicht vermischt sind.

Wozu taugt DTML?

DTML taugt zum Anlegen dynamischer Web-Schnittstellen. Es unterstützt die Wiederbenutzung von Inhalten und Layout, die Formatierung heterogener Daten und trennt Präsentation von Logik und Daten.

Sie können mit DTML beispielsweise vorgegebene Header und Footer mehrfach benutzen:

```
<dtml-var standard_html_header>

<p>Hello world.</p>

<dtml-var standard_html_footer>
```

Diese Web-Seite mischt HTML und DTML miteinwander. DTML-Kommandos werden als Tags geschrieben, die mit *dtml-* beginnen. Dieses Beispiel baut eine Web-Seite auf, indem Standard-Header und -Footer in eine HTML-Seite eingefügt werden. Die entstandene HTML-Seite könnte etwa so aussehen:

```
<html>
<body bgcolor="#FFFFFF">

<p>Hello world.</p>

<hr>
<p>Last modified 2000/10/16 by AmosL</p>
</body>
</html>
```

Wie Sie sehen können, definiert der Standard-Header eine weiße Hintergrundfarbe und der Standard-Footer hat am Fuß eine Zeile eingefügt, die mitteilt, wann und von wem die Seite das letzte Mal geändert wurde.

Zusätzlich zur Wiederverwendung von Inhalten läßt DTML Sie einfach und mächtig alle Arten von Daten formatieren. Sie können DTML benutzen, um Skripte und Suchanfragen aufzurufen, Zope-Objekte zu untersuchen, Formulare zu verarbeiten und mehr.

Wenn Sie zum Beispiel eine Datenmbank mit einem SQL-Skript abfragen, gibt sie typischerweise eine Liste von Resultaten zurück. Hier sehen Sie, wie Sie DTML benutzen können, um jedes Ergebnis einer Datenbank-Abfrage zu formatieren:

```
<ul>
  <dtml-in frogQuery>
  <li><dtml-var animal_name></li>
  </dtml-in>
</ul>
```

Das Tag DTML *in* iteriert durch das Ergebnis der Datenbank-Abfrage und formatiert jedes Ergebnis. Angenommen, es werden vier Ergebnisse von der *frogQuery* zurückgegeben. Hier sehen Sie, wie das abschließende HTML aussehen könnte:

```
<ul>
  <li>Feuerbauchkröte</li>
  <li>Afrikanischer Klauenfrosch</li>
  <li>Nabu-Schilffrosch</li>
  <li>Chilenischer Vieraugenfrosch</li>
</ul>
```

Die Ergebnisse der Datenbankabfrage werden als Punktliste in HTML formatiert.

Beachten Sie: Sie müssen DTML nicht sagen, daß Sie eine Datenbank abfragen und auch nicht, wo es die Argumente findet, um die Datenbankabfrage aufzurufen. Sie sagen ihm nur, welches Objekt aufgrufen werden soll und es wird selbst herausfinden, auf welche Weise das Objekt aufgerufen werden und welche passenden Argumente übergeben werden müssen. Wenn Sie die das *frogQuery*-SQL-Skript mit einer anderen Art von Objekt wie etwa ein andere Skript, einen ZCatalog oder sogar ein anderes DTML-Skript austauschen, werden Sie die Formatierungsvorgaben nicht ändern müssen.

Diese Fähigkeit, alle Arten von Daten zu formatieren, macht DTML zu einem mächtigen Präsentationswerkzeug und läßt Sie ihre Geschäftslogik ändern, ohne ihre Präsentation ändern zu müssen.

Wann DTML nicht benutzt werden soll

DTML ist keine Allzweck-Programmiersprache. Zum Beispiel erlaubt es Ihnen nicht, sehr einfach Variablen anzulegen. Auch wenn es möglich sein mag, komplexe Algorythmen in DTML zu implementieren, ist es doch sehr aufwendig und nicht empfehlenswert. Wenn Sie logische Programmierung implementieren wollen, benutzen Sie Python oder Perl (für mehr Informationen zu diesen Themen siehe Kapitel 9 (Fortgeschrittene Zope-Programmierung)).

Lassen Sie uns zum Beispiel annehmen, daß Sie eine einfache Web-Seite für Mathematik-Schüler geschrieben haben, und Sie auf dieser Seite eine einfache Berechnung illustrieren wollen. Sie würden das Programm, das diese Berechnung anstellt, nicht in DTML schreiben wollen. Es könnte in DTML *gemacht* werden, wäre aber schwer zu verstehen. DTML wäre perfekt zum Beschreiben der Seite, in die ebendiese Berechnung eingefügt wird, aber es wäre

schrecklich, die Berechnung in DTML auszuführen, dagegen wäre es schlicht und einfach in Python oder Perl.

Die Verarbeitung von Strings ist ein anderes Gebiet, wo DTML nicht die erste Wahl ist. Wenn Sie den Input eines Benutzers auf komplexe Weise manipulieren wollen, aber Funktionen nutzen, die Strings m,anipulieren, tun Sie das besser in Python oder Perl, die beide sehr viel bessere Sting-Verarbeitungsfähigkeiten haben als DTML.

DTML ist ein Werkzeug von vielen, die in Zope verfügbar sind. Wenn Sie feststellen, daß Sie sich bei der Ausarbeitung eines komplexen DTML-Konstruktes am Kopf kratzen müssen, ist es wahrscheinlich, daß die Dinge besser laufen, wenn Sie ihr DTML-Skript auftrennen und inm eine Sammluung von DTML- und Python- oder Perl-Skripts umwandeln.

DTML-Tag-Syntax

Die Syntax von DTML ist der von HTML ähnlich. DTML ist eine Tag-basierte Markup-Sprache. In anderen Worten: DTML benutzt Tags, um seine Arbeit zu machen. Here ist ein kleiner Schnipsel DTML:

```
<dtml-var standard_html_header>

<h1>Hello World!</h1>

<dtml-var standard_html_footer>
```

Dieser DTML-Code enthält zwei DTML-*var*-Tags und etwas HTML. Die *h1*-Tags sind HTML, nicht DTML. Typischerweise mischen Sie DTML mit anderen Markup-Sprachen wie HTML. DTML wird normalerweise benutzt, um HTML zu generieren, aber nichts hindert Sie, andere Text-Typen zu generieren. Wie Sie später sehen werden, können Sie DTML genausogut eMail-Nachrichten oder andere textliche Informationen generieren lassen.

DTML enthält zwei Arten von Tags, nämlich *singleton*- und *block*-Tags. Singleton-Tags bestehen aus einem Tag, das von größer-als- (<) und kleiner-als-Symbolen (>) umschlossen werden. Das *var*-Tag ist ein Beispiel für ein Singleton-Tag:

```
<dtml-var parrot>
```

Es ist nicht nötig, das *var*-Tag zu schließen.

Block-Tags bestehen aus zwei Tags (eins öffnet den Block und eins schließt den Block) und Inhalt zwischen ihnen:

```
<dtml-in mySequence>

  <!-- dies ist ein HTML-Kommentar innerhalb eines  comment inside
the in tag block -->

</dtml-in>
```

Das Öffnungs-Tag beginnt den Block und das Schluß-Tag beendet ihn. Das Schluß-Tag hat denselben Namen wie das Öffnungs-Tag mit einem "Slash" am Anfang. Das ist dieselbe Konvention, die HTML und XML benutzen.

DTML-Tag-Attribute benutzen

Alle DTML-Tags haben Attribute. Ein Attribut stellt Informationen darüber zur Verfügung, wie das Tag arbeiten soll. Manche Attribute sind optional. Das *var*-Tag setzt beispielsweise den Wert einer Variablen ein. Es hat ein optional *fehlendes* Attribut, das einen Vorgabewert für den Fall definiert, daß die Variable nicht gefunden werden kann:

```
<dtml-var Spannweite missing="Spannweite unbekannt">
```

Wenn die Variable *Spannweite* nicht gefunden werden kann, wird `Spannweite unbekannt` eingestzt.

Manche Attribute haben keine Werte. Zum Beispiel können Sie eine eingesetzte Variable mit dem *upper*-Attribut in Versalien konvertieren:

```
<dtml-var exclamation upper>
```

Beachten Sie, daß das *upper*-Attribut - anders als das *missing*-Attribut - keinen Wert braucht.

Verschiedene Tags haben verschiedene Attribute. Siehe auch Anhang A (DTML-Referenz) für mehr Informationen über die Syntax der verschiedenen DTML-Tags.

Variablen mit DTML einsetzen

Das Einsetzen von Variablen ist die grundsätzlichste Aufgabe, die Sie mit DTML durchführen können. Sie haben schon gesehen, wie DTML mit dem *var*-Tag einen Header und einen Footer in eine Web-Seite einsetzt. Viele DTML-Tags setzen Variablen einund sie tun das alle auf ähnliche Weise. Lassen Sie uns genauer betrachten, wie Zope Variablen einsetzt.

Angenommen, Sie haben einen Ordner, dessen "id" *Futtertaschen* ist und der den "title" "Roberts Lustige Futtertaschen" hat. Legen Sie in dem Ordner ein DTML-Skript mit der "id" *Preisliste* an. Dann ändern Sie den Inhalt des DTML-Skripts auf Folgendes:

```
<dtml-var standard_html_header>

<h1>Preisliste für <dtml-var title></h1>

<p>
Hanftasche   EUR 2.50
</p>
<p>
Seidentasche EUR 5.00
```

```
</p>

<dtml-var standard_html_footer>
```

Sehen Sie sich nun das DTML-Skript an, indem Sie auf die Registerkarte *View* klicken. Sie sollten eine HTML-Seite sehen, deren Quelltext etwa wie dies aussieht:

```
<html>
<body>

<h1>Preisliste für Roberts Lustige Futtertaschen</h1>

<p>
Hanftasche   EUR 2.50
</p>
<p>
Seidentasche EUR 5.00
</p>

</body>
</html>
```

GHrundsätzlich ist das, was Sie erwarten würden. Zope fügt einen Header, einen Footer und einen Titel in die Web-Seite ein. DTML holt die Werte für diese Variablen von einer Anzahl verschiedener Orte. Zuerst versucht das *var*-Tag eine Variable im aktuellen Objekt zu finden. Dann sieht es in den Behältern des aktuellen Objekts nach. Dann schaut es in der Web-Anfrage nach (Formulare und Cookies). Wenn Zope eine Variable nicht finden kann, gibt es eine Ausnahme aus und hält die Ausführung von DTML an.

Lassen Sie uns diesen DTML-code Schritt für Schritt folgen, um zu sehen, wo die Variablen gefuinden werden. Zuerst schaut Zope nach dem *standard_html_header* im aktuellen Objekt, dem DTML-Skript *Preisliste*. Als nächstes sieht Zope im Behälter des aktuellen Objekts nach. Der Ordner *Futtertaschen* hat auch keine Skripte oder Eigenschaften oder Unter-Objekte mit diesem Namen. Als nächstes untersucht Zope den Behälter des Ordners *Futtertasche* und so weiter, bis es zum Root-Ordner kommt. Der Root-Ordner hat ein Unter-Objekt namens *standard_html_header*. Das Header-Objekt ist ein DTML-Skript. Also ruft Zope das Header-Skript auf und fügt die Ergebnisseein.

Als nächstes sucht Zope nach der Variablen *title*. Zuerst sieht es im DTML-Skript *Preisliste* nach, die keinen Titel hat, also sucht Zope weiter, findet des Titel des Ordners *Futtertasche* und setzt ihn ein.

Schließlich sucht Zope nach der Variable *standard_html_footer*. Es muß den ganzen Weg bis hinauf zum Root-Ordner nachsehen, genau wie bei der Suche nach dem *standard_html_header*.

Diese Übung mag ein wenig weitschweifig erscheinen, aber zu verstehen, wie Zope Variablen findet, ist sehr wichtig. Manche wichtigen Implikationen davon, wie Zope Variablen auffindet beziehen ein, wie Zope-Objekte Inhalte und Verhalten von ihren übergeordneten Objekten übernehmen und wie Inhalte, die an einem Ort definiert sind, von vielen Objekten wieder benutzt werden können.

Eingaben aus Formularen verarbeiten

Formularverarbeitung mit Zope ist einfach. DTML sucht nach Variablen, um sie an einer Anzahl von Orten abzulegen, einschließlich Informationen aus übertragenen HTML-Formularen. Sie brauchen keinerlei spezielle Objekte, DTML-Dokumente und DTML-Skripte werden ausreichen.

Legen Sie zwei DTML-Dokumente an, eine mit der "id" *infoForm*, die andere mit der "id" *infoAction*. Bearbeiten Sie nun die Inhalte der Dokumente. Hier sind die Inhalte des Dokuments *infoForm*:

```
      <dtml-var standard_html_header>

      <p>
      Bitte senden Sie mir Informationen über ihr Erdferkel-
Adoptionsprogramm
      </p>

      <form action="infoAction">
        Name: <input type="text" name="user_name"><br>
        eMail: <input type="text" name="email"><br>
        <input type="submit" name="Absenden">
      </form>

      <dtml-var standard_html_footer>
```

Sehen Sie sich dieses Dokument nun an. Es ist ein Web-Formular, das um Informationen bittet und an das Dokument *infoAction* sendet, wenn Sie das Formular abschicken.

Bearbeiten sie nun den Inhalt des Dokuments *infoAction*, damit es das Formular verarbeiten kann:

```
      <dtml-var standard_html_header>

      <h1>Vielen Dank, <dtml-var user_name>.</h1>

      <p>
      Wir haben ihre Anfrage nach Informationen erhalten und werden Ihnen
eine eMail
      an <dtml-var email> schicken, in der unser Erdferkel-
Adoptionsprogramm
      beschrieben wird, sobald es die abschließende Regierungsgenehmigung
erhalten hat.
      </p>

      <dtml-var standard_html_footer>
```

Dieses Dokument zeigt eine Dankes-Nachticht an, die den Namens- und eMail-Informationen enthalten, die aus dem Web-Formular stammen.

Gehen Sie nun zurück zum *infoForm*-Dokument, sehen Sie es sich an, füllen Sie das Formular aus und senden Sie es ab. Wenn alles klargeht, sollten Sie eine Dankes-Nachricht sehen, die ihrenn Namen und ihre eMail-Adresse enthält.

Das *infoAction*-Dokument hat die Formular-Informationen aus der Web-Anfrage gefunden, die gestellt wurde, als Sie den "Absenden"-Button in *infoForm* geklickt haben. Wie wir im letzten Abschnitt erwähnten, sucht DTML an einer Reihe von Orten nach Variablen, von denen einer die Web-Anfrage ist, es gibt also nichts besonderes zu tun, um ihre Dokumente zur Verarbeitung von Web-Formularen zu befähigen.

Lassen Sie uns ein Experiment machen. Was passiert, wenn Sie das *infoAction*-Dokument direkt ansehen, im Gegensatz dazu, es über das *infoForm*-Dokument zu bekommen? Klicken Sie auf das *infoAction*-Dokument und dann auf die Registerkarte "View", wie in Bild 4.1 gezeigt.

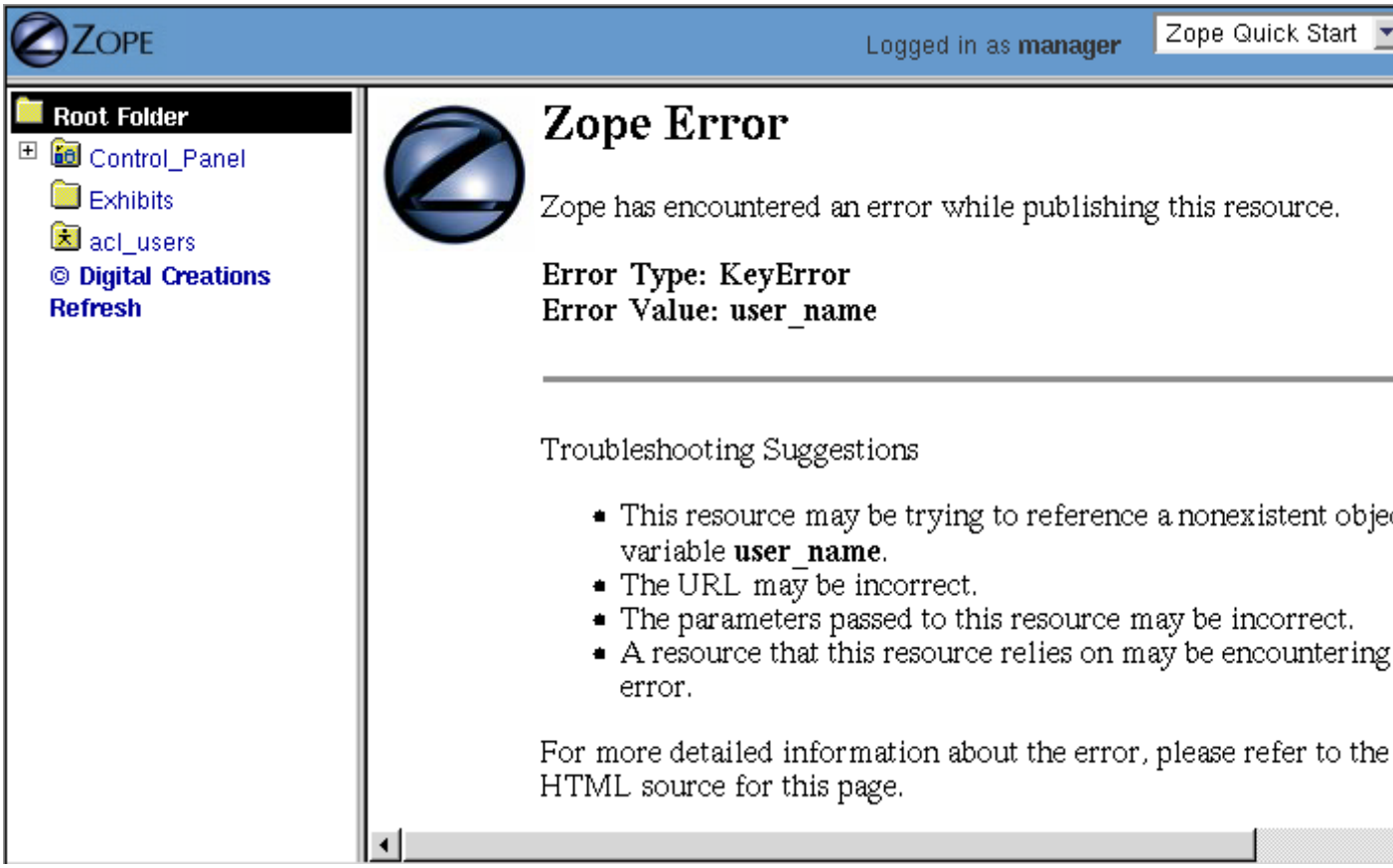


Bild 4.1 DTML-Fehler aus einer gescheiterten Variablen-Suche.

Zope konnte die Variable *user_name* nicht finden, weil sie nicht im aktuellen Objekt, seinen Behältern oder der Web-Anfrage enthalten war. Das ist ein Error, den Sie gelegentlich sehen werden, wenn Sie Zope erlernen. Keine Angst, es heißt nur, daß Sie versucht haben, eine Variable einzufügen, die Zope nicht finden konnte. In diesem Beispiel müssen Sie entweder eine Variable einsetzen, die von Zope gefunden werden kann oder - wie oben beschrieben - im var-Tag das *missing*-Attribut verwenden:

```
      <h1>Vielen Dank, <dtml-var user_name missing="anonymer
Benutzer"></h1>
```

Zu verstehen, wo Zope nach Variablen sucht, wird ihnen helfenProbleme solcher Art zu lösen. In diesem Fall haben Sie ein Dokument angesehen, das eigentlich erst von einem HTML-Formular wie *infoForm* aufgerufen werden muß, um die Variablen zur Verfügung zu stellen, die in die Ausgabe eingefügt werden sollen.

Dynamisch erworbener Inhalt

Zope sucht in den Behältern des aktuellen Objekts nach DTML-Variablen, wenn es die Variable nicht im aktuellen Objekt finden kann. Dieses Verhalten erlaubt ihren Objekten, Inhalte und Verhaltensregeln zu finden und zu benutzen, die in den übergeordneten Objekten definiert sind. Zope benutzt den Begriff des *Erwerbens* (acquisition), um sich auf diesen dynamischen Gebrauch von Inhalten und Verhaltensregeln zu beziehen.

Während Sie nun sehen, wie die Site-Struktur zum dem Weg paßt, auf dem Namen nachgeschlagen werden, können Sie beginnen zu verstehen, daß es sehr wichtig ist, wo Sie Objekte plazieren, nach denen Sie suchen.

Ein Beispiel für Erwerben, das Sie schon kennen, ist, wie Web-Seiten Standard-Header und -Footer benutzen. Um den Standard-Header zu erwerben, bitten Sie Zope nur, es mit dem *var*-Tag einzusetzen:

```
      <dtml-var standard_html_header>
```

Es spielt keine Rolle, wo ihr DTML-Skript oder -Dokument liegt. Zope wird aufwärts suchen, bis es de *standard_html_header* findet, der im Root-Ordner definiert ist.

Sie können die Art nutzen, wie Zope Variablen findet, um ihren Header für verschiedene Bereiche ihrer Site einzurichten. Legen Sie nur einen neuen *standard_html_header* in einem Ordner an und er wird den allgemeinen Header für diesen Ordner und alle darunter ersetzen.

Legen Sie einen Ordner im Root-Ordner mit der "id" *Grün* an. Öffnen Sie den Ordner *Grün* und legen Sie ein DTML-Dokument mit der "id" *Willkommen* an. Bearbeiten Sie das *Willkommen*-Dokument, bis es die folgenden Inhalte hat:

```
      <dtml-var standard_html_header>

      <p>
      Willkommen
      </p>

      <dtml-var standard_html_footer>
```

Sehen Sie sich das *Willkommen*-Dokument nun an. Es sollte wie eine einfache Web-Seite mit dem Wort *Willkommen* aussehen, wie in Bild 4.2.

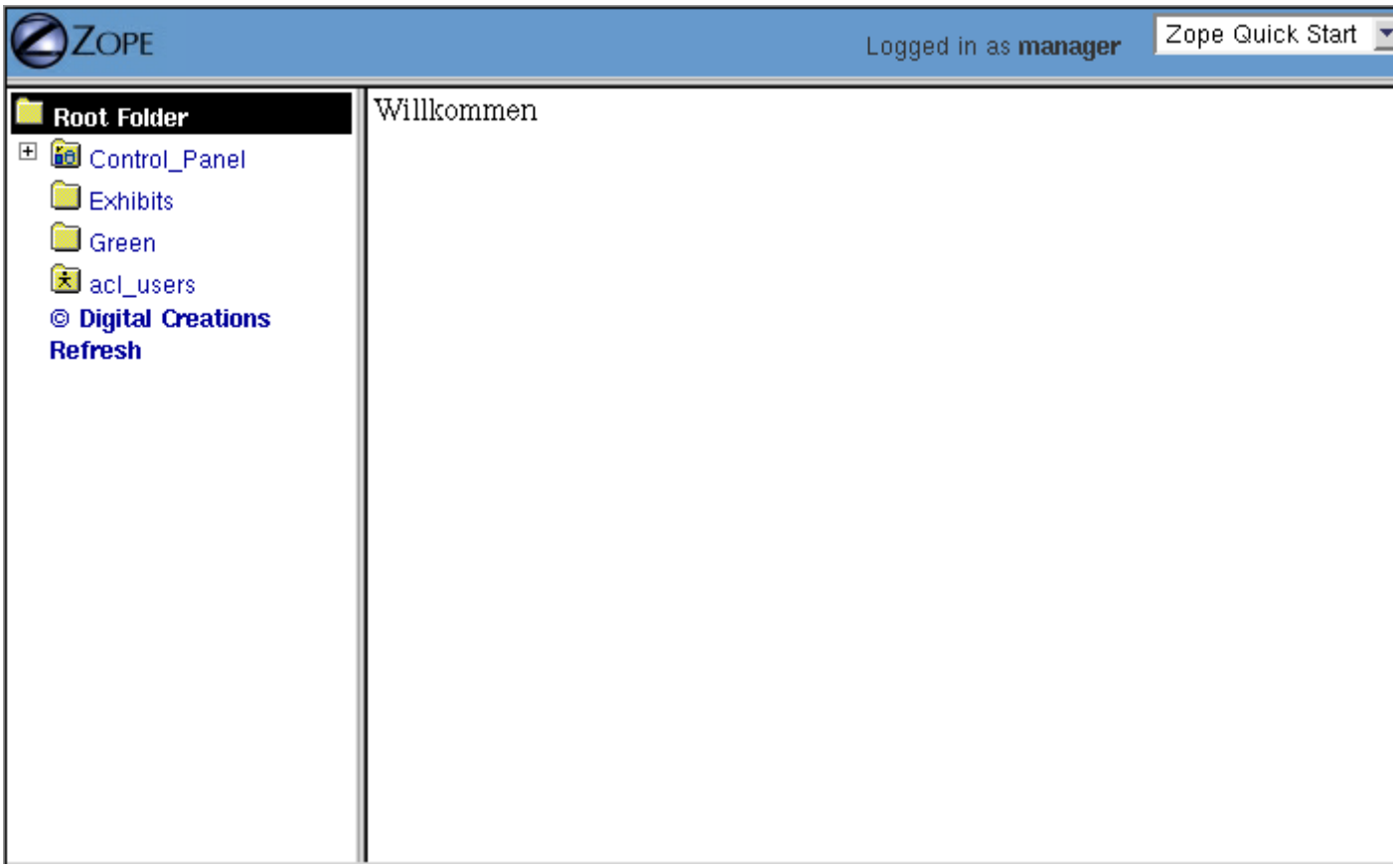


Bild 4.2 Willkommen-Dokument.

Lassen Sie uns nun den Header für den Ordner *Grün* einrichten. Legen Sie ein DTML-Skript mit der "id" *standard_html_header* im Ordner *Grün* an. Dann bearbeiten Sie den Inhalt dieses Headers folgendermaßen:

```
<html>
<head>
  <style type="text/css">
    body {color: #00FF00;}
    p {font-family: sans-serif;}
  </style>
</head>
<body>
```

Beachten Sie, daß dies keine komplette Web-Seite ist. Es ist nur ein HTML-Fragment, das als Header benutzt werden soll. Dieser Header benutzt CSS (Cascading Style Sheets), um einige Änderungen in Aussehen und Verhalten von Web-Seiten vorzunehmen.

Gehen Sie nun zurück zum *Willkommen*-Dokument und sehen Sie es sich - wie in Bild 4.3 gezeigt - noch einmal an.

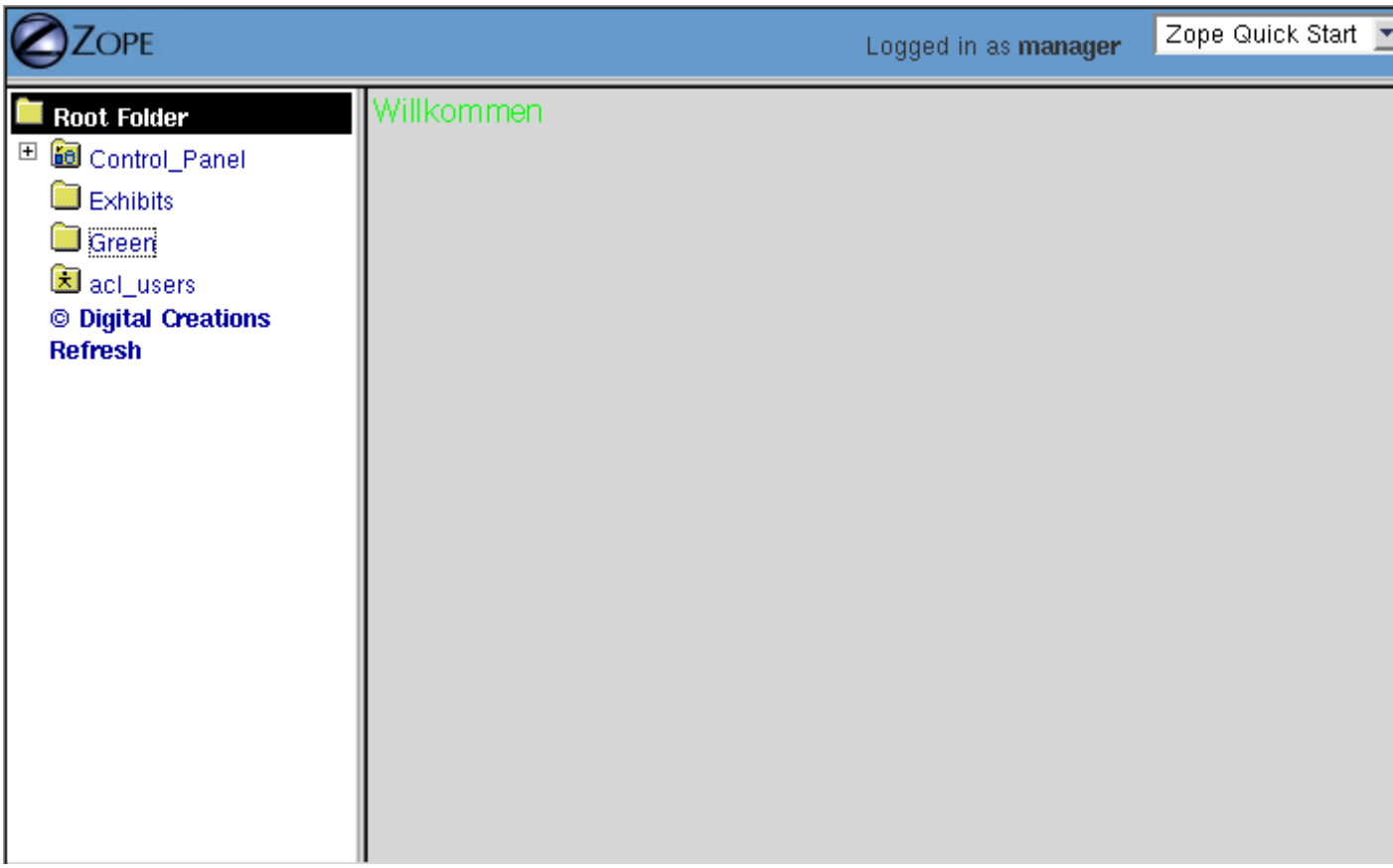


Bild 4.3 Willkommen-Dokument mit eingerichtetem Header.

Das Dokument sieht nun ganz anders aus. Das kommt daher, daß es nun den neuen Header benutzt, den wir dem Ordner *Grün* hinzugefügt haben. Dieser Header wird von allen Web-Seiten im Ordner *Grün* und seinen Unterordnern benutzt werden.

Sie können diesen Prozeß weiterführen, indem Sie im Ordner *Grün* einen weiteren Ordner und darin ein weiteres DTML-Skript *standard_html_header* anlegen. Nun werden Web-Seiten in dem Unterordner ihren lokalen Header benutzen, anstelle des Headers im Ordner *Grün*. Mit diesem Muster können Sie Aussehen und Verhalten ihrer Web-Site schnell ändern. Wenn Sie später beschließen, daß ein Bereich der Site einen anderen Header braucht, legen Sie einfach einen an. Sie müssen keinerlei Änderungen im DTML von einer der Web-Seiten vornehmen; sie werden automatisch den nächsten Header finden und benutzen.

Python-Ausdrücke in DTML benutzen

Bis hierher haben wir einfache DTML-Tags betrachtet. Hier ist ein Beispiel:

```
<dtml-var getNilpferd>
```

Dies wird den Wert der Variable mit der Benennung *getNilpferd* einsetzen, was auch immer das sein mag. DTML wird sich automatisch um die Details wie Finden der Variable und ihren

Aufruf kümmern. Wir nennen diese grundlegende Tag-Syntax *Namens*-Syntax, im Unterschied zu *Ausdrucks*-Syntax.

DTML-Ausdrücke erlauben Ihnen, genauer damit zu sein, wie Variablen gefunden und aufgerufen werden. Ausdrücke sind Tag-Attribute, die Code-Stückchen in der Python-Sprache enthalten. Zum Beispiel können wir - anstatt DTML *getNilpferd* finden und aufrufen zu lassen - auch einen Ausdruck benutzen, um exakte Argumente zu übergeben:

```
<dtml-var expr="getNilpferd('mit einem grossen Netz')">
```

Wir haben hier einen Python-Ausdruck benutzt, um das *getNilpferd*-Skript explizit mit einem String-Argument aufzurufen: `mit einem grossen Netz`. Um mehr über Pythons Syntax herauszufinden, sehen Sie sich das Python Tutorial auf der Website der Python.org an. Viele DTML-Tags können Ausdrucks-Attribute benutzen.

Ausdrücke machen DTML ziemlich mächtig. Indem Sie Python-Ausdrücke benutzen, können Sie zum Beispiel ganz einfach Bedingungen testen:

```
<dtml-if expr="foo < bar">
  Foo ist weniger als bar.
</dtml-if>
```

Ohne Ausdrücke würde diese sehr einfache Aufgabe in ein separates Skript aufgebrochen werden müssen und würde eine Menge Übergewicht für etwas produzieren, das derartig simpel ist.

Bevor Sie sich in Ausdrücken verlieren, seien Sie vorsichtig. Ausdrücke können es schwer machen, ihr DTML zu verstehen. Code, der schwierig zu verstehen ist, kann eher Fehler enthalten und ist schwieriger zu warten. Ausdrücke können auch zu vermischter Logik in ihrer Präsentation führen. Wenn Sie bemerken, daß Sie einen Ausdruck länger als fünf Sekunden anstarren müssen, halten Sie an. Schreiben Sie das DTML ohne den Ausdruck neu und benutzen Sie ein Skript, um ihre Logik umzusetzen. Nur weil es möglich ist, komplexe Dinge mit DTML tun, sollten Sie das noch lange nicht machen.

Fallstricke in DTML-Ausdrücken

Der Umgang mit Python-Ausdrücken kann haarig sein. Ein verbreiteter Fehler ist die Verwechslung von Ausdrücken mit grundsätzlicher Tag-Syntax. Zum Beispiel Ihnen:

```
<dtml-var objectValues>
```

und:

```
<dtml-var expr="objectValues">
```

zwei völlig verschiedene Ergebnisse liefern. Das erste Beispiel des DTML-*var*-Tags wird die Variablen automatisch interpretieren. In anderen Worten: Es wird versuchen, das Richtige zu tun, um ihre Variable einzufügen, egal was diese Variable sein mag. Generell heißt das, daß wenn die Variable ein Skript ist, sie mit den entsprechenden Argumenten aufgerufen wird. Dieser Prozeß wird tiefer in Kapitel 8 (Variablen und fortgeschrittenes DTML) behandelt.

In einem Ausdruck haben Sie völlige Kontrolle über die Interpretation einer Variablen. Im Fall unseres Beispiels ist *objectValues* ein Skript. Also:

```
<dtml-var objectValues>
```

wird ein Skript aufrufen. Aber

```
<dtml-var expr="objectValues">
```

wird *kein* Skript aufrufen, sondern einfach versuchen, es einzufügen. Das Ergebnis wird keine Liste von Objekten sein, sondern ein String wie etwa `<Python Method object at 8681298>`. Wann immer Sie Ergebnisse wie dieses sehen, es es gut möglich, daß ein Skript zurückgegeben anstatt aufgerufen wird.

Um ein Skript über einen Ausdruck aufzurufen, muß die Standard-Aufruf-Syntax von Python mit Klammern benutzt werden:

```
<dtml-var expr="objectValues()">
```

Die Lektion besteht darin, daß Sie beim Benutzen von Python-Ausdrücken wissen sollten, was für eine Art Variable Sie einsetzen und die passende Python-Syntax benutzen, damit die Variable entsprechend interpretiert werden kann.

Bevor wir das Thema der Ausdrucks-Variablen verlassen, sollten wir erwähnen, daß es eine abzulehnende Form der Ausdruckssyntax gibt. Sie können den Teil "expr=" in einem Ausdrucks-Tag für eine Variable weglassen. Aber *bitte* tun Sie das nicht. Es ist viel zu leicht, dabei folgendes zu verwechseln, nämlich:

```
<dtml-var aName>
```

mit:

```
<dtml-var "aName">
```

und dabei völlig verschiedene Resultate zu bekommen. Diese "Abkürzungen" wurden vor langer Zeit in DTML eingebaut, aber wir bestärken Sie nicht darin, Sie zu benutzen, wenn Sie

nicht bereit sind, das Durcheinander und die Bereinigungsprobleme in Kauf zu nehmen, die aus diesem subtilen Unterschied der Syntax kommen können.

Das *var*-Tag

Das *var*-Tag setzt Variable in DTML-Skripts und -Dokumente ein. Wir haben schon viele Beispiele gesehen, wie das *var*-Tag benutzt werden kann, um Strings in Web-Seiten einzufügen.

Wie Sie gesehen haben, sucht das *var*-Tag zuerst im aktuellen Objekt, dann in seinen Behälternm und schließlich in der Web-Anfrage nach den Variablen.

Das *var*-Tag kann auch Python-Ausdrücke benutzen, um mehr Kontrolle über das Auffinden und Aufrufen von Variablen zu gewähren.

var-Tag-Attribute

Sie können das Verhalten des *var*-Tags über seine Attribute beeinflussen. Das *var*-Tag hat viele Attribute, die Ihnen in häufigen Formatierungssituationen helfen. Die Attribute sind in Anhang A zusammengefaßt. Hier ist eine Zusammenstellung von *var*-Tag-Attributen.

html_quote
: Dieses Attribut läßt die eingesetzten Werte als HTML-Zitate erscheinen. Das bedeutet, daß `<`, `>` und `&` nicht dargestellt werden.

missing
: Das Attribut "missing" erlaubt Ihnen die Spezifizierung eines vorgegebenen Wertes, der benutzt wird, wenn Zope die Variable nicht finden kann. Zum Beispiel:

```
<dtml-var bananen missing="Bananen hamwanich">
```

fmt
: Das Attribut "fmt" erlaubt Ihnen, das Format der *var*-Tag-Ausgabe zu beeinflussen. Es gibt viele verschiedene Formate, die genauer im Anhang A beschrieben sind.

  Ein Nutzen des *fmt*-Attributs liegt in der Formatierung von Währungs-Werten. Legen Sie zum Beispiel eine *float*-Eigenschaft namens *erwachsen_preis* in ihrem Root-Ordner an. Diese Eigenschaft stellt die Kosten für den Zoobesuch eines Erwachsenen dar. Geben Sie dieser Eigenschaft den Wert `2.2`.

  Sie können diese Kosten in einem DTML-Dokument oder -Skript etwa so ausgeben:

```
Eine Erwachsenen-Karte: <dtml-var erwachsen_preis fmt=dollars-
and-cents>
```

  Damit wird ausgegeben: "$2.20". Genauere Werte werden kaufmännisch auf zwei Hinter-Komma-Stellen gerundet.

Syntax von *var*-Tag-Ausdrücken

Zope bietet für das einfache *var*-Tag eine verkürzte Syntax an. Weil das *var*-Tag ein Singleton ist, kann es durch eine Syntax ähnlich einem *HTML-Ausdruck* dargestellt werden:

```
&dtml-cockatiel;
```

Das entspricht:

```
<dtml-var name="cockatiel" html_quote>
```

Der Hauptgrund, Ausdruckssyntax zu verwenden, ist das Vermeiden von DTML-Tags innerhalb von HTML-Tags. Anstatt beispielsweise zu schreiben:

```
<input type="text" value="<dtml-var name="defaultValue">">
```

können Sie die Ausdruckssyntax verwenden, um die Dinge für Sie und ihren Text-Editor besser lesbar zu machen:

```
<input type="text" value="&dtml-defaultValue;">
```

Die *var*-Tag-Ausdruckssyntax ist sehr begrenzt. Sie können Python-Ausdrücke nicht zusammen mit einigen Tag-Attributen verwenden. In Anahng A finden Sie mehr Informationen zur *var*-Tag-Ausdruckssyntax.

Das *if*-Tag

Einer der großen Vorteile von DTML ist die Möglichkeit, ihre Web-Seiten anzupassen. Oft bedeutet "Anpassung", daß die Prüfabfragen angemessen laufen. Dieses *if*-Tag läßt Sie eine Bedingung auswerten und aufgrund des Ergebnisses verschiedene Aktionen ausführen.

Was ist eine Bedingung? Eine Bedingung ist ein Wert, der entweder wahr oder falsch ist. Generell gelten alle Objekte als wahr, es sei denn sie wären 0, Nichts, eine leere Sequenz oder ein leerer String.

Hier ist eine Beispiels-Bedingung:

objectValues
: Wahr, wenn die Variable *objectValues* existiert und wahr ist. Das heißt: Wenn nach Auffinden und Auswertens *objectValues* nicht 0, Nichts, eine leere Sequenz oder ein leerer String ist.

Genau wie beim *var*-Tag können Sie hier Namens- und Ausdrucks-Syntax verwenden. Hier sind einihe Bedingungen als DTML-Ausdrücke dargestellt.

expr="1"

Immer wahr.

expr="Nashorn"

Wahr, wenn die Variable "Nashorn" wahr ist.

expr="x < 5"

Wahr, wenn x kleiner als 5 ist.

expr="objectValues('File')"

Wahr, wenn der Aufruf des Skripts *objectValues* mit dem Argument *File* einen wahren Wert zurückgibt. Dieses Skript wird im Kapitel genauer erläutert.

Das *if*-Tag ist ein Block-Tag. Der Block innerhalb des ist *if*-Tags wird ausgeführt, wenn die Bedingung wahr ist.

Hier sehen Sie, wie Sie einen Variablen-Ausdruck mit dem *if*-Tag benutzen können, um eine Bedingung zu prüfen:

```
<p>Wie viele Affen gibt es?</p>

<dtml-if expr="Affen > Affen_limit">
  <p>
  Es gibt zu viele Affen!
  </p>
</dtml-if>
```

Im obigen Beispiel werden Sie den ersten und zweiten HTML-Absatz sehen, wenn der Python-Ausdruck `Affen > Affen_limit` wahr ist. Wenn die Bedingung unwahr ist, werden Sie nur den ersten sehen.

if-Tags können zu jeder Tiefe verschachtelt werden, zum Beispiel könnten Sie folgendes haben:

```
<p>
Gibt es zu viele blaue Affen?
</p>

<dtml-if "Affen.Farbe == 'blau'">
  <dtml-if expr="Affen > Affen_limit">
    <p>
    Es gibt zu viele blaue Affen!
    </p>
  </dtml-if>
</dtml-if>
```

Verschachtelte *if*-Tags arbeiten sich durch die Auswertung der ersten Bedingung, und wenn diese Bedingung wahr ist, werten sie die nächste aus. Im Allgmeinen arbeiten *if*-Tags in DTML genauso wie *if*-Statements in Python.

Unterschiede zwischen Namens- und Ausdrucks-Syntax

Die Namens-Syntax prüft sowohl gegen die *Existenz* eines Namens als auch gegen seinen Wert. Zum Beispiel:

```
<dtml-if Affen_haus>
  <p>
  Da <em>ist</em> ein Affenhaus, Mama!
  </p>
</dtml-if>
```

Wenn die Variable *Affen_haus* nicht existiert, ist die Bedingung unwahr. Wenn es eine Variable *Affen_haus* gibt, aber sie unwahr ist, ist diese Bedingung ebenso unwahr. Die Bedingung ist nur dann wahr, wenn es die Variable *Affen_haus* gibt und sie weder 0 noch Nichts, eine leere Sequenz oder ein leerer Sting ist.

Die Python-Ausdrucks-Syntax prüft nicht gegen die Existenz einer Variablen. Das kommt daher, daß der Ausdruck gültiges Python sein muß. Zum Beispiel:

```
<dtml-if Affen_haus>
  <p>
  Da <em>ist</em> ein Affenhaus, Mama!
  </p>
</dtml-if>
```

Das wird solange wie erwartet arbeiten, wie *Affen_haus* existiert. Wenn die Variable *Affen_haus* nicht existiert, wird Zope eine *KeyError*-Ausnahme zurückgeben, sobald es versucht, die Variable zu finden.

Else- und *Elif*-Tags

Das *if*-Tag läßt nur dann eine Aktion zu, wenn eine Bedingung wahr ist. Sie mögen auch bei einer unwahren Bedingung eine andere Aktion aufrufen wollen. Das kann mit Hilfe des DTML-*else*-Tags geschehen. Der *if*-Block kann auch ein *else*-Singleton-Tag enthalten. Zum Beispiel:

```
<dtml-if expr="Affen > Affen_limit">
   <p>
   Es gibt zuviele Affen!
   </p>
<dtml-else>
   <p>
   Die Affen sind fröhlich!
   </p>
</dtml-if>
```

Das *else*-Tag spaltet den *if*-Tag-Block in zwei Böcke, der erste wird ausgeführt, wenn die Bedingung wahr ist, der zweite wird ausgeführt, wenn die Bedingung nicht wahr ist.

Ein *if*-Tag-Block kann auch ein *elif*-Singleton-Tag enthalten. Das *elif*-Tag spezifiziert - wie ein zusätzlichs *if*-Tag eine weitere Bedingung. Das läßt Sie mehrere Bedingungen in einem Block darstellen:

```
<dtml-if expr="Affen > Affen_limit">
  <p>
  Es gibt zuviele Affen!
  </p>
<dtml-elif expr="Affen < Affen_minimum">
  <p>
  Es gibt nicht genug Affen!
  </p>
<dtml-else>
  <p>
  Es gibt gerade genug Affen.
  </p>
</dtml-if>
```

Ein *if*-Tag-Block kann jede Anzahl von *elif*-Tags enthalten, aber nur ein *else*-Tag. Das *else*-Tag muß immer hinter den *elif*-Tags stehen. *elif*-Tags können können Bedingungen sowohl mit Namens- als auch mit Ausdrucks-Syntax prüfen.

Cookies im *if*-Tag benutzen

Lassen Sie uns ein etwas gehaltvolleres *if*-Tag-Beispiel betrachten. Wenn Sie Besucher auf ihrer Site haben, werden Sie ihnen oft ein Cookie geben wollen, um sie mit irgendeiner Art speziellem Wert zu identifizieren. Cookies werden überall im Internet gelegentlich verwendet und wenn sie richtig benutzt werden, sind sie sehr nützlich.

Nehmen Sie an, wir wollen neue Besucher von Leuten unterscheiden, die schon auf unserer Site waren. Wenn ein Benutzer die Site besucht, können wir einen Cookie setzen. Dann können wir bei der Seitenanzeige gegen den Cookie prüfen. Wenn der Benutzer schon auf der Site war, wird er diesen Cookie haben. Wenn er den Cookie noch nicht hat, bedeutet es, daß er neu ist.

Angenommen wir haben etwas Besonderes. Erstbesucher kommen zum halben Preis in den Zoo. Hier ist ein DTML-Fragment, das mit Hilfe der Variablen *warSchonImZoo* gegen einen Cookie prüft und den Eintrittspreis gemessen daran anzeigt, ob der Benutzer schonmal im Zoo war oder nicht:

```
<dtml-if warSchonImZoo>
  <p>
  Zoo-Eintritt <dtml-var erwachsen_preis fmt="dollars-and-cents">.
  </p>
<dtml-else>
  <p>
  Zoo-Eintritt für Erstbesucher <dtml-var expr="erwachsen_preis/2"
fmt="dollars-and-cents">
  </p>
</dtml-if>
```

Dieses Fragment prüft gegen die Variable *warSchonImZoo*. Wenn der Benutzer den Zoo schon besucht hat , gibt es den normalen Eintrittspreis aus. Wenn der Besucher zum ersten Mal hier ist, kommt er zum halben Preis rein.

Nur der Vollständigkeit halber ist hier noch eine Ausführung des *warSchonImZoo*-Skripts auf Python-Basis:

```
## Script(Python) "warSchonImZoo"
## parameters=REQUEST, RESPONSE
##
"""
Gibt wahr zurück, wenn der Benutzer den Zoo schon benutzt hat
Benutzt Cookies, um Zoo-Besuche zu verfolgen.
"""
if REQUEST.has_key('zooBesuchCookie'):
    return 1
else:
    RESPONSE.setCookie('zooBesuchCookie', '1')
    return 0
```

In Kapitel 10 (Fortgeschrittenes Zope-Scripting) werden wir einen genaueren Blick darauf werfen, wie Geschäftslogik mit Python oder Perl umgesetzt wird. Bis hierher reicht es, zu sehen, daß das Skript nach einem Cookie sucht und - abhängig vom Auffinden des Cookies - einen wahren oder unwahren Wert zurückgibt. Beachten Sie, wie Python *if*- und *else*-Statements benutzt, genau wie DTML *if*- und *else*-Tags. *if*- und *else*-Tags in DTML basieren auf denen von Python. Tatsächlich hat Python auch ein *elif*-Statement, genau wie DTML.

Das *in*-Tag

Das *in*-Tag von DTML iteriert durch eine Folge von Objekten und führt für jeden Gegenstand in der Folge einen Prozeß-Block aus. Beim Programmieren wird das oft *Iteration* oder *Looping* genannt.

Das *in*-Tag ist ein Block-Tag wie das *if*-Tag. Der Inhalt des *in*-Tags wird für jede Iteration des *in*-Tag-Kreises einmal ausgeführt. Zum Beispiel:

```
<dtml-in Aufgaben_liste>
  <p>
  <dtml-var Beschreibung>
  </p>
</dtml-in>
```

Dieses Beispiel kreist durch eine Liste von Objekten, die *Aufgaben_liste* heißt. Für jeden Gegenstand fügt es einen HTML-Absatz mit einer Beschreibung des Aufgaben-Gegenstandes ein.

In vielen Web-Aufgaben ist Iteration sehr nützlich. Stellen Sie sich eine Web-Site vor, wo Häuser zum Kauf angeboten werden. Die Benutzer werden auf ihrer Site nach Häusern suchen, die verschiedene Kriterien erfüllen. Sie werden all diese Ergebnisse in konsitenter Weise auf ihrer Seite fomatieren wollen, weil Sie über jedes einzelne Suchergebnis iterieren müssen und einen ähnlichen HTML-Block für jedes Ergebnis ausgeben wollen.

Die Inhalte eines *in*-Tag-Blocks sind eine Art von *Template*, das einmal auf jeden Gegenstand der Sequenz angewendet wird.

Durch Ordnerinhalte iterieren

Hier ist ein Beispiel davon, durch die Inhalte eines Ordners zu iterieren. Dieses DTML wird durch alle Dateien in einem Ordner kreisen und einen Link zu jedem einzelnen anzeigen. Dieses Beispiel zeigt Ihnen, wie alle "File"-Objekte in einem Ordner angezeigt werden, also werden Sie eine Dateien in Zope laden müssen wie im vorherigen Kapitel besachrieben:

```
<dtml-var standard_html_header>
<ul>
  <dtml-in expr="objectValues('File')">
    <li><a href="&dtml-absolute_url;"><dtml-var title_or_id></a></li>
  </dtml-in>
</ul>
<dtml-var standard_html_footer>
```

Dieser Code zeigt die folgende Datei-Liste, wie in Bild 4.4.

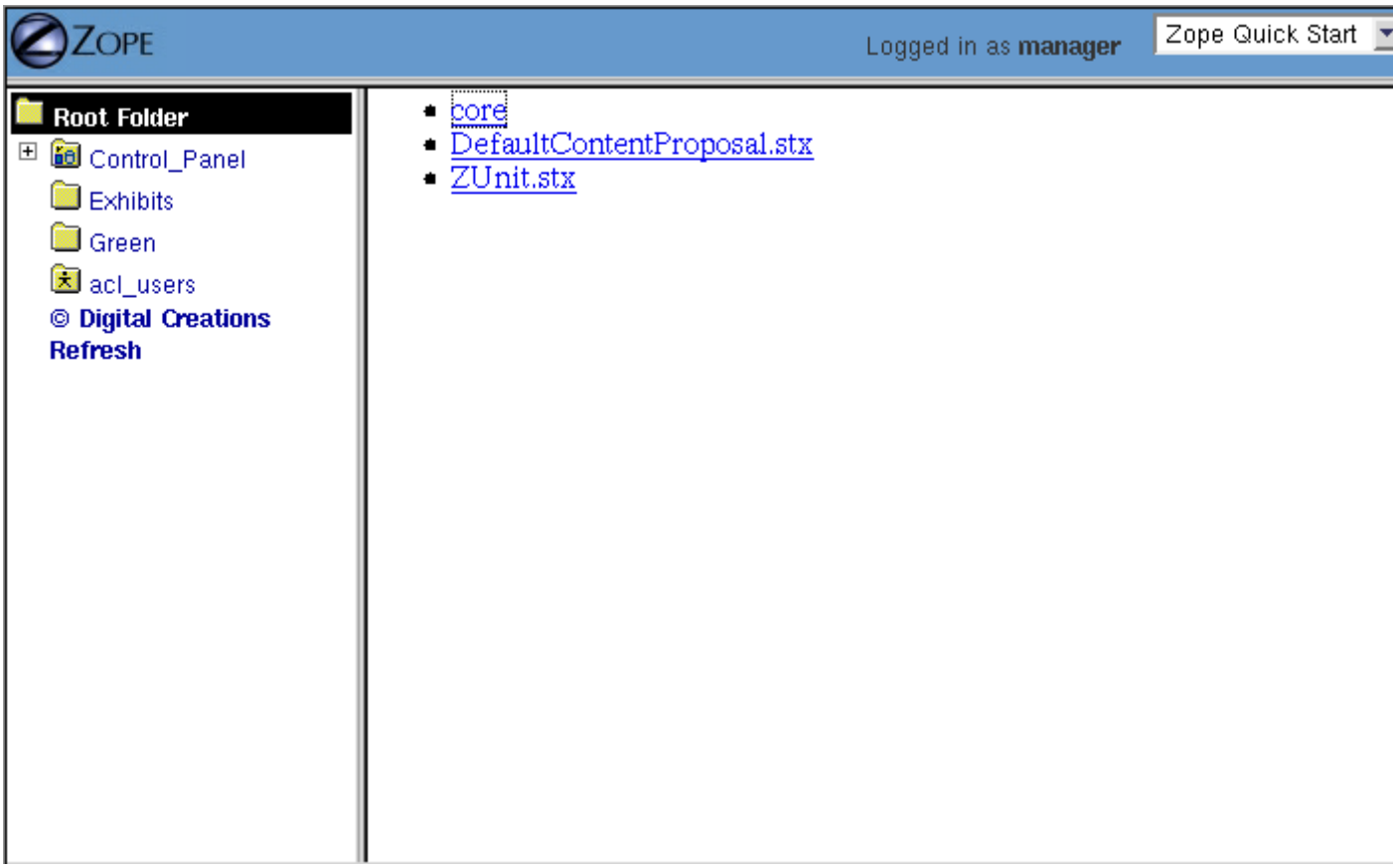


Bild 4.4 Über eine Liste von Dateien iterieren.

Lassen Sie uns dieses DTML-Beispiel Schritt für Schritt ansehen. Zuerst wird das *var*-Tag benutzt, um ihren gewohnten Header in das Dokument einzufügen. Als nächstes haben Sie das *ul*-HTML-Tag, um anzuzeigen, daß der Browser eine Punktliste anzeigen soll.

Dann gibt es das *in*-Tag. Das Tag hat einen Ausdruck, der das Zope-API-Skript *objectValues* aufruft. Dieses Skript gibt eine Sequenz von Objekten im aktuellen Ordner zurück, die bestimmte Kriterien erfüllen. In diesem Fall müssen die Objekte Dateien (Files) sein. Es wird durch alle Gegenstände im aktuellen Ordner kreisen.

Das *in*-Tag wird durch jeden Gegenstand in dieser Sequenz kreisen. Wenn es vier Datei-Objekte im aktuellen Ordner gibt, , wird das *in*-Tag den Code in seinem Block viermal ausführen; einmal für jedes Objekt in der Sequenz.

Während jeder Iteration sucht das *in*-Tag nach Variablen, zuerst im aktuellen Objekt. In Kapitel 8 (Variablen und fortgeschrittenes DTML) werden wir uns näher damit befassen, wie DTML Variablen findet.

Zum Beispiel iteriert dieses *in*-Tag durch eine Sammlung von Datei-Objekten und benutzt das *var*-Tag zum Auffinden der Variablen in jeder Datei:

```
<dtml-in expr="objectValues('File')">
  <li><a href="&dtml-absolute_url;"><dtml-var title_or_id></a></li>
</dtml-in>
```

Das erste *var*-Tag ist ein Ausdruck und das zweite ein normales DTML-*var*-Tag. Sobald das *in*-Tag über das erste Objekt kreist, werden seine Variablen *absolute_url* und *title_or_id* in den ersten Gegenstand der Punktliste eingefügt:

```
<ul>
  <li><a href="http://localhost:8080/FirstFile">FirstFile</a></li>
```

Während der zweiten Iteration werden die Variablen des zweiten Objekts - *absolute_url* und *title_or_id* - in die Ausgabe eingefügt:

```
<ul>
  <li><a href="http://localhost:8080/FirstFile">FirstFile</a></li>
  <li><a href="http://localhost:8080/SecondFile">SecondFile</a></li>
```

Dieser Prozeß wird weitergehen, bis das *in*-Tag über jede Datei im aktuellen Ordner iteriert ist. Nach dem *in*-Tag beenden Sie schließlich ihre HTML-Punktliste mit einem schließenden */ul*-HTML-Tag und der *standard_html_footer* wird eingefügt, um das Dokument zu schließen.

Besondere Variablen des *in*-Tags

Das *in*-Tag versorgt Sie mit erinigen nützlichen Informationen, die Ihnen die Anpassung ihrer HTML-Inhalte während des Iterierens über eine Sequenz ermöglichen. Sie können zum Beispiel ihr Dateiverzeichnis leichter lesbar machen, indem Sie es in eine HTML-Tabelle einfügen und jede zweite Tabellenzele in einer anderen Farbe anzeigen lassen, wie die in Bild 4.5 gezeigte.

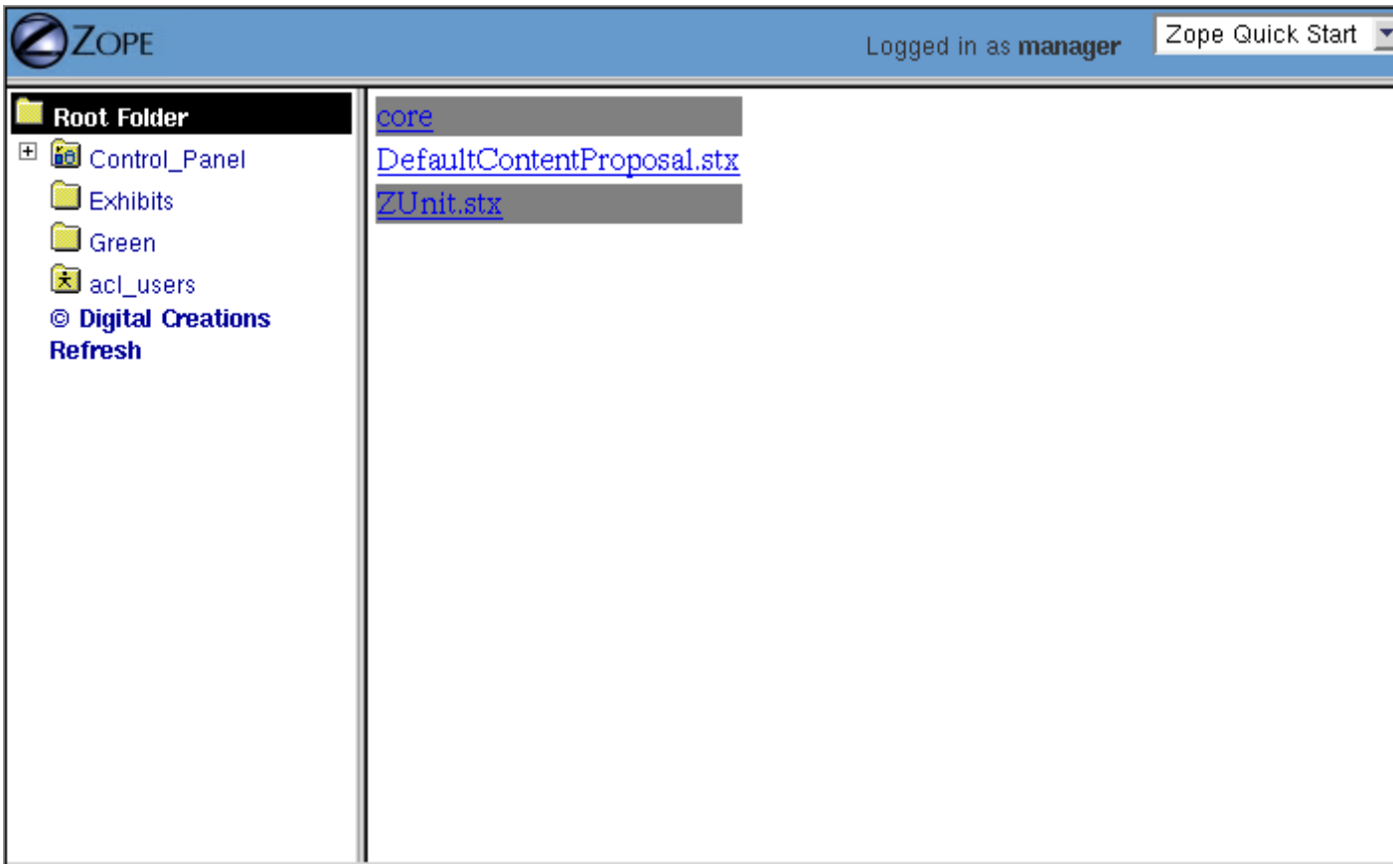


Figure 4.5 Dateilistung mit wechselnden Zeilenfarben.

Das *in*-Tag macht das einfach. Ändern Sie ihr Dateiverzeichnis-Skript ein bißchen, damit es so aussieht:

```
      <dtml-var standard_html_header>

      <table>

        <dtml-in expr="objectValues('File')">

          <dtml-if sequence-even>
            <tr bgcolor="grey">
          <dtml-else>
            <tr>
          </dtml-if>

            <td><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a></td>
          </tr>
        </dtml-in>

      </table>

      <dtml-var standard_html_footer>
```

Hier wird ein *if*-Tag benutzt, um gegen eine spezielle Variable zu prüfen, die `sequence-even` heißt. Das *in*-Tag setzt diese Variable bei jedem Loop auf einen wahren oder unwahren Wert. Wenn die aktuelle Iterationszahl ungerade ist, gilt sie als unwahr.

Als Ergebnis wird abwechselnd je ein *tr*-Tag mit entweder grauem oder ohne Hintergrund in das Dokument eingefügt. Wie erwartet, gibt es eine `sequence-odd`, die immer den entgegengesetzten Wert wie `sequence-even` hat.

Es gibt viele spezielle Variablen, die ein *in*-Tag für Sie definieren kann. Hier sind die gebräuchjlichsten und nützlichsten:

sequence-item
: Diese Spezial-Variable ist der aktuelle Gegenstanmd der Iteration.
Im Fall des Dateiverzeichnis-Beispiels wird jedesmal beim Loop die aktuelle Datei der Iteration als *sequence-item* betrachtet. Es ist oft nützlich, eine Referenz auf das aktuelle Objekt der Iteration zu haben.

sequence-index
: steht für die aktuelle Zahl von abgeschlossenen Iterationen, beginnend bei 0. Wenn diese Zahl gerade ist, ist `sequence-even` wahr und `sequence-odd` ist unwahr.

sequence-number
: Die aktuelle Zahl von abgeschlossenen Iterationen, beginnend bei 1. Dies kann als die kardinale Position (erste, zweite, dritte etc.) des aktuellen Objekts im Loop betrachtet werden. Wenn diese Zahl gerade ist, gilt `sequence-even` als unwahr und `sequence-odd` als wahr.

sequence-start
: Diese Variable ist für die erste Iteration wahr.

sequence-end
: Diese Variable ist für die Schluß-Iteration wahr.

Diese Spezial-Variablen werden detaillierter im Anhang A dargestellt.

DTML ist ein mächtiges Werkzeug zur Schaffung dynamischer Inhalte. Es erlaubt Ihnen, ziemlich komplexe Berechnungen anzustellen. In Kapitel 8 (Variablen und fortgeschrittenes DTML) werden Sie etwas über viel mehr DTML-Tags und mächtigere Wege herausfinden, die Tags zu benutzen, die Sie schon kennengelernt haben. Unabhängig von den Möglichkeiten sollten Sie der Versuchung widerstehen, DTML für komplexe Skripte zu benutzen. In Kapitel 10 (Fortgeschrittenes Zope-Sktipting) werden Sie herausfinden, wie Sie Python und Perl für das Sktipting von Geschäftslogik verwenden können.

Zopebuch: Inhaltsverzeichnis

Kapitel 5: Zope-Seitenvorlagen verwenden

Seitenvorlagen sind ein Werkzeug zum Erzeugen von Webseiten. Sie helfen Programmierern und Designern, beim erstellen dynamischer Webseiten für Zope-Web-Anwendungen zusammenzuarbeiten. Designer können sie verwenden, um Seiten zu pflegen, ohne ihre Werkzeuge aufgeben zu müssen, während sie die ersparte Arbeitszeit nutzen können, um jene Seiten in eine Anwendung einzubetten. In diesem Kapitel lernen Sie die Grundlagen von Seitenvorlagen, einschließlich dem, wie Sie sie in Ihrer Website verwenden können, um

einfach dynamische Webseiten zu erstellen. In Kapitel 9, "Seitenvorlagen für Fortgeschrittene", erlernen Sie fortschrittliche Seitenvorlagen-Funktionen.

Das Ziel von Seitenvorlagen ist es, Designern und Programmierern zu ermöglichen, leicht zusammenzuarbeiten. Ein Designer kann einen WYSIWYG-HTML-Editor verwenden, um eine Vorlage zu erstellen, dann kann sie ein Programmierer überarbeiten, um sie zum Teil einer Anwendung zu machen. Falls erforderlich, kann der Designer die Vorlage *zurück* in seinen Editor laden und weitere Änderungen an ihrer Struktur und Erscheinung vornehmen. Durch Ergreifen vernünftiger Maßnahmen, um die vom Programmierer vorgenommenen Änderungen zu erhalten, zerstört der Designer die Anwendung nicht.

Seitenvorlagen verfolgen dieses Ziel durch das Anwenden von drei Prinzipien:

1. Gutes Zusammenspiel mit Bearbeitungswerkzeugen.
2. Was man sieht, ist dem, was man bekommt, sehr ähnlich.
3. Code aus Vorlagen heraus halten, außer struktureller Logik.

Eine Seitenvorlage ist wie ein Modell der Seiten, die sie generiert. Vor allem ist es eine gültige HTML-Seite.

Zope-Seitenvorlagen gegenüber DTML

Zope hat schon DTML. Warum braucht man eine andere Vorlagensprache? Zuallererst zielt DTML nicht auf HTML-Designer ab. Sobald eine Seite in eine Vorlage umgewandelt worden ist, ist es ungültiges HTML, was es schwierig macht, damit außerhalb der Anwendung zu arbeiten. Zweitens leidet DTML an dem fehlenden Trennung zwischen Darstellung, Logik und Inhalt (Daten). Dies vermindert die Skalierbarkeit von Bemühungen im Bereich Content-Management- und Website-Entwicklung, die diese Systeme verwenden.

DTML kann Dinge, die Seitenvorlagen nicht können, wie das dynamische Erzeugen von E-Mail-Nachrichten (Seitenvorlagen können nur HTML und XML generieren). Also wird DTML nicht aussterben. Jedoch sehen wir, dass Seitenvorlagen fast alle den gesamten Bereich der HTML-/XML-Darstellung in Zope übernimmt.

Wie Seitenvorlagen funktionieren

Seitenvorlagen verwenden die Template Attribute Language (TAL). TAL besteht aus speziellen Tag-Attributen. So ähnlich könnte zum Beispiel ein dynamischer Seitentitel aussehen:

```
<title tal:content="here/title">Seitentitel</title>
```

Das Attribut `tal:content` ist eine TAL-Anweisung. Da es einen XML-Namensraum besitzt (der `tal:`-Teil) werden sich die meisten Bearbeitungswerkzeuge nicht darüber beschweren, dass sie es nicht verstehen und werden es auch nicht entfernen. Die Struktur oder Erscheinung der Vorlage wird nicht verändert, wenn Sie sie in einen WYSIWYG-Editor oder einen Web-Browser laden. Der Name `content` zeigt an, dass es den Inhalt des `title`-Tags setzt und der Wert "here/title" ist ein Ausdruck, der den Text liefert, der in das Tag eingefügt wird.

Alle TAL-Anweisungen bestehen aus Tag-Attributen deren Name mit `tal:` beginnt und alle TAL-Anweisungen besitzen mit ihnen verknüpfte Werte. Der Wert einer TAL-Anweisung befindet sich innerhalb der Anführungszeichen. Siehe Anhang C, "Zope-Page-Templates-Referenz" für weitere Informationen über die TAL.

Für den HTML-Designer, der ein WYSIWYG-Werkzeug verwendet, ist das dynamische Titel Beispiel völlig gültiges HTML und wird im Editor so dargestellt, wie ein Titel aussehen sollte. Mit anderen Worten: Seitenvorlagen arbeiten gut mit Bearbeitungswerkzeugen zusammen.

Dieses Beispiel demonstriert auch das Prinzip "Was man sieht, ist dem, was man bekommt, sehr ähnlich". Wenn Sie die Vorlage in einem Editor betrachten, wirkt der Titel-Text als Platzhalter für den dynamischen Titeltext. Die Vorlage bietet ein Beispiel dafür, wie generierte Dokumente aussehen.

Wenn diese Vorlage in Zope gespeichert und von einem Benutzer betrachtet wird, wandelt Zope den Beispiel-Inhalt in dynamischen Inhalt und ersetzt "Seitentitel" durch das, was auch immer "here/title" liefert. In diesem Fall liefert "here/title" den Titel des Objekts auf das die Vorlage angewendet wird. Diese Ersetzung wird dynamisch durchgeführt, wenn die Vorlage betrachtet wird.

Es gibt Vorlagenanweisungen für das Ersetzen von ganzen Tags, deren Inhalten oder lediglich einiger ihrer Attribute. Sie können ein Tag mehrmals wiederholen oder es ganz weglassen. Sie können Teile mehrerer Vorlagen kombinieren und eine einfache Fehlerbehandlung festlegen. All diese Fähigkeiten werden verwendet, um Dokumentstrukturen zu generieren. Trotz dieser Fähigkeiten können Sie keine Unterroutinen oder Klassen erstellen, komplexe Ablaufsteuerungen durchführen oder auf einfache Weise komplexe Algorithmen ausdrücken. Für diese Aufgaben sollten Sie python-basierte Skripte oder Anwendungskomponenten verwenden.

Die Seitenvorlagensprache ist absichtlich nicht so mächtig und allgemein verwendbar, wie sie sein konnte. Sie ist dazu bestimmt, innerhalb eines Rahmenwerks (wie etwa Zope) verwendet zu werden, bei dem andere Objekte die Geschäftslogik verarbeiten und für Aufgaben verwendet werden, die nicht mit dem Seitenlayout zusammenhängen.

Die Vorlagensprache wäre zum Beispiel nützlich, um eine Rechnungsseite aufzubauen, indem je eine Zeile pro Einzelposten erzeugt wird und Beschreibung, Menge, Preis usw. in den Text jeder Zeile eingefügt wird. Sie würde dazu verwendet werden, um den Rechnungseintrag in einer Datenbank zu erzeugen oder mit einer Einrichtung zur Kreditkartenverarbeitung zu interagieren.

Seitenvorlagen erstellen

Beim Entwerfen von Seiten verwendet man wahrscheinlich FTP oder WebDAV statt dem Zope-Managment-Interface (ZMI) um Seitenvorlagen zu erstellen und zu bearbeiten. Informationen zum Bearbeiten von Seitenvorlagen über das Netz erhalten Sie später in diesem Kapitel, im Abschnitt "FTP und WebDAV verwenden". Für die kleinen Beispiele in diesem Kapitel ist einfacher das ZMI zu benutzen.

Verwenden Sie Ihren Web-Browser, um sich als Manager im Zope-Managment-Interface anzumelden. Suchen Sie sich einen Ordner in dem Sie arbeiten möchten (Der Wurzelordner

root ist in Ordnung) und wählen Sie "Page Template" (Seitenvorlage) aus der Produktauswahlliste. Geben Sie "einfache_seite" in das Feld *Id* des Erstellformulars ein und klicken Sie dann auf die Schaltfläche "Add and Edit" ("Hinzufügen und Bearbeiten").

Sie sollten jetzt die Hauptbearbeitungsseite der neuen Seitenvorlage sehen. Der Titel (Title) ist leer, der Inhaltstyp (Content-Type) ist `text/html` und im bearbeitungsfeld befindet sich der Standardvorlagentext.

Lassen Sie uns nun eine einfache dynamische Seiten erstellen! Geben Sie den Text "eine einfache Seite" in das Feld *Title* ein. Ändern Sie den Vorlagetext danach so ab, dass er folgendermaßen aussieht:

```
<html>
 <body>
   <p>
     Dies ist <b tal:replace="template/title">der Titel</b>.
   </p>
 </body>
</html>
```

Drücken Sie jetzt die Schaltfläche *Save Changes* (Änderungen speichern). Zope sollte eine Mitteilung anzeigen die bestätigt, dass Ihre Änderungen gespeichert wurden.

Falls ein HTML-Kommentar in den Vorlagentext eingefügt wird, der mit `<-- Page Template Diagnostics` beginnt, überprüfen Sie, ob Sie das Beispiel korrekt abgetippt haben und speichern Sie es erneut. Dieser Kommentar ist eine Fehlermeldung, die einem mitteilt, dass etwas fehlerhaft ist. Sie müssen den Fehlerkommentar nicht löschen. Sobald der Fehler korrigiert wurde verschwindet er.

Klicken Sie auf den Reiter *Test*. Sie sollten eine Seite sehen, auf der oben "Dies ist eine einfache Seite" steht. Beachten Sie, dass der Text noch nicht formatiert ist. Nichts ist fettgedruckt.

Gehen Sie zurück und klicken Sie auf den Verweis *Browse HTML source* (HTML-Quelltext betrachten) unter dem Feld Content-Type. Daraufhin wird Ihnen der *uninterpretierte* Quelltext der Vorlage angezeigt. Sie sollten den Text "Dies ist **der Titel**" sehen. Gehen Sie wieder zurück, sodass Sie das Beispiel weiter bearbeiten können.

Das Feld *Content-Type* erlaubt Ihnen den Inhaltstyp Ihrer Seite festzulegen. Normalerweise werden Sie den Inhaltstyp `text/html` für HTML oder den Typ `text/xml` für XML verwenden.

Wenn man den Inhaltstyp auf `text/html` setzt, analysiert Zope Ihre Vorlage im HTML-Kompatibilitätsmodus, der die Verwendung der toleranten HTML-Syntax erlaubt. Wenn man den Inhaltstyp auf etwas anderes als `text/html` setzt, nimmt Zope an, dass sich bei der Vorlage um wohlgeformtes XML handelt. Zope benötigt auch eine explizite Deklaration der XML-Namensräume TAL und METAL für wohlgeformtes XML.

Die Funktion *Expand macros with editing* ist in Kapitel 9 "Seitenvorlagen für Fortgeschrittene" beschrieben.

Einfache Ausdrücke

Der Ausdruck "template/title" in Ihrer einfachen Seitenvorlage ist ein *Pfadausdruck*. Dies ist der gängigste Ausdruckstyp. Es gibt mehrere andere Arten von Ausdrücken die durch den Standard "TAL Expression Syntax" (TALES) definiert werden. Für weitere Informationen über TALES sehen Sie in Anhang C, "Zope Seitenvorlagen verweisen" nach.

Der Pfadausdruck "template/title" holt die Eigenschaft `title` der Vorlage. Hier sind einige andere gebräuchliche Pfadausdrücke:

- `request/URL`: Der URL der gegenwärtigen Web-Anfrage.
- `user/getUserName`: Der Login-Name des angemeldeten Benutzers.
- `container/objectIds`: Eine Liste von Ids der Objekte die sich in demselben Ordner wie die Vorlage befinden..

Jeder Pfad beginnt mit einem Variablennamen. Wenn die Variable den gewünschten Wert enthält, belassen Sie es hierbei. Ansonsten fügen Sie einen Schrägstrich ('/') hinzu und den Namen eines Unterobjekts oder einer Eigenschaft. Es kann sein, dass Sie sich Ihren Weg durch mehrere Unterobjekte bahnen müssen, um zu dem Wert zu kommen, den Sie suchen.

Zope definiert einen kleinen Satz an eingebauten Variablen wie `request` und `user`, die in Kapitel 9, "Seitenvorlagen für Fortgeschrittene" beschrieben werden. In diesem Kapitel erfahren Sie auch, wie Sie Ihre eigenen Variablen definieren können.

Text einfügen

In Ihrer Vorlage "einfache_seite" verwendeten Sie die Anweisung `tal:replace` in einem bold-Tag. Während Sie sie testeten, ersetzte Zope das ganze Tag durch den Titel der Vorlage. Als Sie den Quelltext überflogen, haben Sie den Vorlagentext fettgedruckt gesehen. Wir verwendeten ein bold-Tag, um den Unterschied hervorzuheben.

Um dynamischen Text in einen anderen Text zu setzen, verwenden Sie normalerweise `tal:replace` in einem span-Tag anstatt in einem bold-Tag. Fügen Sie Ihrem Beispiel beispielsweise die folgenden Zeilen hinzu:

```
<br>
Der URL ist <span tal:replace="request/URL">URL</span>.
```

Das Tag `span` ist strukturell, nicht visuell, so dass dies aussieht, wie "Der URL ist URL." wenn Sie den Quelltext in einem Editor oder Browser betrachten. Wenn Sie die dargestellte Version betrachten, kann sie in etwa so aussehen:

```
<br>
Der URL ist http://localhost:8080/einfache_seite.
```

Wenn Sie Text in ein Tag einfügen, aber das Tag selbst unbeeinflusst lassen wollen, verwenden Sie die Anweisung `tal:content`. Um den Titel Ihrer Beispielseite auf die title-Eigenschaft der Vorlage zu setzen, fügen Sie die folgenden Zeilen zwischen das html- und die body-Tag ein:

```
<head>
  <title tal:content="template/title">Der Titel</title>
</head>
```

Wenn Sie den Reiter "Test" in einem neuen Browser-Fenster öffnen, wird der Titel des Fensters "eine einfache Seite" sein. Wenn Sie den Quelltext der Seite betrachten, sehen Sie, in etwa so etwas:

```
<html>
  <head>
    <title>eine einfache Seite</title>
  </head>
...
```

Zope fügte den Titel Ihrer Vorlage ins `title`-Tag ein.

Sich wiederholende Strukturen

Lassen Sie uns jetzt Ihrer Seite etwas Kontext in Form einer Liste der Objekte geben, die sich im selben Ordner wie die Vorlage befinden. Sie werden eine Tabelle erstellen, die eine nummerierte Zeile für jedes Objekt und Spalten für die Id, den Metatyp und den Titel besitzt. Fügen Sie dem unteren Teil Ihrer Beispielvorlage diese Zeilen hinzu:

```
<table border="1" width="100%">
  <tr>
    <th>Nummer</th>
    <th>Id</th>
    <th>Metatyp</th>
    <th>Titel</th>
  </tr>
  <tr tal:repeat="item container/objectValues">
    <td tal:content="repeat/item/number">Nr.</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Metatyp</td>
    <td tal:content="item/title">Titel</td>
  </tr>
</table>
```

Die Anweisung `tal:repeat` in der Tabellenzeile bedeutet "diese Zeile für jedes Element in der Liste von Objektwerten meines Behälters wiederholen". Die repeat-Anweisung legt die Objekte der Liste nacheinander in die Variable item (diese wird *Wiederholungsvariable* genannt) und erstellt eine Kopie der Zeile mit dieser Variable. Der Wert von "item/getId" ist in jeder Zeile die Id des Objektes für diese Zeile und ebenso ist es mit "item/meta_type" und "item/title".

Sie können jeden beleiebigen Namen für die Wiederholungsvariable verwenden ("item" ist nur ein Beispiel), solange er mit einem Buchstaben beginnt und nur Buchstaben, Ziffern und Unterstriche ('_') enthält. Die Wiederholungsvariable wird nur im Wiederholungs-Tag definiert. Wenn Sie versuchen, sie über- oder unterhalb des Tags `tr` zu verwenden, erhalten Sie einen Fehler.

Sie können den Namen der Wiederholungsvariable auch verwenden, um Informationen über die gegenwärtige Wiederholung zu bekommen. Indem Sie ihn nach der standardmäßig festegelegten Variablen `repeat` in einen Pfad setzen, können Sie auf den Wiederholungszähler von Null beginnend ('index'), von eins beginnend ('number'), von "A" ab ('Letter') und auf mehrere andere Art zugreifen. So ist der Ausdruck `repeat/item/number` in der ersten Reihe `1`, in der zweiten Reihe `2` usw.

Da eine `tal:repeat`-Schleife auch in eine andere gesetzt werden kann, kann mehr als eine zur selben Zeit aktiv sein. Dies ist der Grund dafür, dass Sie `repeat/item/number` statt nur `repeat/number` schreiben müssen. Sie müssen angeben, welche Schleife Sie interessiert, indem Sie den Schleifennamen mit angeben.

Betrachten Sie jetzt die Seite und achten Sie darauf, wie sie all die Objekte im selben Ordner wie die Vorlage auflistet. Versuchen S ie, dem Ordner Objekte hinzuzufügen oder zu löschen, und achten Sie darauf, wie die Seite diese Änderungen wiedergibt.

Bedingte Elemente

Durch das Verwenden von Seitenvorlagen können Sie Ihre Umgebung dynamisch abfragen und selektiv Text je nach bestimmten Bedingungen einfügen. Zum Beispiel könnten Sie spezielle Informationen als Antwort auf einen Cookie anzeigen:

```
<p tal:condition="request/cookies/verbose | nothing">
  Hier ist die von Ihnen gewünschte Zusatzinformation.
</p>
```

Dieser Absatz wird nur in die Ausgabe miteinbezogen, wenn einen Cookie namens `verbose` gibt. Der Ausdruck 'request/cookies/verbose | nothing' ist nur wahr, wenn es ein Cookie mit dem Namen `verbose` gesetzt ist. Sie erfahren mehr über diese Art von Ausdruck in Kapitel 9, "Seitenvorlagen für Fortgeschrittene".

Mit Hilfe der Anweisung `tal:condition`, können Sie allelei Bedingungen überprüfen. Eine `tal:condition`-Anweisung tut nichts, wenn ihr Ausdruck einen wahren Wert hat, entfernt jedoch das komplette Ausdrucks-Tag, einschließlich dessen Inhalt, wenn der Wert falsch ist. Zope erachtet die Zahl Null, eine leere Zeichenkette, eine leere Liste und die eingebaute Variable `nothing` als "falsche" Werte. Fast jeder andere Wert ist wahr, einschließlich Zahlen ungleich Null, Zeichenketten mit irgendeinem Inhalt (sogar Leerzeichen!).

Eine andere häufige Anwendung von Bedingungen ist es, eine Folge testen, um zu sehen, ob sie leer ist, bevor Sie über sie iterieren. Zum Beispiel haben Sie im letzten Abschnitt gesehen, wie man eine Tabelle erstellt, indem man über eine Sammlung von Objekten iteriert. Hier sehen Sie, wie man der Seite man eine Überprüfung hinzufügen kann, so dass, wenn die Liste von Objekten leer ist, keine Tabelle gezeichnet wird:

```
<table tal:condition="container/objectValues"
       border="1" width="100%">
  <tr>
    <th>Nummer</th>
    <th>Id</th>
    <th>Metatyp</th>
    <th>Titel</th>
  </tr>
  <tr tal:repeat="item container/objectValues">
    <td tal:content="repeat/item/number">Nr.</td>
    <td tal:content="item/getId">Id</td>
    <td tal:content="item/meta_type">Metatyp</td>
    <td tal:content="item/title">Titel</td>
  </tr>
</table>
```

Wenn der Ausdruck `container/objectValues` falsch ist, wird die komplette Tabelle weggelassen.

Attribute ändern

Die meisten, wenn nicht alle, der von Ihrer Vorlage aufgelisteten Objekte haben eine `icon`-Eigenschaft, die den Pfad zum Symbol für diese Art von Objekt enthält. Um dieses Symbol in der Spalte Metatyp zu zeigen, müssen Sie diesen Pfad ins Attribut `src` eines `img`-Tags einfügen. Bearbeiten Sie die Spalte Metatyp in beiden Zeilen so, dass diese folgendermaßen aussehen:

```
<td><img src="/misc_/OFSP/Folder_icon.gif"
         tal:attributes="src item/icon">
    <span tal:replace="item/meta_type">Metatyp</span>
</td>
```

Die Anweisung `tal:attributes` ersetzt das Attribut `src` des `img`-Tags durch den Wert von `item/icon`. Das Attribut `src="/misc_/OFSP/Folder_icon.gif"` in der Vorlage fungiert als Platzhalter.

Achten Sie darauf, dass wir das Attribut `tal:content` in der Tabelle mit einer tal:replace-Anweisung in einem `span`-Tag ersetzt haben. Diese Änderung erlaubt Ihnen, sowohl eine Abbildung als auch Text in der Tabellenzelle zu haben.

Ein Dateiarchiv mit Seitenvorlagen erstellen

Hier ist ein Beispiel dafür, wie man Seitenvorlagen mit Zope verwendet, um mit einer Vorlage, etwas Pythoncode und einigen Dateien ein einfaches Dateiarchiv zu erstellen.

Erstellen Sie zuerst einen Entwurf der Dateiarchivseite mit Hilfe Ihres HTML-Editors. Die Beispiele in diesem Kapitel wurden mit Amaya erzeugt. Sie müssen es mit dem Entwurf nicht übertreiben, er zeigt lediglich einige fiktive Angaben. Hier ist ein Entwurf eines Dateiarchivs, die eine Datei enthält:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Dateiarchiv</title>
  <style type="text/css">
  .header {
    font-weight: bold;
    font-family: helvetica;
    background: #DDDDDD;
  }
  h1 {
    font-family: helvetica;
  }
  .filename {
    font-family: courier
  }
  </style>
  <meta name="GENERATOR" content="amaya 5.1">
</head>

<body>
<h1>Dateiarchiv</h1>

<p>Klicken Sie auf eine der untenstehenden Dateien um sie
herunterzuladen.</p>

<table border="1" cellpadding="5" cellspacing="0">
  <tbody>
    <tr>
      <td class="header">Name</td>
      <td class="header">Typ</td>
      <td class="header">Größe</td>
      <td class="header">Zuletzt geändert</td>
    </tr>
    <tr>
      <td><a href="Beispiel.tgz"
class="dateiname">Beispiel.tgz</a></td>
      <td>application/x-gzip-compressed</td>
      <td>22 K</td>
      <td>2001/09/17</td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

Melden Sie sich jetzt an Ihrem Zope an und erzeugen Sie einen Ordner namens `Dateiarchiv`. Erstellen Sie in diesem Ordner eine `index_html` genannte Seitenvorlage, indem Sie `Page Template` in der Produktauswahlliste wählen, als Id `index_html` im Formular angeben und auf *Add* (Hinzufügen) klicken.

Speichern Sie jetzt mit Ihrem HTML-Editor den obigen HTML-Code in den URL der Seitenvorlage `index_html`, gefolgt von `/source.html`, zum Beispiel `http://localhost:8080/DateiArchiv/index_html/source.html`. Achten Sie darauf, dass der URL zum *Speichern* der Seite `index_html` mit `source.html` endet. Da Seitenvorlagen dynamisch sind, benötigen Sie eine Möglichkeit, den Quelltext der Vorlage zu

bearbeiten, der noch nicht durch die Seitenvorlagensprache interpretiert wurde. Das Anhängen von `source.html` an eine Seitenvorlage, gibt Ihnen diesen Quelltext. Beachten Sie, dass, wenn der Inhaltstyp Ihrer Seite `text/xml` ist, Sie `source.xml` statt `source.html` verwenden müssen.

Nun, da Sie die Vorlage gespeichert haben, können Sie zu Zope zurückgehen und auf `index_html` klicken und dann auf dessen Reiter *Test* klicken, um die Vorlage zu betrachten. Sie sieht genauso aus wie der Entwurf, also läuft alles gut.

Lassen Sie uns die oben genannte HTML-Seite etwas aufbohren und ihr ein bisschen dynamische Zauberei beibringen. Zuerst wollen wir, dass der Titel der Vorlage dynamisch ist. In Zope werden Sie bemerken, dass die Seitenvorlage ein Formularfeld namens *title* hat, das Sie ausfüllen können. Statt statischem HTML, wollen wir, dass Zope den Seitenvorlagentitel dynamisch in die wiedergegebene Fassung der Vorlage einfügt. Hier sehen Sie wie:

```
<head>
  ...
  <title tal:content="template/title">Dateiarchiv</title>
  ...

<body>
<h1 tal:content="template/title">Dateiarchiv</h1>
...
```

Gehen Sie jetzt zu Zope und ändern Sie den Titel der Seitenvorlage `index_html` ändern. Klicken Sie, nach dem Speichern dieser Änderung, auf den Reiter *Test*. Wie Sie sehen können, hat die Seitenvorlage den Titel des Vorlagenobjekts dynamisch in die Ausgabe der Vorlage eingefügt.

Beachten Sie das neue Tag-Attribut `content` im XML-Namensraum `tal`. Dieses Attribut bedeutet "den *Inhalt* (content) dieses Tags durch die Variable 'template/title' ersetzen." In diesem Fall ist `template/title` der Titel der Seitenvorlage `index_html`.

Das nächste Stück Zauberei soll eine dynamische Dateiliste aufbauen, die Ihnen alle Dateiobjekte im Ordner `Dateiarchiv` zeigt.

Um anfangen zu können, müssen Sie nur eine Zeile Python schreiben. Gehen Sie zum Ordner `Dateiarchiv` und erstellen Sie ein `Script (Python)` in diesem Ordner. Geben Sie dem Skript die Id `dateien` und klicken Sie auf *Add and Edit* (Hinzufügen und Bearbeiten). Ändern Sie das Skript so ab,, dass es den folgenden Pythoncode enthält:

```
## Script (Python) "dateien"
##
return container.objectValues('File')
```

Dies gibt eine Liste von Datei (File)-Objekten im Ordner Dateiarchiv zurück. Bearbeiten Sie Jetzt Ihre Seitenvorlage `index_html` und fügen Sie Ihrem Entwurf noch einige `tal`-Attribute hinzu:

```
      ...
      <tr tal:repeat="item container/dateien">
        <td><a href="Beispiel.tgz" class="dateiname"
               tal:attributes="href item/getId"
               tal:content="item/getId">Beispiel.tgz</a></td>
        <td tal:content="item/content_type">application/x-gzip-
compressed</td>
        <td tal:content="item/getSize">22 K</td>
        <td tal:content="item/bobobase_modification_time">2001/09/17</td>
      </tr>
      ...
```

Der interessante Teil daran ist das Attribut `tal:repeat` im HTML-Tag `tr`. Dieses Attribut veranlasst die Vorlage dazu, über die Werte zu laufen (zu iterieren), die von "container/dateien" (das Pythonskript das Sie erstellt haben) zurückgegeben wurden, und eine neue Tabellenzeile für jede dieser Dateien zu erstellen. Während jeder Wiederholung wird dem gegenwärtigen Dateiobjekt über das gerade iteriert wird, die Name `item` zugeteilt.

Die Zellen jeder Zeile haben alle `tal:content`-Attribute, die die Daten beschreiben, die in jede Zelle kommen sollen. Während jeder Wiederholung der Tabellenzeilenschleife ersetzen die Id, der Inhaltstyp, die Größe und das Änderungsdatum die fiktiven Daten in den Zeilen. Beachten Sie auch, wie der Hypertext-Verweis dynamisch auf die gegenwärtige Datei zeigt, indem mit Hilfe von `tal:attributes` das Attribut `href` überschrieben wird.

Diese Daten stammen vom Objekt `item`, durch das Aufrufen von Zope-API-Methoden darauf, weil wir wissen, dass es ein File-Objekt ist. Die Methoden `item/getId`, `item/content_type`, `item/getSize`, `item/bobobase_modification_time` sind alle Standard-API-Funktionen, die in Zopes Online-Hilfesystem dokumentiert sind.

Gehen Sie zu Zope und testen sie dieses Skript, indem Sie zuerst einige Dateien in den Ordner `Dateiarchiv` hochladen. Dies geschieht durch Auswählen von `File` in der Produktauswahlliste und das Klicken auf die Schaltfläche `upload` (hochladen) auf der nächsten Seite. Nach dem Hochladen Ihrer Datei können Sie einfach auf *Add* (hinzufügen) klicken. Wenn Sie keine Id angeben, dann wird der Dateiname der Datei verwendet, die Sie hochladen.

Nach dem Sie einige Dateien hochgeladen haben, gehen Sie zur Seitenvorlage `index_html` und klicken Sie auf den Reiter *Test*. Jetzt können Sie sehen, dass die Seitenvorlage ein sehr einfaches Dateiarchiv dargestellt hat, mit nur wenigen Änderungen an HTML-Tag-Attributen.

Es gibt einige kosmetische Probleme an dem Dateiarchiv, so wie es jetzt ist. Die Darstellungen von Größe und Datum sind nicht sehr hübsch und passen nicht zum Format des fiktiven Inhalts. Sie möchten, dass die Größe der Dateien in K oder MB statt Bytes angezeigt wird. Hier ist eine python-basiertes Skript, das Sie dafür verwenden können:

```
      ## Script (Python) "datei_groesse"
      ##
      """
      Eine Zeichenkette zurueckgeben, die die Groesse einer Datei
beschreibt.
      """
```

```
bytes=context.getSize()
k=bytes/1024.0
mb=bytes/1048576.0
if mb > 1:
    return "%.2f MB" % mb
if k > 1:
    return "%d K" % k
return "%d Bytes" % bytes
```

Erstellen Sie dieses Skript mit der Id `datei_groesse` in Ihrem Ordner `Dateiarchiv`. Es berechnet die Größe einer Datei in Kilobytes und Megabytes und gibt eine entsprechende Zeichenkette zurück, die die Größe der Datei beschreibt. Jetzt können Sie das Skript statt des Ausdrucks `item/getSize` verwenden:

```
...
<td tal:content="item/datei_groesse">22 K</td>
...
```

Sie können auch die Datumsformatierungsprobleme mit ein bisschen Python lösen. Erstellen Sie ein Skript mit dem Namen `datei_datum` in Ihrem Ordner `Dateiarchiv`:

```
## Script (Python) "datei_datum"
##
"""
Das Aenderungsdatum als Zeichenkette der Form YYYY/MM/DD zurueckgeben
"""
datum=context.bobobase_modification_time()
return "%s/%s/%s" % (datum.year(), datum.mm(), datum.day())
```

Ersetzen Sie jetzt den Ausdruck `item/bobobase_modification_time` durch einen Verweis auf dieses Skript:

```
...
<td tal:content="item/datei_datum">2001/09/17</td>
...
```

Herzlichen Glückwunsch, Sie haben erfolgreich einen Entwurf genommen und ihn zu einer dynamischen Seitenvorlage gemacht. Dieses Beispiel verdeutlicht, wie gut Seitenvorlagen als "Darstellungsschicht" in Ihren Anwendungen funktionieren. Die Seitenvorlagen stellen die Anwendungslogik (die python-basierten Skripte) dar und die Anwendungslogik arbeitet mit den Daten in Ihrer Site (den Dateien).

Bearbeiten entfernter Dateien mit FTP und WebDAV

Sie können Seitenvorlagen über das Netz sowohl mit FTP und WebDAV bearbeiten, als auch über das Veröffentlichen per HTTP PUT. Mit Hilfe dieser Methoden können Sie Seitenvorlagen verwenden, ohne fortgeschrittene WYSIWYG-Editoren wie DreamWeaver verlassen zu müssen.

Der vorherige Abschnitt hat Ihnen gezeigt, wie man eine Seite bearbeiten und Amaya über das Netz verwenden kann, das HTTP PUT verwendet, um Seiten hochzuladen. Sie können dasselbe mit FTP und WebDAV mit Hilfe derselben Schritte machen.

1. Erstellen Sie eine Seitenvorlage im Zope-Management-Interface. Sie können ihr eine beliebige Dateinamenserweiterung geben. Viele Leute bevorzugen `.html`, während andere `.zpt` bevorzugen. Beachten Sie, dass einige Namen wie `index_html` spezielle Bedeutungen zu Zope haben.
2. Fragen Sie die Datei mit Hilfe des URL Ihrer Seitenvorlage plus `/source.html` oder `/source.xml` ab. Dies liefert Ihnen den Quelltext Ihrer Seitenvorlage.
3. Bearbeiten Sie Ihre Datei mit Ihrem Editor und speichern Sie sie. Wenn Sie sie speichern, sollten Sie denselben Quellen-URL verwenden, den Sie verwendet haben, um sie abzufragen.
4. Laden Sie Ihre Seite wahlweise neu, nachdem Sie sie bearbeitet haben, um sie auf Fehlerkommentare zu überprüfen. Sehen Sie im nächsten Abschnitt für weitere Details über Fehlerbereinigung nach.

In späteren Versionen von Zope werden Sie wahrscheinlich in der Lage sein, Seitenvorlagen ohne das Zope-Management-Interface erstellen zu können.

Fehler bereinigen und testen

Zope hilft Ihnen Fehler in Ihren Seitenvorlagen aufzuspüren und zu korrigieren. Zope bemerkt Probleme zu zwei verschiedenen Zeitpunkten: wenn Sie eine Seitenvorlage bearbeiten und wenn Sie eine Seitenvorlage betrachten. Wenn Sie bearbeiten fängt Zope andere Arten von Problemen ab, als wenn Sie eine Seitenvorlage betrachten..

Sie sind wahrscheinlich schon mit Fehlerkommentaren vertraut, die Zope in Ihre Seitenvorlagen einfügt, wenn es auf Probleme trifft,. Diese Kommentare zeigen Ihnen Probleme an, die Zope findet, während Sie Ihre Vorlagen bearbeiten. Die Art von Problemen, die Zope findet, wenn Sie bearbeiten, sind hauptsächlich Fehler in Ihren TAL-Anweisungen. Zum Beispiel:

```
      <!-- Page Template Diagnostics
       Compilation failed
       TAL.TALDefs.TALError: bad TAL attribute: 'contents', at line 10,
column 1
      -->
```

Diese Diagnosemeldung teilt Ihnen mit, dass Sie versehentlich `tal:contents an`statt `tal:content` in Zeile 10 Ihrer Vorlage verwendet haben. Andere Diagnosemeldungen beschreiben Ihnen Probleme mit Ihren Vorlagenausdrücken und Makros.

Wenn Sie das Zope-Management-Interface verwenden, um Seitenvorlagen zu bearbeiten, ist es leicht, diese Diagnosemeldungen zu entdecken. Wenn Sie jedoch WebDAV oder FTP verwenden, kann man diese Nachrichten leicht übersehen. Falls Sie zum Beispiel eine Vorlage mit FTP auf Zope sichern, erhalten Sie keinen FTP-Fehler, der Ihnen das Problem beschreibt. Vielmehr müssen Sie die Vorlage von Zope neu laden, um die Diagnosemeldung

zu sehen. Wenn man FTP und WebDAV verwendet, ist es gut, Vorlagen neu zu laden, nachdem Sie sie bearbeitet haben, um sich zu vergewissern, dass sie keine Diagnosemeldungen enthalten.

Wenn Sie die Diagnosemeldung nicht bemerken und versuchen, eine Vorlage mit Problemen darzustellen, sehen Sie eine Nachricht wie diese:

```
Error Type: RuntimeError
Error Value: Page Template hallo.html has errors.
```

Das ist Ihr Signal, die Vorlage neu zu laden und die Diagnosemeldung zu überprüfen.

Zusätzlich zu Diagnosemeldungen beim Bearbeiten bekommen Sie gelegentlich normale Zope-Fehler, wenn Sie eine Seitenvorlage betrachten. Diese Probleme werden normalerweise durch Probleme in Ihren Vorlagenausdrücken verursacht. Zum Beispiel könnten Sie einen Fehler bekommen, wenn ein Ausdruck eine Variable nicht ausfindig machen kann:

```
Error Type: Undefined
Error Value: "einhorn" not found in "here/einhorn"
```

Diese Fehlermeldung teilt Ihnen mit, dass die Variable `einhorn` nicht gefunden werden kann, auf die im Ausdruck `here/einhorn` verwiesen wird. Um Ihnen zu helfen, herauszufinden, was falsch lief, bezieht Zope Angaben über die Umgebung ins Traceback ein. Wenn Sie sich im Debugging-Modus befinden, erhalten Sie diese Angaben unten auf der Fehlerseite. Sehen Sie sich ansonsten den Quelltext der Fehlerseite an, um das Traceback zu sehen. Das Traceback beinhaltet Angaben über die Umgebung:

```
...
'here': <Application instance at 01736F78>,
'modules': <Products.PageTemplates.ZRPythonExpr._SecureModuleImporter
instance at 016E77FC>,
'nothing': None,
'options': {'args': ()},
'request': ...
'root': <Application instance at 01736F78>,
'template': <ZopePageTemplate instance at 01732978>,
'traverse_subpath': [],
'user': amos})
...
```

Diese Angaben sind ein wenig kryptisch, aber mit etwas Detektivarbeit können sie Ihnen helfen, zu begreifen, was falsch lief. In diesem Fall sagen sie uns, dass die Variable `here` eine "Application instance" (Anwendungsinstanz) ist. Dies bedeutet, dass es der Wurzelordner von Zope ist, (beachten Sie, dass die Variable `root` dieselbe "Application instance" ist). Vielleicht ist das Problem, dass Sie die Vorlage auf einen Ordner anwenden wollten, der eine Eigenschaft namens `einhorn` besitzt. Das Traceback liefert nicht viel Hilfe, aber es kann Ihnen manchmal weiterhelfen.

XML-Vorlagen

Ein anderes Beispiel für die Flexibilität von Seitenvorlagen ist, dass sie sowohl XML als auch HTML dynamisch wiedergeben können. Zum Beispiel haben Sie in Kapitel 5, "Erstellung einfacher Zope-Anwendungen" den folgenden XML-Code erzeugt:

```
<gaestebuch>
  <eintrag>
    <kommentar>Mein Kommentar</kommentar>
  </eintrag>
  <eintrag>
    <kommentar>Ich mag deine Webseite</kommentar>
  </eintrag>
  <eintrag>
    <kommentar>Bitte keine Blink-Tags</kommentar>
  </eintrag>
</gaestebuch>
```

Dieser XML-Code wurde durch Iterieren über alle DTML-Dokumente in einem Ordner und das Einfügen ihres Quelltexts in `kommentar`-Elemente geschaffen. In diesem Abschnitt zeigen wir Ihnen, wie man Seitenvorlagen dazu verwendet, um den gleichen XML-Code zu erzeugen.

Erstellen Sie eine neue Seitenvorlage namens "eintraege.xml" in Ihrem Gästebuch-Ordner mit dem folgenden Inhalt:

```
<gaestebuch xmlns:tal="http://xml.zope.org/namespaces/tal">
  <eintrag tal:repeat="eintrag python:here.objectValues('DTML
Document')">
    <kommentar tal:content="eintrag/document_src">Hierher kommt der
Kommentar...</kommentar>
  </eintrag>
</gaestebuch>
```

Vergewissern Sie sich, dass Sie den Inhaltstyp auf `text/xml` stellen. Klicken Sie auf *Save Changes* und den Reiter *Test*. Wenn Sie Netscape verwenden, fordert es Sie dazu auf, ein XML-Dokument herunterzuladen, wenn Sie MSIE 5 oder höher verwenden, können Sie das XML Dokument im Browser betrachten.

Beachten sie wie die Anweisung `tal:repeat` über alle DTML-Dokumente iteriert. Die Anweisung `tal:content` fügt den Quelltext jedes Dokuments in das Element `kommentar` ein. Das Attribut `xmlns:tal` ist eine XML-Namensraum-Anweisung. Es sagt Zope, dass das Namen die mit `tal` beginnen, Seitenvorlagenbefehle sind. Siehe Anhang C, "Zope-Page-Templates-Referenz" für weitere Informationen über die XML-Namensräume TAL- und METAL.

XML mit Seitenvorlagen zu erzeugen, ist fast genauso wie das Erzeugen von HTML. Der wichtigste Unterschied ist, dass Sie explizite XML-Namensraum-Deklarationen verwenden müssen. Ein anderer Unterschied ist, dass Sie die Inhaltstyp auf `text/xml`, oder wie auch immer der Inhaltstyp für Ihr XML sein soll, stellen sollten. Der letzte Unterschied ist, dass Sie

sich den Quelltext einer XML-Vorlage durch Gehen zu `source.xml` statt `source.html` ansehen können.

Vorlagen mit Inhalt verwenden

Im Allgemeinen unterstützt Zope Inhalts-, Darstellungs- und Logikkomponenten. Seitenvorlagen sind Darstellungskomponenten, und sie können verwendet werden, um Inhaltskomponenten anzuzeigen.

Zope 2.5 wird mit mehreren Inhaltsbestandteilen ausgeliefert: ZSQL-Methoden, Files und Images. DTML-Dokumente und -Methoden sind nicht wirklich reine Inhaltsbestandteile, da sie Inhalt beinhalten und DTML-Code ausführen können. Zur Zeit besitzt Zope standardmäßig kein gutes, universal einsetzbares Inhaltsobjekt. Sie können Files für Textinhalt verwenden, da Sie den Inhalt von Files bearbeiten können, wenn die Datei weniger als 64 K groß ist und Text enthält. Jedoch ist das File-Objekt ziemlich spartanisch.

Zopes Content Management Framework (CMF) löst dieses Problem, indem es ein Sortiment von vielfältigen Inhaltskomponenten bietet. Das CMF ist Zopes Content-Management-Erweiterung. Es führt viele Arten von Verbesserungen ein, einschließlich Workflow, Skins und Inhaltsobjekte ein. Das CMF verwendet Seitenvorlagen ausgiebig. Eine spätere Version von Zope wird das CMF wahrscheinlich mit beinhalten.

Zusammenfassung

Zope-Seitenvorlagen helfen Ihnen, Webseiten für Ihre Web-Anwendungen zu erstellen. Vorlagen erleichtern Ihnen, normale HTML-Werkzeuge und -Techniken zu verwenden, um Webseiten zu erstellen. Sie bieten auch bequeme Anpassungsmöglichkeiten, die es Ihnen erlauben, sie in Ihre Anwendungen einzubinden. Seitenvorlagen helfen Gestaltern und Programmierern, zusammen zu arbeiten, um Web-Anwendungen zu produzieren. In Kapitel 9, "Seitenvorlagen für Fortgeschrittene" werden Sie etwas über mächtige Techniken für Vorlagen lernen, wie Python-Ausdrücke und Makros.

Zopebuch: Inhaltsverzeichnis

Dieses Dokument wird momentan übersetzt und ist zur Zeit leider teilweise noch in Englisch vorhanden.

Kapitel 6: Erstellen einfacher Zope-Anwendungen

In Kapitel 3, "Benutzen einfacher Zope-Objekte" und Kapitel 4, "Dynamischer Inhalt mit DTML" lernten Sie grundlegende Zope-Objekte und DTML kennen. In diesem Kapitel werden Sie erfahren, wie Sie einfache, aber leistungsfähige Web-Anwendungen mit Hilfe dieser Mittel. In späteren Kapiteln dieses Buches werden Sie etwas über vielschichtigere Objekte und komplizierteres DTML erfahren. Wie auch immer, die in diesem Kapitel besprochenen Design-Techniken werden auch später angewendet.

Anmerkung: in Kapitel 3, "Einfache Zope-Objekte", erklärten wir, inwiefern "Zope Page Templates" neu für Zope sind und dass sie für Präsentationen benutzt werden sollten. Wir haben dieses Kapitel allerdings noch nicht von DTML auf "Page Templates" umgestellt, werden es aber bald, um die neuen - auf "Page Templates" basierenden - Methodiken darzustellen, überarbeiten.

Erstellen von Anwendungen mit Hilfe von Ordnern

Ordner sind die "grundlegenden Bausteine" jeder Zope-Anwendung. Sie erlauben Ihnen die Organisation Ihrer Zope-Objekte und aktives Teilnehmen in Ihren Web-Anwendungen. Ordner bekommen durch das Hinzufügen von Skripten ein Verhalten.

Skripte und Ordner arbeiten zusammen, um einfache Anwendungen zu realisieren. Ordner stellen die Informations-Struktur und ein Rahmenwerk für das Verhalten Ihrer Site zur Verfügung. Später in diesem Kapitel wird Ihnen ein Beispiel dieses Design-Konzeptes anhand einer einfachen Gästebuch-Anwendung vorgestellt. Ein Ordner wird zum Bereithalten der Methoden, Skripte und Daten der Gästebuch-Anwendung benutzt. Die Skripte stellen das Verhalten zur Verfügung, die Methoden sind für die Darstellung der Anwendung verantwortlich.

Nehmen Sie - als Beispiel - an, Sie haben einen Ordner *Rechnungen* um Rechnungen zu speichern. Sie könnten 2 Objekte *RechnungHinzufuegen* und *RechnungBearbeiten* in diesem Ordner erzeugen, welche Ihnen erlauben, Rechnungen hinzuzufügen beziehungsweise zu bearbeiten. Jetzt wäre Ihr *Rechnungen*-Ordner eine kleine Anwendung.

(A. d. Übersetzers): Diese Objekte könnten Skripte oder Methoden sein, die entsprechende Funktionalität muss von Ihnen implementiert werden. Genaueres dazu finden Sie im Kapitel 4 und 10.

Zope's simple and expressive URLs are used to work with the invoices application.

Wie Sie gesehen haben, können Sie Zope Objekte anzeigen, indem Sie die entsprechende URL in Ihrem Browser aufrufen. So ruft, zum Beispiel, die URL *http://localhost:8080/Rechnungen/RechnungHinzufuegen* das Objekt *RechnungHinzufuegen* im Ordner *Rechnungen* auf. Diese URL könnte Sie zu einer Eingabemaske führen, auf der Sie Rechnungen hinzufügen können. Auf die gleiche Art würde die URL *http://localhost:8080/Rechnungen/RechnungBearbeiten?rechnungs_nummer=42* das Objekt *RechnungBearbeiten* im *Rechnungen*-Ordner aufrufen und ihm das Argument *rechnungs_nummer* mit dem Wert 42 übergeben. Diese URL könnte Sie zu einer Maske führen, in der Sie die Rechnung mit der Nummer 42 editieren können.

Aufrufen von Objekten in Ordnern mit Hilfe von URLs

Das Beispiel Rechnungen veranschaulicht eine nützliches Zope-Eigenschaft. Sie können Objekte in Ordnern aufrufen, indem Sie zu einer URL gehen, die aus der URL des Ordners gefolgt von der ID des Objektes besteht. Dieses Verfahren wird überall in Zope benutzt und ist ein sehr allgemeines "design pattern". Sie sind nicht darauf beschränkt, nur Objekte in Ordnern aufzurufen - Sie werden später sehen, dass Sie die gleiche URL-Technik auf alle Objekte (nicht nur Ordner) anwenden können.

Nehmen wir als Beispiel an, Sie wollen ein Objekt *OrdnerBetrachten* in einem Ihrer Ordner aufrufen. Vielleicht haben Sie viele verschiedene *OrdnerBetrachten*-Objekte an unterschiedlichen Stellen. Zope findet heraus, welches *OrdnerBetrachten* Sie meinen, indem es zuerst in dem Ordner sucht, in dem Sie das Objekt aufgerufen haben. Findet es das Objekt nicht, sucht Zope in dem darüberliegenden Ordner. Ist es dort nicht zu finden, wird in dem darüberliegenden Ordner gesucht. Dieser Prozess wird solange fortgeführt, bis Zope das Objekt findet oder der Wurzelordner *(A. d. Übersetzers: zum Beispiel "http://localhost:8080/")* erreicht ist. Wird das Objekt auch dort nicht gefunden, gibt Zope auf und wirft eine Ausnahme.

Sie werden dieses dynamische Verhalten an vielen verschiedenen Stellen in Zope wiedersehen. Diese Technik wird *acquisition* genannt. Ein Ordner akquiriert ein Objekt, indem er in seinen Kontainern sucht.

Das besondere Objekt *index_html*

Wie Sie gesehen habe, können Sie alle Arten von Objekten akquirieren. Es ein besonderes Objekt, das Zope dazu benutzt, einen Ordner anzuzeigen. Es heisst *index_html*.

Das *index_html*-Objekt stellt eine Standard-Darstellung des Ordners zur Verfügung. Es ist vergleichbar dazu, wie eine *index.html* als Standard-Darstellung für einen Ordner bei Apache- oder anderen Webservern dient.

Wenn Sie ein Objekt *index_html* in Ihrem *Rechnungen*-Ordner erzeugen und dann auf den "View"-Tab klicken oder die URL *http://localhost:8080/Rechnungen/* besuchen, wird Zope das Objekt *index_html* im Ordner *Rechnungen* aufrufen.

Ein Ordner kann ein *index_html*-Objekt genauso von seinem Vater-Ordner akquirieren, wie es jedes beliebige Objekt akquirieren kann. Sie können dieses Verhalten dazu benutzen, vielen (Unter-)Ordnern ein vorgegebenes Aussehen zu geben. Um ein abweichendes Aussehen für einen bestimmten Ordner zu erreichen, erzeugen Sie einfach ein neues *index_html*-Objekt in dem entsprechenden Ordner. Auf diese Weise können Sie das in dem *index_html*-Objekt im Vater-Ordner vordefinierte Aussehen überschreiben.

Erstellung der Zope Zoo Webseite

In diesem Abschnitt werden Sie eine einfache Webseite für den Zope Zoo erstellen. Als Webmaster des Zoos ist es Ihr Job, die Seite benutzbar und wartbar zu machen. Hier ein paar Dinge, die Sie bachten müssen:

- Die Benutzer des Zoos müssen genauso einfach durch den Zoo laufen können, wie in einem echten Zoo.
- Alle gemeinsamen Layout-Tools, wie zum Beispiel Cascading Style Sheets (CSS), müssen an einer einfach zu wartenden Stelle platziert sein.
- Sie müssen eine einfache Datei-Bibliothek von, die Tiere beschreibenden, Dokumenten verfügbar mache.
- Sie benötigen eine site map, so dass Ihre Besucher schnell einen Eindruck vom Aufbau des Zoos bekommen.
- Ein Gästebuch muss erzeugt werden, damit Besucher Ihnen Rückmeldungen und Kommentare über Ihre Seite zukommen lassen können.

- Ein Abschnitt "What's new" muss erstellt werden, der den Besuchern mitteilt, welche Äderungen sich vor Kurzem ereignet haben.

Im Zoo navigieren

Damit die Navigation funktioniert benötigt Ihre Seite ein paar einfache Strukturen. Erzeugen Sie eine Handvoll Ordner, die die Struktur des Zoos abbilden. Benutzen Sie dafür das folgende Layout (siehe Abbildung 5-1).

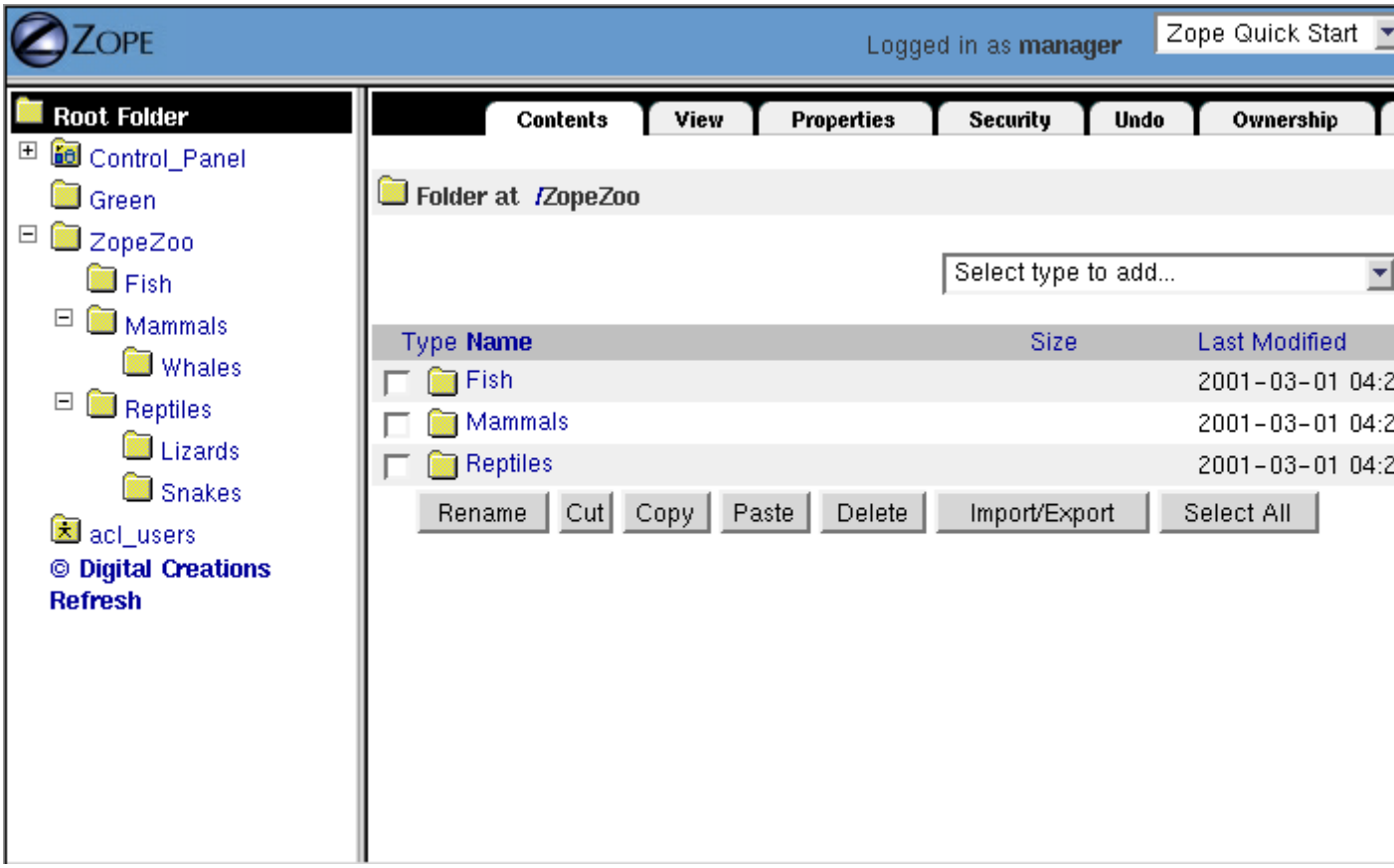


Abbildung 5-1 Zoo Ordner-Struktur.

Die Hauptstruktur des Zoos enthält 3 übergeordnete Ordner, *Reptiles*, *Mammals* und *Fish*. Um in Ihrem Zoo zu navigieren sollten die Besucher zuerst Ihre Hauptseite besuchen. Anschließend können Sie einen der 3 übergeordneten Ordner anklicken, um einen bestimmten Bereich des Zoos zu betreten. Sie sollten außerdem ein ähnliches Prinzip benutzen können, um tiefer in den Zoo, zum Beispiel zur snakes-Abteilung, vorzudringen. Dem Besucher sollte es zusätzlich möglich gemacht werden, einen Abschnitt wieder zu verlassen und in einen übergeordneteren Bereich zu kommen.

Diese Arbeiten können mit Zope auf einfache Weise gelöst werden. Erstellen Sie eine DTML-Methode (DTML Method) *navigation* mit folgendem Inhalt in Ihrem *ZopeZoo*-Ordner:

```
    <ul>
      <dtml-in expr="objectValues('Folder')">
        <li><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a></li><br>
      </dtml-in>
    </ul>
```

Diese, gerade von Ihnen erzeugte Methode, generiert eine Liste mit Links zu den verschiedenen Unter-Abschnitten Ihres Zoos. Es ist wichtig zu wissen, dass diese Methode in jedem Ordner arbeiten kann, da sie keine Annahmen über den enthaltenden Ordner macht. Da wir die Methode im *ZopeZoo*-Ordner abgelegt haben, kann jeder Zoo-Ordner diese Methode akquirieren.

Now, you need to incorporate this method into the site. Let's put a reference to it in the *standard_html_header* object so that the navigation system is available on every page of the site. Your *standard_html_header* could look like this:

```
<html>
<head><title><dtml-var title></title></head>
<body>
<dtml-var navigation>
```

Next we need to add a front page to the Zoo site and then we can view the site and verify that the navigation works correctly.

Adding a Front Page to the Zoo

Now, you need a front page that serves as the welcome screen for Zoo visitors. Let's create a DTML Method in the *ZopeZoo* folder called *index_html* with the following content:

```
<dtml-var standard_html_header>

  <h1>Welcome to the Zope Zoo</h1>

  <p>Here you will find all kinds of cool animals.  You are in
  the <b><dtml-var getId></b> section.</p>

<dtml-var standard_html_footer>
```

Take a look at how your site appears by clicking on the *View* tab in the root folder, as shown in Figure 5-2.

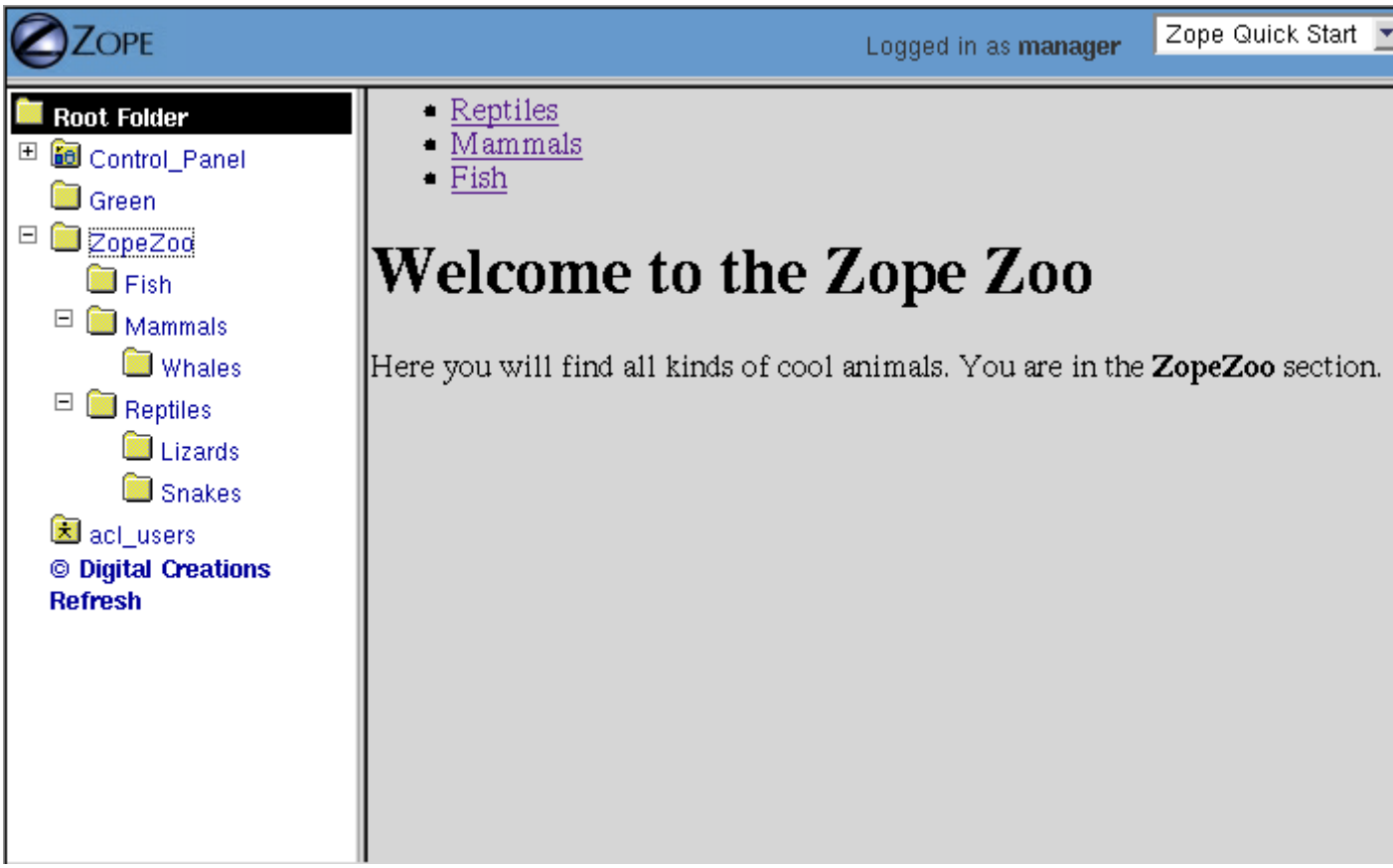


Figure 5-2 Zope Zoo front page.

Here you start to see how things come together. At the top of your main page you see a list of links to the various subsections. These links are created by the *navigation* method that is called by the *standard_html_header* method.

You can use the navigation links to travel through the various sections of the Zoo. Use this navigation interface to find the reptiles section.

Zope builds this page to display a folder by looking for the default folder view method ,*index_html*. It walks up the zoo site folder by folder until it finds the *index_html* method in the *ZopeZoo* folder. It then calls this method on the *Reptiles* folder. The *index_html* method calls the *standard_html_header* method which in turn calls the *navigation* method. Finally, the *index_html* method displays a welcome message and calls the *standard_html_footer*.

What if you want the reptile page to display something besides the welcome message? You can replace the *index_html* method in the reptile section with a more appropriate display method and still take advantage of the zoo header and footer including navigation.

In the *Reptile* folder create a DTML Method named *index_html*. Give it some content more appropriate to reptiles:

```
<dtml-var standard_html_header>
```

```
<h1>The Reptile House</h1>

<p>Welcome to the Reptile House.</p>

<p>We are open from 6pm to midnight Monday through Friday.</p>

<dtml-var standard_html_footer>
```

Now take a look at the reptile page by going to the *Reptile* folder and clicking the View tab.

Since the *index_html* method in the *Reptile* folder includes the standard headers and footers, the reptile page still includes your navigation system.

Click on the *Snakes* link on the reptile page to see what the Snakes section looks like. The snakes page looks like the *Reptiles* page because the *Snakes* folder acquires its *index_html* display method from the *Reptiles* folder.

Improving Navigation

The navigation system for the zoo works pretty well, but it has one big problem. Once you go deeper into the site you need to use your browser's *back* button to go back. There are no navigation links to allow you to navigate up the folder hierarchy. Let's add a navigation link to allow you to go up the hierarchy. Change the *navigation* method in the root folder:

```
<a href="..">Return to parent</a><br>

<ul>
<dtml-in expr="objectValues('Folder')">
  <li><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a><br></li>
</dtml-in>
</ul>
```

Now browse the Zoo site to see how this new link works, as shown in Figure Figure 5-3.

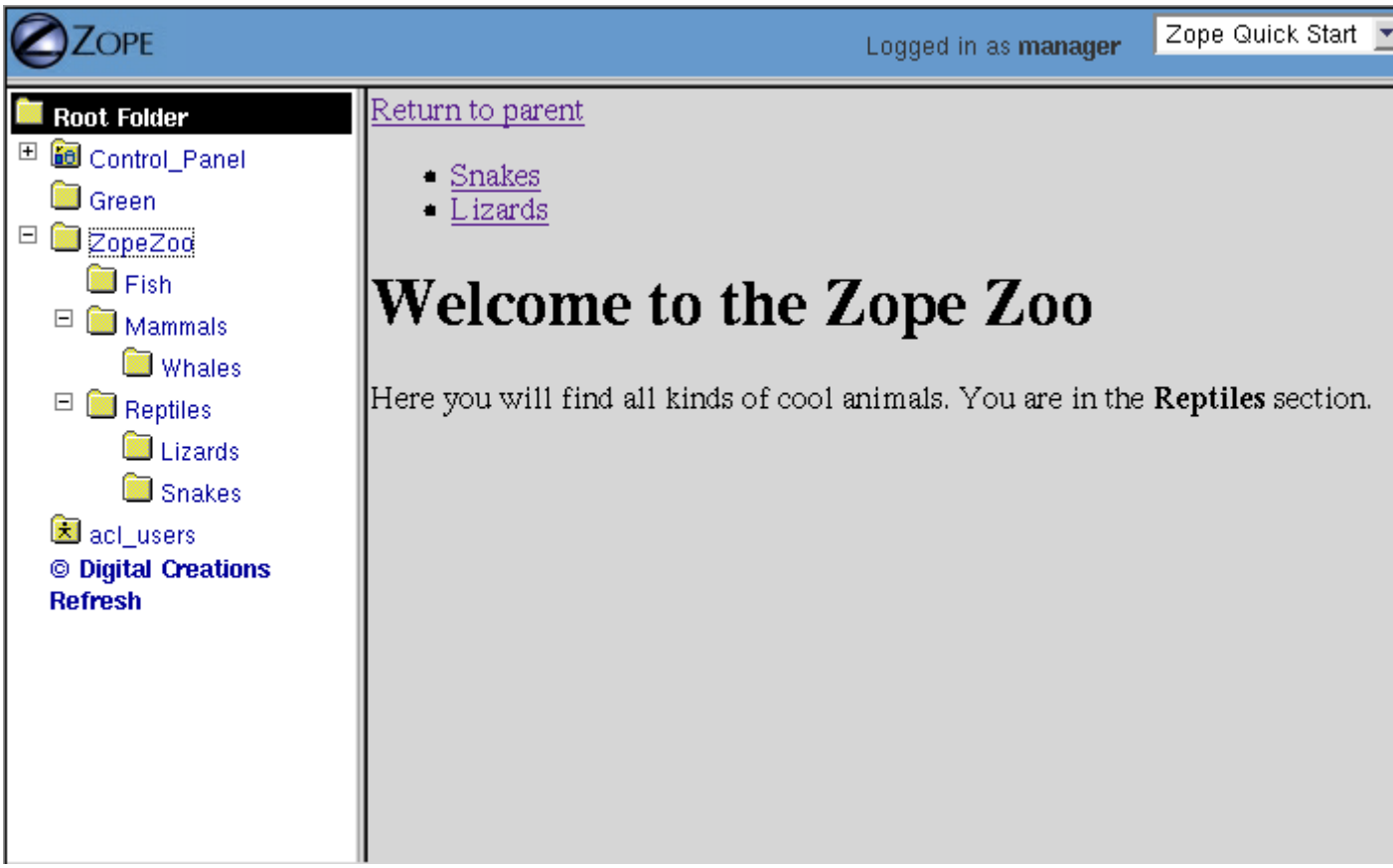


Figure 5-3 Improved zoo navigation controls.

As you can see, the *Return to parent* link allows you to go back up from a section of the site to its parent. However some problems remain; when you are at the top level of the site you still get a *Return to parent* link which leads nowhere. Let's fix this by changing the *navigation* method to hide the parent link when you're in the *ZopeZoo* folder:

```
        <dtml-if expr="_.len(PARENTS) > 2">
          <a href="..">Return to parent</a><br>
        </dtml-if>

        <ul>
        <dtml-in expr="objectValues('Folder')">
          <li><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a><br></li>
        </dtml-in>
        </ul>
```

Now the method tests to see if the current object has any parents before it display a link to the parent. *PARENTS* is a list of the current object's parents, and *len* is a utility function which returns the length of a list. See Appendix A for more information on DTML utility functions. Now view the site. Notice that now there is no parent link when you're viewing the main zoo page.

There are still some things that could be improved about the navigation system. For example, it's pretty hard to tell what section of the Zoo you're in. You've changed the reptile section, but the rest of the site all looks pretty much the same with the exception of having different navigation links. It would be nice to have each page tell you what part of the Zoo you're in.

Let's change the *navigation* method once again to display where you are:

```
        <dtml-if expr="_.len(PARENTS) > 2">
          <h2><dtml-var title_or_id> Section</h2>
          <a href="..">Return to parent</a><br>
        </dtml-if>

        <ul>
        <dtml-in expr="objectValues('Folder')">
          <li><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a><br></li>
        </dtml-in>
        </ul>
```

Now view the site again.

Figure 5-4 Zoo page with section information.

As you can see in Figure 5-4, the navigation method now tells you what section you're in along with links to go to different sections of the zoo.

Factoring out Style Sheets

Zoo pages are built by collections of methods that operate on folders. For example, the header method calls the navigation method to display navigation links on all pages. In addition to factoring out shared behavior such as navigation controls, you can use different Zope objects to factor out shared content.

Suppose you'd like to use CSS (Cascading Style Sheets) to tailor the look and feel of the zoo site. One way to do this would be to include the CSS tags in the *standard_html_header* method. This way every page of the site would have the CSS information. This is a good way to reuse content, however, this is not a flexible solution since you may want a different look and feel in different parts of your site. Suppose you want the background of the snakes page to be green, while the rest of the site should have a white background. You'd have to override the *standard_html_header* in the *Snakes* folder and make it exactly the same as the normal header with the exception of the style information. This is an inflexible solution since you can't vary the CSS information without changing the entire header.

You can create a more flexible way to define CSS information by factoring it out into a separate object that the header will insert. Create a DTML Document in the *ZopeZoo* folder named *style_sheet*. Change the contents of the document to include some style information:

```
<style type="text/css">
h1{
  font-size: 24pt;
  font-family: sans-serif;
}
p{
  color: #220000;
}
body{
  background: #FFFFDD;
}
</style>
```

This is a CSS style sheet that defines how to display *h1*, *p* and *body* HTML tags. Now let's include this content into our web site by inserting it into the *standard_html_header* method:

```
<html>
<head>
<dtml-var style_sheet>
</head>
<body>
<dtml-var navigation>
```

Now, when you look at documents on your site, all of their paragraphs will be dark red, and the headers will be in a sans-serif font.

To change the style information in a part of the zoo site, just create a new *style_sheet* document and drop it into a folder. All the pages in that folder and its sub-folders will use the new style sheet.

Creating a File Library

File libraries are common on web sites since many sites distribute files of some sort. The old fashioned way to create a file library is to upload your files, then create a web page that contains links to those files. With Zope you can dynamically create links to files. When you upload, change or delete files, the file library's links can change automatically.

Create a folder in the *ZopeZoo* folder called *Files*. This folder contains all of the file you want to distribute to your web visitors.

In the *Files* folder create some empty file objects with names like *DogGrooming* or *HomeScienceExperiments*, just to give you some sample data to work with. Add some descriptive titles to these files.

DTML can help you save time maintaining this library. Create an *index_html* DTML Method in the *Files* folder to list all the files in the library:

```
<dtml-var standard_html_header>

<h1>File Library</h1>

<ul>
<dtml-in expr="objectValues('File')">
  <li><a href="&dtml-absolute_url;"><dtml-var title_or_id></a></li>
</dtml-in>
</ul>

<dtml-var standard_html_footer>
```

Now view the *Files* folder. You should see a list of links to the files in the *Files* folder as shown in Figure 5-5.

Figure 5-5 File library contents page.

If you add another file, Zope will dynamically adjust the file library page. You may also want to try changing the titles of the files, uploading new files, or deleting some of the files.

The file library as it stands is functional but Spartan. The library doesn't let you know when a file was created, and it doesn't let you sort the files in any way. Let's make the library a little fancier.

Most Zope objects have a *bobobase_modification_time* method that returns the time the object was last modified. We can use this method in the file library's *index_html* method:

```
        <dtml-var standard_html_header>

        <h1>File Library</h1>

        <table>
          <tr>
            <th>File</th>
            <th>Last Modified</th>
           </tr>

        <dtml-in expr="objectValues('File')">
          <tr>
             <td><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a></td>
             <td><dtml-var bobobase_modification_time fmt="aCommon"></td>
```

```
          </tr>
        </dtml-in>

        </table>

        <dtml-var standard_html_footer>
```

The new file library method uses an HTML table to display the files and their modification times.

Finally let's add the ability to sort this list by file name or by modification date. Change the *index_html* method again:

```
        <dtml-var standard_html_header>

        <h1>File Library</h1>

        <table>
          <tr>
            <th><a href="&dtml-URL0;?sort=name">File</a></th>
            <th><a href="&dtml-URL0;?sort=date">Last Modified</a></th>
           </tr>

        <dtml-if expr="_.has_key('sort') and sort=='date'">
          <dtml-in expr="objectValues('File')"
                   sort="bobobase_modification_time" reverse>
            <tr>
               <td><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a></td>
               <td><dtml-var bobobase_modification_time fmt="aCommon"><td>
            </tr>
          </dtml-in>
        <dtml-else>
          <dtml-in expr="objectValues('File')" sort="id">
            <tr>
               <td><a href="&dtml-absolute_url;"><dtml-var
title_or_id></a></td>
               <td><dtml-var bobobase_modification_time fmt="aCommon"><td>
            </tr>
          </dtml-in>
        </dtml-if>

        </table>

        <dtml-var standard_html_footer>
```

Now view the file library and click on the *File* and *Last Modified* links to sort the files. This method works with two sorting loops. One uses the *in* tag to sort on an object's *id*. The other does a reverse sort on an object's *bobobase_modification_time* method. The *index_html* method decides which loop to use by looking for the *sort* variable. If there is a *sort* variable and if it has a value of *date* then the files are sorted by modification time. Otherwise the files are sorted by id.

Building a Guest Book

A guest book is a common and useful web application that allows visitors to your site to leave messages. Figure Figure 5-6 shows what the guest book you're going to write looks like.

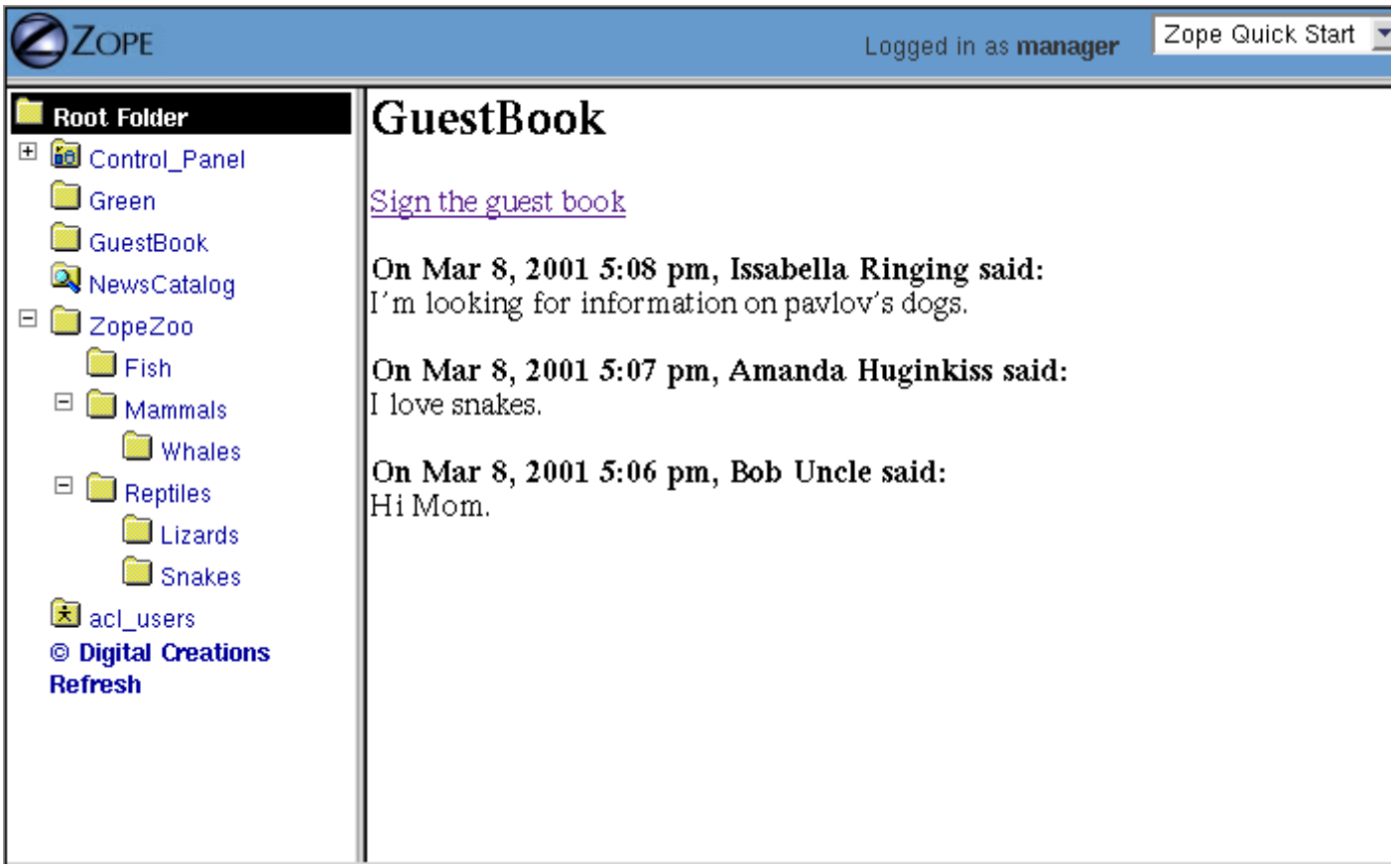


Figure 5-6 Zoo guest book.

Start by creating a folder called *GuestBook* in the root folder. Give this folder the title `The Zope Zoo Guest Book`. The *GuestBook* folder will hold the guest book entries and methods to view and add entries. The folder will hold everything the guest book needs. After the guest book is done you will be able to copy and paste it elsewhere in your site to create new guest books.

You can use Zope to create a guest book several ways, but for this example, you'll use one of the simplest. The *GuestBook* folder will hold a bunch of DTML Documents, one document for each guest book entry. When a new entry is added to the guest book, a new document is created in the *GuestBook* folder. To delete an unwanted entry, just go into the *GuestBook* folder and delete the unwanted document using the management interface.

Let's create a method that displays all of the entries. Call this method *index_html* so that it is the default view of the *GuestBook* folder:

```
<dtml-var standard_html_header>

<h2><dtml-var title_or_id></h2>

<!-- Provide a link to add a new entry, this link goes to the
```

```
        addEntryForm method -->

        <p>
          <a href="addEntryForm">Sign the guest book</a>
        </p>

        <!-- Iterate over each DTML Document in the folder starting with
        the newest documents first. -->

        <dtml-in expr="objectValues('DTML Document')"
                 sort="bobobase_modification_time" reverse>

        <!-- Display the date, author and contents of each document -->

          <p>
          <b>On <dtml-var bobobase_modification_time fmt="aCommon">,
             <dtml-var guest_name html_quote null="Anonymous">
said:</b><br>

          <dtml-var sequence-item html_quote newline_to_br>

          <!-- Make sure we use html_quote so the users can't sneak any
          HTML onto our page -->

        </p>

        </dtml-in>

        <dtml-var standard_html_footer>
```

This method loops over all the documents in the folder and displays each one. Notice that this method assumes that each document will have a *guest_name* property. If that property doesn't exist or is empty, then Zope will use *Anonymous* as the guest name. When you create a entry document you'll have to make sure to set this property.

Next, let's create a form that your site visitors will use to add new guest book entries. In the *index_html* method above we already created a link to this form. In your *GuestBook* folder create a new DTML Method named *addEntryForm*:

```
        <dtml-var standard_html_header>

        <p>Type in your name and your comments and we'll add it to the
        guest book.</p>

        <form action="addEntryAction" method="POST">
        <p> Your name:
          <input type="text" name="guest_name" value="Anonymous">
        </p>
        <p> Your comments: <br>
          <textarea name="comments" rows="10" cols="60"></textarea>
        </p>

        <p>
          <input type="submit" value="Send Comments">
        </p>
        </form>

        <dtml-var standard_html_footer>
```

Now when you click on the *Sign Guest Book* link on the guest book page you'll see a form allowing you to type in your comments. This form collects the user's name and comments and submits this information to a method named *addEntryAction*.

Now create an *addEntryAction* DTML Method in the *GuestBook* folder to handle the form. This form will create a new entry document and return a confirmation message:

```
<dtml-var standard_html_header>

<dtml-call expr="addEntry(guest_name, comments)">

<h1>Thanks for signing our guest book!</h1>

<p><a href="<dtml-var URL1>">Return</a>
to the guest book.</p>

<dtml-var standard_html_footer>
```

This method creates a new entry by calling the *addEntry* method and returns a message letting the user know that their entry has been added.

The last remaining piece of the puzzle is to write the script that will create a document and sets its contents and properties. We'll do this in Python since it is much clearer than doing it in DTML. Create a Python-based Script in the *GuestBook* folder called *addEntry* with parameters *guest_name* and *comments*:

```
## Script (Python) "addEntry"
##parameters=guest_name, comments
##
"""
Create a guest book entry.
"""
# create a unique document id
id='entry_%d' % len(context.objectIds())

# create the document
context.manage_addProduct['OFSP'].manage_addDTMLDocument(id,
                                          title="", file=comments)

# add a guest_name string property
doc=getattr(context, id)
doc.manage_addProperty('guest_name', guest_name, 'string')
```

This script uses Zope API calls to create a DTML Document and to create a property on that document. This script performs the same sort of actions in a script that you could do manually; it creates a document, edits it and sets a property.

The guest book is now almost finished. To use the simple guest book, just visit *http://localhost:8080/GuestBook/*.

One final thing is needed to make the guest book complete. More than likely your security policy will not allow anonymous site visitors to create documents. However the guest book application should be able to be used by anonymous visitors. In Chapter 7, User and Security, we'll explore this scenario more fully. The solution is to grant special permission to the *addEntry* method to allow it to do its work of creating a document. You can do this by setting the *Proxy role* of the method to *Manager*. This means that when the method runs it will work as though it was run by a manager regardless of who is actually running the method. To change the proxy roles go to the *Proxy* view of the *addEntry* method, as shown in Figure 5-7.

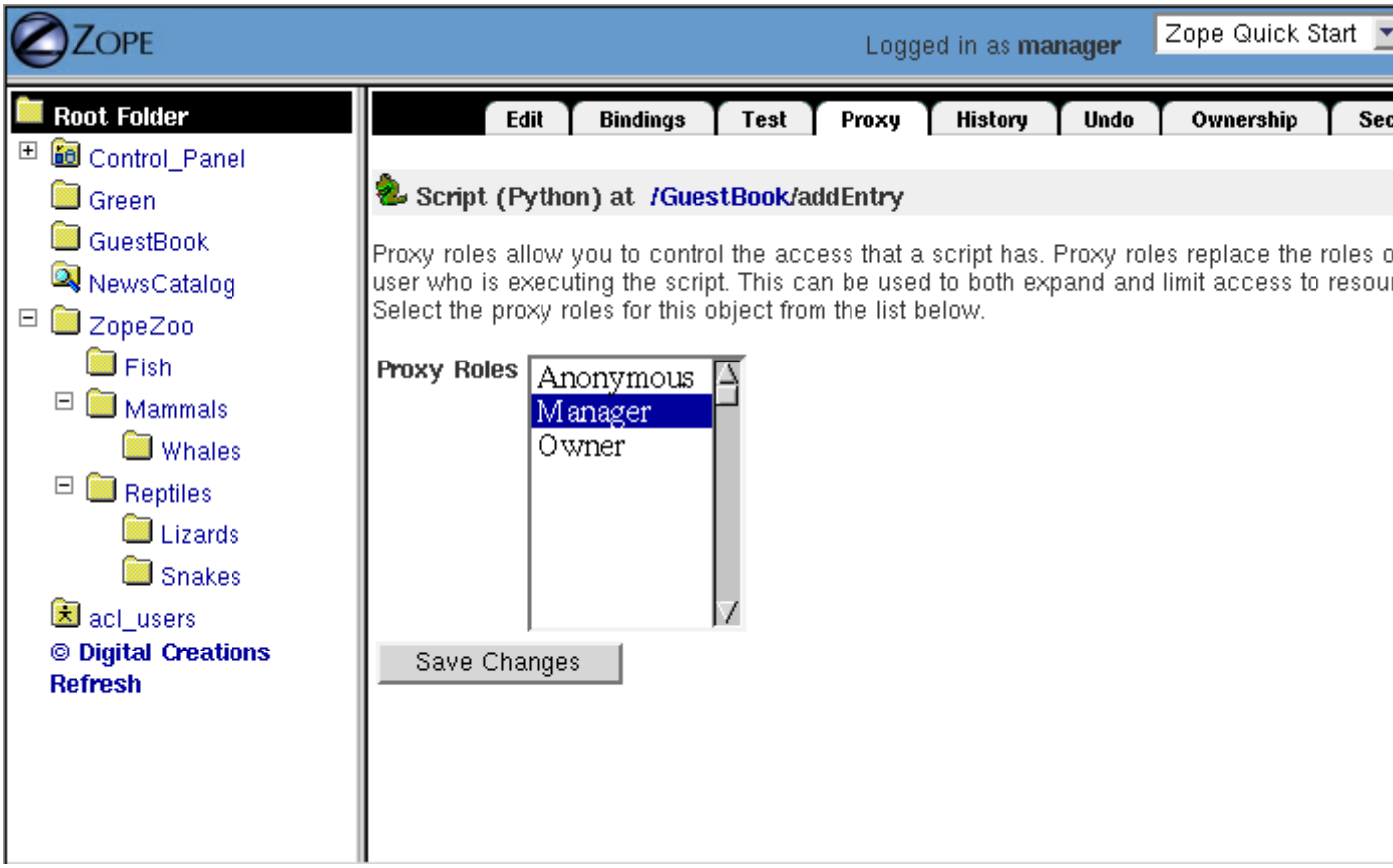


Figure 5-7 Setting proxy roles for the addEntry method.

Now select *Manager* from the list of proxy roles and click *Change*.

Congratulations, you've just completed a functional web application. The guest book is complete and can be copied to different sites if you want.

Extending the Guest Book to Generate XML

All Zope objects can create XML. It's fairly easy to create XML with DTML. XML is just a way of describing information. The power of XML is that it lets you easily exchange information across the network. Here's a simple way that you could represent your guest book in XML:

```
<guestbook>
```

```
  <entry>
    <comments>My comments</comments>
  </entry>
  <entry>
    <comments>I like your web page</comments>
  </entry>
  <entry>
    <comments>Please no blink tags</comments>
  </entry>
</guestbook>
```

This XML document may not be that complex but it's easy to generate. Create a DTML Method named "entries.xml" in your guest book folder with the following contents:

```
<guestbook>
  <dtml-in expr="objectValues('DTML Document')">
  <entry>
    <comments><dtml-var document_src html_quote></comments>
  </entry>
  </dtml-in>
</guestbook>
```

As you can see, DTML is equally adept at creating XML as it is at creating HTML. Simply embed DTML tags among XML tags and you're set. The only tricky thing that you may wish to do is to set the content-type of the response to *text/xml*, which can be done with this DTML code:

```
<dtml-call expr="RESPONSE.setHeader('content-type', 'text/xml')">
```

The whole point of generating XML is producing data in a format that can be understood by other systems. Therefore you will probably want to create XML in an existing format understood by the systems you want to communicate with. In the case of the guest book a reasonable format may be the RSS (Rich Site Summary) XML format. RSS is a format developed by Netscape for its *my.netscape.com* site, which has since gained popularity among other web logs and news sites. The Zope.org web site uses DTML to build a dynamic RSS document.

Congratulations! You've XML-enabled your guest book in just a couple minutes. Pat yourself on the back. If you want extra credit, research RSS enough to figure out how to change *entries.xml* to generate RSS.

The Next Step

This chapter shows how simple web applications can be made. Zope has many more features in addition to these, but these simple examples should get you started on create well managed, complex web sites.

In the next chapter, we'll see how the Zope security system lets Zope work with many different users at the same time and allows them to collaborate together on the same projects.

Kapitel 7: Benutzer und Sicherheit

Übersetzung von zopa (V 1.0)

Alle Web-Applikationen müssen mit Sicherheitsaspekten umgehen. Mit Sicherheitsaspekten umzugehen, bedeutet, zu kontrollieren, wer Zugang zu ihrer Applikation hat und festzulegen, was derjenige dort tun kann. Sicherheit ist kein Nachzügler, der einem laufenden System hinzugefügt werden kann. Stattdessen sollte Sicherheit ein wichtiges Design-Element sein, daß Sie bedenken, während Sie ihre Zope-Anwendungen aufbauen.

In diesem Kapitel werden Sie herausfinden, wie Sie Benutzer-Accounts einrichten und verwalten und wie die Weise zu kontrollieren ist, auf die Benutzer Zugriff auf ihre Anwendung haben, indem Sie Sicherheitsregeln aufstellen.

Sicherheit vorstellen

Sicherheit kontrolliert, was die Benutzer ihrer Website tun können und wie Sie und andere ihre Site pflegen können. Mit sorgfältigen Sicherheitsüberlegungen können Sie ihren Benutzern umfangreiche Möglichkeiten bieten und großen Gruppen von Menschen erlauben, auf sichere Weise zusammenzuarbeiten, um ihre Site zu pflegen. Wenn Sie sich um Sicherheit nicht kümmern, wird es schwierig sein, ihren -Benutzern sichere Kontrolle zu bieten und der Umgang mit ihrer Site wird ein lästiges Durcheinander werden. Ihre Site wird nicht nur unter Leuten leiden, die schädliche Dinge tun, zu denen sie gar nicht fähig sein sollten, sondern es wird auch schwer für Sie werden, ihren Benutzern irgendeinen Wert zu vermitteln und die zu kontrollieren, die ihre Site verwalten.

Zope flicht Sicherheit in fast jeden Aspekt der Web-Anwendungsaufbaus. Zope benutzt dasselbe Sicherheitssystem zur Kontrolle des Zope-Managements wie das, was Sie benutzen, um Benutzer für ihre Anwendung anzulegen. Zope macht keinen Unterschied zwischen dem Gebrauch und der Verwaltung einer Applikation. Zope mag verwirrend erscheinen, aber tatsächlich ermöglicht es Ihnen, Zopes Sicherheits-Framework für die Erfodernisse ihrer Anwendung einzustielen.

Bei Zope ein- und ausloggen

Wie wir in Kapitel 2 ("Zope benutzen") gesehen haben, loggen Sie sich bei Zope ein, indem Sie mit ihrem Web-Browser den Management-URL aufrufen und Benutzernamen und Paßwort eingeben. Wir haben in Kapitel 2 ("Zope benutzen") auch darauf hingewiesen, daß aufgrund der verbreiteten Browser-Arbeitsweise der Browser beendet werden muß, um sich aus Zope auszuloggen.

Wenn Sie versuchen, eine geschützte Ressource aufzurufen, für die Sie keine Zugangsrechte haben, wird Zope Sie auffordern, sich einzuloggen. Das kann auch geschehen, wenn Sie schon eingeloggt sind. Generell ist es nicht nötig, sich in Zope einzuloggen, wenn Sie nur öffentliche Ressourcen benutzen wollen.

Authentifizierung und Autorisierung

Sicherheit im weitesten Sinn kontrolliert zwei Funktionen, *Authentifizierung* und *Autorisierung*. Authentifizierung bedeutet, herauszufinden, wer Sie sind und Autorisierung bedeutet, festzulegen, was Sie tun können. Zope bietet getrennte Einrichtungen zur Verwaltung der Identifizierungsprozesse von Benutzern und Zugang zu kontrollierten Aktionen zu gewähren.

Wenn Sie eine geschützte Ressource aufrufen (wenn Sie sich beispielsweise eine private Website ansehen), wird Zope Sie auffordern, sich einzuloggen und nach ihrem Benutzer-Account suchen. Das ist der Authentifizierungsprozeß. Beachten Sie, daß Zope Sie nur authentifizieren wird, wenn Sie eine geschützte Ressource aufrufen; wenn Sie nur öffentliche Ressourcen aufrufen, wird Zope weiterhin annehmen, Sie seien anonym.

Sobald Sie authentifiziert sind, stellt Zope fest, ob Sie Zugang zu der geschützten Ressource haben oder nicht. Dieser Prozeß bezieht zwei vermittelnde Schichten zwischen Ihnen und der geschützten Ressource ein, nämlich *Rollen* und *Erlaubnisse*. Benutzer haben Rollen, die beschreiben, was sie tun dürfen, wie etwa "Autor", "Manager" oder "Redakteur". Zope-Objekte haben Erlaubnisse, die beschreiben, was mit ihnen getan werden kann, wie etwa "Ansehen", "Objekte löschen" oder "Eigenschaften verwalten".

Sicherheitsregeln teilen Rollen Erlaubnisse zu; in anderen Worten: Sie sagen, wer was tun darf. Zum Beispiel mag eine Sicherheitsregel die "Manager"-Rolle mit der Erlaubnis "Objekte löschen" assoziieren. Das erlaubt Managern, Objekte zu löschen. Auf diese Weise authorisiert Zope Benutzer, geschützte Aktionen durchzuführen.

In den folgenden Abschnitten werden wir einen genaueren Blick auf Authentifizierung und Autorisierung werfen und darauf, wie Sicherheitsregeln effektiv gesetzt werden können. Zuerst werden Sie etwas über Authentifizierung mit Hilfe von Benutzern und Benutzerverzeichnissen lernen, dann werden Sie etwas über die Kontrolle von Autorisierung mit Hilfe von Sicherheitsregeln herausfinden.

Authentifizierung und Verwaltung von Benutzern

Ein Zope-*Benutzer* definiert einen Benutzer-Account. Ein Zope-Benutzer hat einen Namen, ein Paßwort und optional zusätzliche Daten über jemenden, der Zope benutzt. Um sich in Zope einzuloggen, müssen Sie einen Benutzer-Account haben. Lassen Sie uns untersuchen, wie Benutzer-Accounts angelegt und verwaltet werden.

Benutzer in Benutzer-Verzeichnissen anlegen

Um Benutzer-Accounts in Zope anzulegen, fügen Sie Benutzer zu Benutzer-Verzeichnissen hinzu. In Kapitel 2 ("Zope benutzen") sollten Sie ihrem obersten Ordner bereits einen "Manager"-Benutzer hinzugefügt haben.

Lassen Sie uns einen neuen Benutzer anlegen, damiot ihr Mitarbeiter ihnen bei der Verwaltung der Zope-Site helfen kann. Gehen Sie zum Root-Ordner von Zope. Klicken Sie auf den Benutzer-Ordner namens *acl_users*. Der Benutzer-Ordner enthält Objekte, die Zope-Benutzer-Accounts definieren. Klicken Sie auf den *Add*-Button, um einen neuen Benutzer anzulegen.

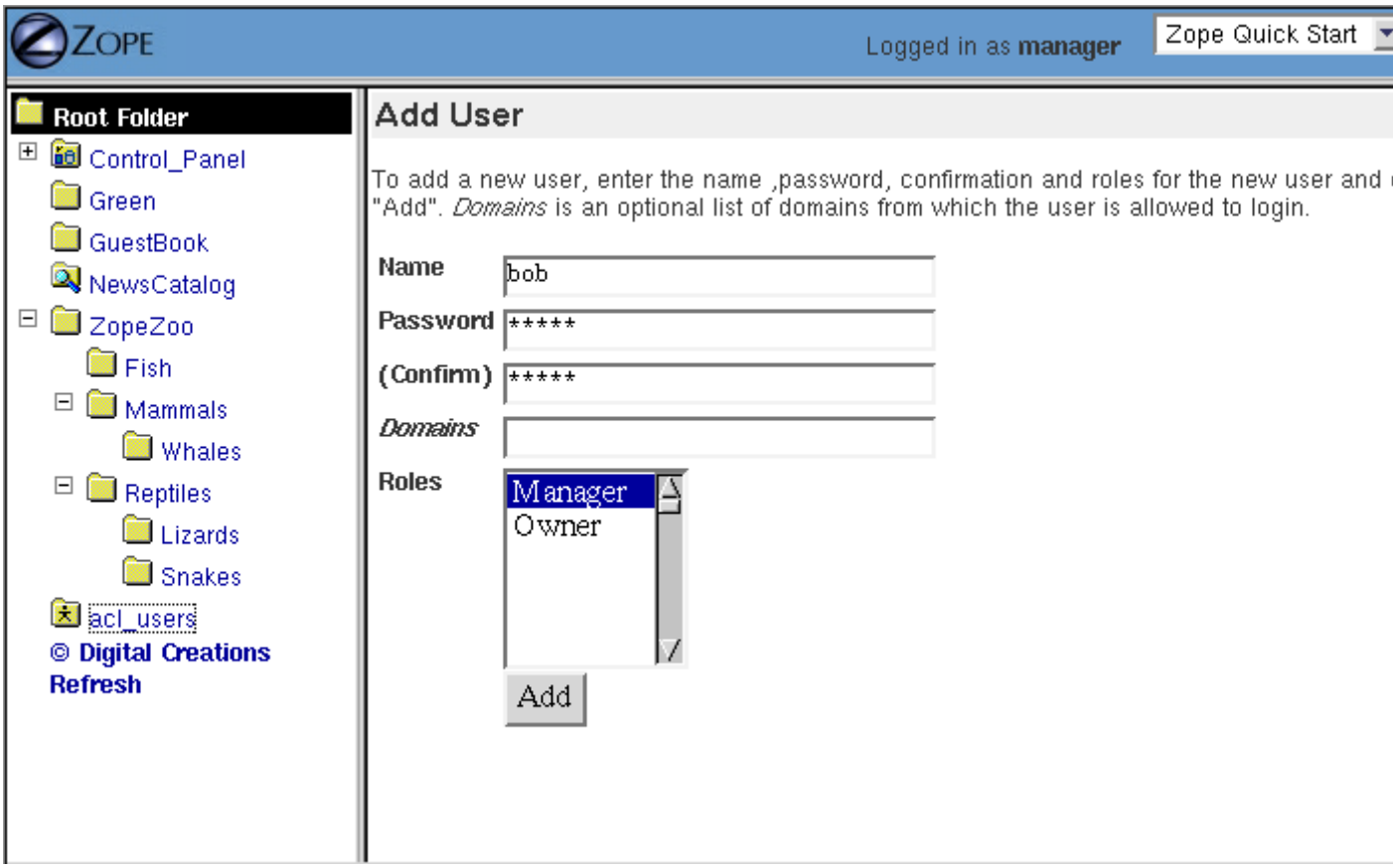


Bild 6.1 Einen Benutzer zu einem Benutzer-Ordner hinzufügen.

Das Formular in Bild 6.1 läßt Sie den Benutzer definieren. Geben Sie im Feld *Name* den Namen ihres Mitarbeiters ein, zum Beispiel "michel". Der Benutzername kann Buchstaben, Leerzeichen und Zahlen enthalten. Er ist fallempfindlich.

Wählen Sie ein Paßwort für ihren Mitarbeiter und geben sie es in das *Password*- und das *(Confirm)*-Feld ein. Wir werden die Dinge so aufbauen, daß ihr Mitarbeiter sein Paßwort später ändern kann, wenn er sich einloggt. Sie können ein Paßwort wie "change me" benutzen, um ihm zu helfen, sich an die Änderung des Paßworts zu erinnern.

Das Feld *Domains* läßt Sie die Internet-Domains einschränken, von denen aus sich der Benutzer einloggen kann. Das ermöglicht Ihnen, dem Account eine weitere Sicherheitskontrolle hinzuzufügen. Wenn Sie beispielsweise möchten, daß sich ihr Mitarbeiter immer von seinem Arbeitsplatz aus einloggt, könnten Sie die Domain des Arbeitsplatzes im Domain-Feld eingeben, zum Beispiel "myjob.com". Sie können mehrere Domains - getrennt durch Leerzeichen - angeben, um dem Benutzer das Einloggen von mehreren Domains aus zu ermöglichen. Wenn Sie beispielsweise beschließen, daß ihr Mitarbeiter Zope auch von zuhause aus verwalten soll, könnten Sie die Domains auf "myjob.com myhome.net" setzen. Sie können auch IP-Nummern mit Sternchen verwenden, um Wildcards anstelle von Domain-Namen zu verwenden, um Domains zu spezifizieren. Beispielsweise würde "209.67.167.*" alle IP-Adressen definieren, die mit "209.67.167" beginnen.

Die Auswahlliste *Roles* zeigt an, welche Rollen der Benutzer haben soll. Im Allgemeinen sollten Benutzer mit Verwaltungsaufgaben die *Manager*-Rolle zugewiesen bekommen. Im

Falle ihres Mitarbeiters wählen Sie die Manager-Rolle. Die *Owner*-Rolle ist in den meisten Fällen nicht angebracht, weil ein Benutzer normalerweise der Eigentümer ("Owner") bestimmter Objekte ist, nicht ein Eigentümer im Allgemeinen. Eigentümerschaft werden wir uns später im Kapitel genauer ansehen. Wir werden später auch sehen, wie Sie ihre eigenen Rollen definieren können wie etwa *Redakteur* oder *Lektor*.

Um einen neuen Benutzer anzulegen, klicken Sie auf den *Add*-Button. Sie sollten ein neues Benutzer-Objekt im Benutzer-Ordner sehen.

Benutzer bearbeiten

Sie können bestehende Benutzer bearbeiten, indem Sie sie anklicken. Das ruft ein Formular auf, das dem sehr ähnlich ist, daß Sie benutzt haben, um den Benutzer anzulegen. Tatsächlich können Sie hier alle Vorgaben kontrollieren, die wir gerade für dieses Formular diskutiert haben. Nachdem ihr Mitarbeiter sich bei dem für ihn Account eingeloggt hat, sollte er diesen Management-Bildschirm und sein Paßwort hier ändern.

Wie alle Verwaltungsfunktionen in Zope ist auch das Bearbeiten von Benutzer-Accountsvon den Sicherheitsregeln geschützt. Ein Benuitzer kann sein Paßwort nur ändern, wenn er die Erlaubnis *Manage Users* hat, die Manager als Voreinstellung besitzen.

Daher ist es nach Voreinstellung einem in einem bestimmten Benutzer-Ordner definierten Manager möglich, die Accounts anderer Manager zu ändern, wenn beide im selben Benutzerordner definiert sind. Das mag sein, was Sie wollen oder auch nicht. Später sehen wir uns Möglichkeiten an, dieses potentielle Problem zu vermeiden. Sie können allerdings beruhigt sein, denn es ist niemandem möglich, vom Management-Bildschirm aus ihr Paßwort herauszufinden. Ein anderer Manager mag Zugang zur *änderung* ihres Paßwortes haben, kann aber ihr gegenwärtiges Paßwort nicht herausfinden, ohne es zu ändern.

Im Allgemeinen funktionieren Benutzer-Ordner wie normale Zope-Ordner; Sie können darin Objekte anlegen, bearbeiten und löschen. Dennoch bieten Benutzer-Ordner nicht dieselben Möglichkeiten wie normale Ordner. Sie können in Benutzer-Ordnern nicht kopieren und einfügen und Sie können in einem Benutzer-Ordner nichts anderes anlegen als Benutzer.

Um einen existierenden Benutzer aus einem Benutzer-Ordner zu löschen, wählen sie einen Benutzer aus und klicken auf den *Delete*-Button. Denken Sie daran: Wie alle Zope-Aktionen kann das rückgängig gemacht werden, wenn Sie einen Fehler machen.

Benutzeranordnung festlegen

Zope kann mehrere Benutzer-Ordner an verschiedenen Stellen der Objekthierarchie enthalten. Ein Zope-Benutzer kann auf Ressourcen über dem Benutzer-Ordner, in dem er definiert ist, nicht zugreifen. Wo ihr Benutzer-Ordner definiert ist, entscheidet darüber, zu welchen Zope-Ressourcen Sie Zugang haben.

Wenn ihr Account in einem Benutzer-Ordner auf der Root-Ebene definiert ist, haben Sie Zugang zum Root-Ordner. Das ist wahrscheinlich der Ort, wo der Account, den Sie jetzt gerade benutzen, definiert ist. Sie können jedoch Benutzer in jedem Zope-Ordner definieren.

Betrachtenb Sie den Fall eines Benutzer-Ordners unter */WellnessSchule/Haar/acl_users*. Nehmen Sie an, daß der Benutzer *Edgar Scherenhand* in diesem Benutzer-Ordner definiert

ist. Edgar kann sich oberhalb des Ordners */WellnessSchule/Haar* in Zope einloggen. Edgars Blick auf die Zope-Site ist praktisch begrenzt auf den Ordner */WellnessSchule/Haar* und den darunterliegenden Bereich. Gelichgültig, welche Rollen Edgar zugeordnet sind, kann er nicht auf geschützte Ressourcen oberhalb dieses Ortes zugreifen.

Mit der Anwendung dieser Technik ist es leicht, einfache Sicherheitsregeln aufzubauen. Eins der verbreitetsten Zope-Verwaltungsmuster ist es, verwandte Objekte gemeinsam in einem Ordner zu plazieren und dann dort einen Benutzer-Ordner anzulegen, um die Leute zu definieren, die für diese Objekte verantwortlich sind.

Nehmen Sie beispielsweise an, daß die Leute in ihrer Organisation Uniformen tragen. Sie schaffen ein Intranet, das Informationen über ihre Organisation enthält, einschließlich Informationen über Uniformen. Sie mögen einen Ordner *Uniformen* irgendwo in der Intranet-Zope-Site anlegen. In diesem Ordner könnten sie Objekte ablegen wie Bilder von Uniformen und Beschreibungen, wie sie zu tragen und zu reinigen sind. Dann könnten Sie einen Benutzer-Ordner im Uniformen-Ordner anlegen und einen Account für die Chefschneiderin einrichten. Wenn ein neuer Uniform-Schnitt herauskommt, braucht die Schneiderin nicht den Webmaster zu bitten, die Site zu aktualisieren, sondern kann ihren eigenen Abschnitt das selbst tun, ohne irgendjemanden belästigen zu müssen. Außerdem hat die Chefschneiderin keinen Zugang zu Ordnern oberhalb von *Uniformen*, was bedeutet, daß sie keine Objeklte verwalten kann außer denen im *Uniformen*-Ordner.

Dieses Sicherheitsmuster wird *Delegation* genannt und ist in Zope-Anwendungen sehr verbreitet. Durch das Deligieren von verschiedenen Bereichen ihrer Zope-Site an verschiedene Benutzer können Sie die Last der Site-Administration von einer kleinen Gruppe von Managern nehmen und auf verschiedene spezifische Benutzergruppen verteilen. Später im Kapitel werden wir andere Sicherheitsmuster betrachten.

Mit alternativen Benutzer-Ordnern arbeiten

Es kann sein, daß Sie ihren Benutzer-Account nicht durch das Web verwalten wollen. Das mag daran liegen, daß Sie schon eine Benutzerdatenbank haben oder Sie wollen vielleicht andere Tools benutzen, um ihre Acoount-Informationen zu warten. Zope ermöglicht Ihnen, mit Hilfe von wechselnden Benutzer-Ordnern alle Arten von Autentifikationstechniken zu benutzen. Sie können auf der Zope-Website unter *http://www.zope.org/Products/user_management* viele wechselnde Benutzer-Ordner finden. Zur Zeit der Niederschrift gibt es 19 eingetragene wechselnde Benutzer-Ordner (Stand 08/2002: 55 Produkte; d.ü.). Hier steht eine Zusammenstellung einiger der populäreren alternativen Benutzer-Ordner zur Verfügung.

LoginManager
: Dies ist ein flexibler und mächtiger Benutzer-Ordner, der Ihnen erlaubt, ihre eigenen Authorisierungsmethoden einzubinden. Zum Beispiel können Sie LoginManager benutzen, um sich aus einer Datenbank heraus zu authentifizieren.

etcUserFolder
: Dieser Benutzer-Ordner authentifiziert über Standard-UNIX */etc/password* style files.

LDAPAdapter
: Dieser Benutzer-Ordner läßt Authetifizierung über einen LDAP-Server zu.

NTUserFolder
: Dieser Benutzer-Ordner authetifiziert aus NT-Benutzer-Accounts heraus. Er funktioniert nur, wenn Sie Zope unter Windows NT oder Windows 2000 betreiben.

Manche Benutzer-Ordner bieten wechselnde Kontrollen für Login und Logout wie etwa Login-Webformulare, eher als browserbasierte HTTP-Autorisierungskontrollen. Abgesehen von diesem Unterschied nutzen alle Benutzer-Ordner dieselbe Login-Prozedur des Abfragens ihrer Angaben, sobald Sie eine geschützte Ressource aufrufen.

Währedn die meisten Benutzer über Benutzer-Ordner der einen oder anderen Art verwaltet werden, hat Zope einige besondere Benutzer-Accounts, die nicht üpber den Benutzer-Ordner verwaltet werden.

Besondere Benutzer-Accounts

Zope bietet drei besondere Benutzer-Accounts, die nicht in Benutzer-Ordnern definiert sind, den *anonymen Benutzer*, den *Notfall-Benutzer* (emergency user) und den *Ursprungs-Manager* (initial manager). Der anonyme Benutzer wird gelegentlich benutzt, während die Accounts von Notfall-Benutzer und Ursprungs-Manager selten benutzt werden, es aber wichtig ist, über sie bescheid zu wissen.

Anonymer Zope-Benutzer

Zope hat einen eingebauten Benutzer-Account für Gäste, den anonymen Benutzer. Wenn Sie keinen Benutzer-Account auf Zope haben, werden Sie für den anonymen Benutzer gehalten.

Der anonyme Benutzer hat Sicherheitskontrollen wie jeder andere auch, er hat die Rolle *Anonymous*. Als Voreinstellung kann die Rolle *Anonymous* nur auf öffentliche Ressourcen zugreifen und keine Zope-Objekte ändern. Sie können diese Regel ändern, aber meistens werden Sie die voreingestellten Sicherheitseinstellungen angemessen finden.

Wie wir bereits oben im Kapitel erwähnt haben, müssen sie versuchen, auf eine geschützte Ressource zuzugreifen, um Zope zu einer Authentifizierung zu bringen. Schließlich hält Zope Sie auch dann für anonym, wenn Sie einen Benutzer-Account haben, solange Sie noch nicht eingeloggt sind.

Zope-Notfall-Benutzer

Zope hat einen besonderen Benutzer-Account für den Notfall-Gebrauch, bekannt als *Notfall-Benutzer* (emergency user). Wir haben den Notfall-Benutzer bereits in Kapitel 2 ("Zope benutzen") kurz besprochen. Der Notfall-Benutzer ist nicht durch normale Sicherheitseinstellungen beschränkt. Dennoch kann der Notfall-Benutzer keinerlei neue Objekte anlegen, ausgenommen neue Benutzer-Objekte.

Der Notfall-Benutzer ist nur für zwei Dinge wirklich geeignet: das Reparieren durcheinandergebrachter Berechtigungen und das Anlegen von Manager-Accounts. Wie wir in Kapitel 2 ("Zope benutzen") gesehen haben, können Sie sich als Notfall-Benutzer einloggen und einen Manager-Account anlegen, wenn keiner existiert. Nach dem Anlegen eines Manager-Accounts sollten sie sich als Notfall-Benutzer ausloggen und als Manager wieder einloggen.

Ein anderer Grund, den Notfall-Benutzer-Account zu nutzen, besteht, wenn Sie sich selbst aus Zope ausgesperrt haben, indem Sie Berechtigungen entfernt haben, die Sie brauchen, um

Zope zu verwalten. In diesem Fall loggen Sie sich als Notfall-Benutzer ein und stellen sicher, daß ihr Manager-Account die Berechtigungen `View management screens` und `Change permissions` hat. Dann loggen sie sich aus, loggen sich mit ihrem Manager-Account wieder ein und sollten genug Zugriffsmöglichkeiten haben, alles andere zu reparieren, was beschädigt ist.

Ein verbreitetes Problem mit dem Notfall-Benutzer ist der Versuch, ein neues Objekt anzulegen.

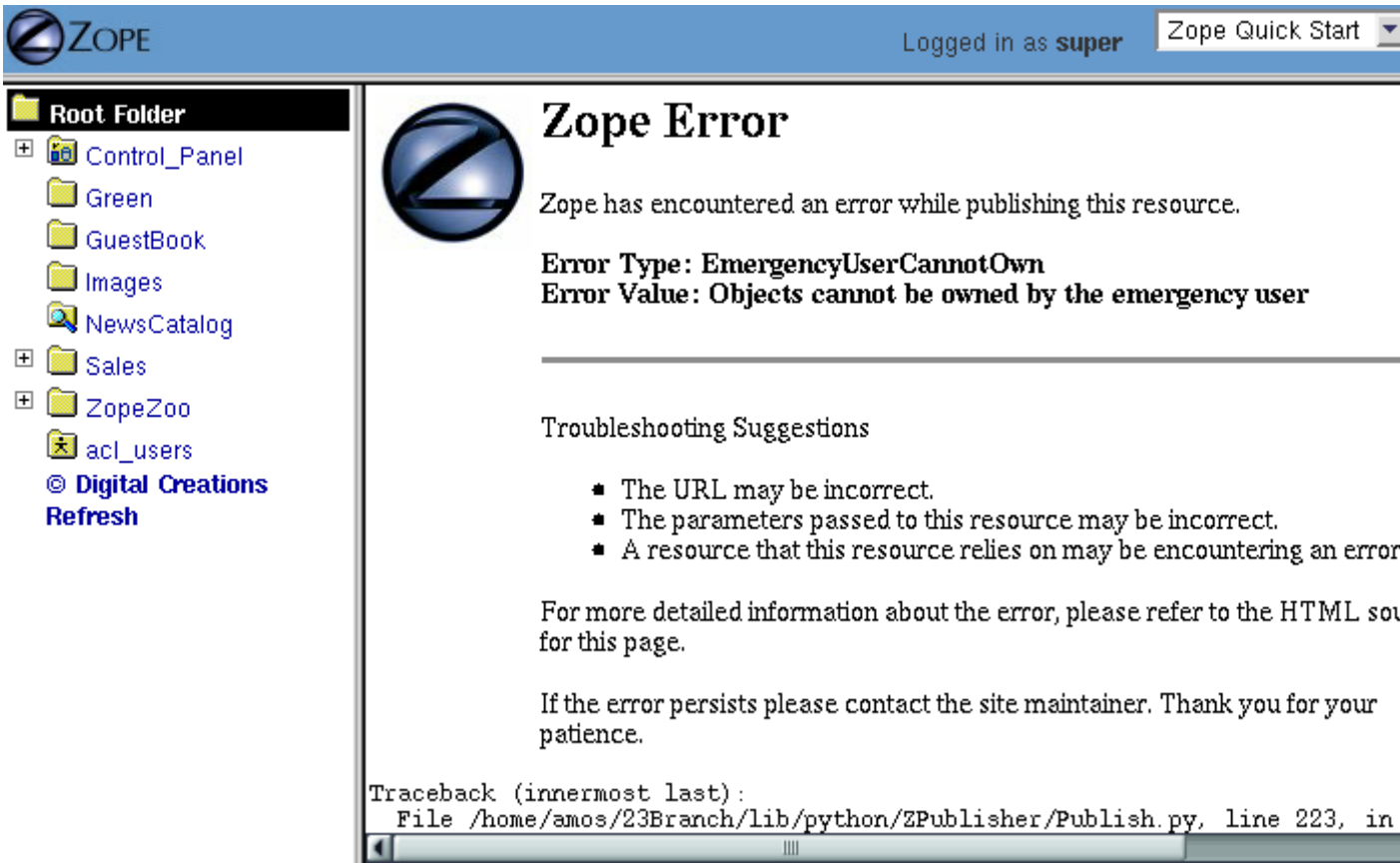


Bild 6.2 Error ausgelöst durch den Versuch, während des Logins als Notfall-Benutzer ein neues Objekt anzulegen.

Der Error in Bild 6.2 läßt Sie wissen, daß der Notfall-Benutzer keine neuen Objekte anlegen kann. Die Ursache dafüpr ist ein wenig komplex, wird aber später im Kapitel klarer werden, wenn wir Besitzerschaft behandeln. Die Kurzversion der Geschichte ist, daß es unsicher wäre, den Notfall-Benutzer Objekte anlegen zu lassen, da sie nicht Gegenstand derselben Sicherheitszwänge wären wie andere Objekte.

Einen Notfall-Benutzer anlegen

Anders als normale Benutzer-Accounts, die durch das Web definiert werden, wird der Notfall-Benutzer im Datei-System definiert. Sie können den Notfall-Benutzer-Account durch Bearbeitung der *access*-Datei im Zope-Verzeichnis ändern. Zope enthält eine Kommandozeilen-Anwendung, *zpasswd.py*, um den Notfall-Benutzer zu verwalten. Starten Sie *zpasswd.py*, indem Sie ihm den Pfad der Zugangsdatei übergeben:

```
$ python zpasswd.py access

Username: superuser
Password:
Verify password:

Please choose a format from:

SHA - SHA-1 hashed password
CRYPT - UNIX-style crypt password
CLEARTEXT - no protection.

Encoding: SHA
Domain restrictions:
```

Das *zpasswd.py*-Skript leitet Sie durch den Prozeß des Anlegens eines Notfall-Benutzer-Accounts. Beachten Sie, daß das von Ihnen eingegebene Paßwort nicht auf dem Bildschirm dargestellt wird. Sie können `zpasswd.py` auch ohne Argumente aufrufen, um eine Liste von Kommandozeiolen-Optionen zu erhalten.

Zope-Ursprungs-Manager

Der Ursprungs-Manager-Account wird vom Zope-Installer angelegt, damit Sie sich das erste Mal bei Zope einloggen können. Wenn Sie Zope das erste Mal installieren, sollten Sie eine Nachricht wie diese sehen:

```
creating default inituser file

Note:
The initial user name and password are 'admin' and 'IVX3kAwU'.

You can change the name and password through the web interface or
using the 'zpasswd.py' script.
```

Das teilt Ihnen den Namen das Paßwort des Ursprungs-Managers mit. Sie können diese Informationen benutzen, um sich zum ersten Mal als Manager bei Zope einzuloggen. Anschließend können Sie zusätzliche Benutzer-Accounts anlegen.

Ursprungs-Benutzer werden auf ähnliche Weise definiert wie der Notfall-Benutzer; sie sind in einer Datei im Dateisystem definiert, die *inituser* heißt. Das *zpasswd.py*-Programm kann benutzt werden, um diese Datei auf dieselbe Weise zu bearbeiten, wie sie benutzt wird, um die Notfall-Benutzer-Datei *access* zu editieren

```
$ python zpasswd.py inituser

Username: bob
Password:
Verify password:

Please choose a format from:

SHA - SHA-1 hashed password
```

```
CRYPT - UNIX-style crypt password
CLEARTEXT - no protection.

Encoding: SHA
Domain restrictions:
```

Das wird einen neuen Ursprungs-Benutzer namens "bob" einrichten und sein Paßwort setzen (das Paßwort wird nicht auf dem Bildschirm dargestellt, während Soie es eintippen). Wenn Zope startet, untersucht es diese Datei nach Benutzern und stellt sicher, daß sie sich in Zope einloggen können. Normalerweise werden Ursprungs-Benutzer vom Zope-Installer für Sie angelegt und Sie sollten sich nicht darum kümmern müssen, sie zu ändern. Wenn Sie zusätzliche Benutzer anlegen wollen, werden Sie das mit der Zope-Web-Management-Schnittstelle tun.

Bisher haben wir uns damit beschäftigt, wie Benutzer und Benutzer-Ordner Authentifizierung kontrollieren. Als Nächstes werden wir betrachten, wie Autorisierung mit Sicherheitsregeln kontrolliert wird

Autorisierung und Verwaltung von Sicherheit

Zope-Sicherheitsregeln kontrollieren Autorisierung; sie definieren, wer was tun kann. Sicherheitsregeln beschreiben, welche Rollen welche Berechtigungen haben. Rollen bezeichnen Klassen von Benutzern und Berechtigungen schützen Objekte. Daher definieren Sicherheitsregeln, welche Benutzer-Klassen (Rollen) welche Art von Aktionen (Berechtigungen) in einem bestimmten Teil der Site durchführen können.

Anstatt festlegen zu müssen, welcher bestimmte Benutzer welche bestimmten Aktionen mit welchem bestimmten Objekt durchführen darf, erlaubt Ihnen Zope, zu definieren, welche Art von Benutzern welche verschiedenen Aktionen in welchen Bereichen der Site durchführen können. Diese Art der Generalisierung macht ihre Sicherheitsregheln einfach und mächtig. Natürlich können Sie für bestimmte Benutzer, Aktionen und Objekte Ausnahmen von diesen Regeln machen.

In den folgenden Abschnitten werden wir Rollen, Berechtigungen und Sicherheitsregeln etwaxs näher untersuchen und einen Blick auf den Aufbau einfacher und effektiver Sicherheitsregeln werfen.

Mit Rollen arbeiten

Zope-Benutzer haben *Rollen*, die bestimmten, welche Arten von Aktionen sie durchführewn können. Rollen legen Benutzer-Klassen wie *Manager*, *Anonym* (Anonymous) und *Authentifiziert* (Authenticated) fest.

Rollen sind darin UNIX-Gruppen ähnlich, daß sie Gruppen von Benutzern abstrahieren. Und wie UNIX-Gruppen können Zope-Benutzer mehr als eine Rolle haben.

Rollen machen es leichter für Sie, Sicherheitsaspekte zu verwalten. Anstatt zu definieren, was jeder einzelne Benutzer tun kann, können Sie einfach ein paar verschiedene Sicherheitsregeln für verschiedene Benutzerrollen festlegen.

Zope hat vier eingebaute Rollen:

Manager
: Diese Rollen wird an Benutzer vergeben, die Standard-Verwaltungsfunktionen von Zope durchführen wie etwa anlegen und bearbeiten von Zope-Ordern und Dokumenten.

Anonym (Anonymous)
: Der anonyme Zope-Benutzer hat diese Rolle. Sie sollte die öffentlichen Ressourcen sehen ("view") dürfen. Generell sollte dieser Rolle nicht gestattet sein, Zope-Objekte zu ändern.

Besitzer (Owner)
: Diese Rolle wird automatisch Benutzern in dem Kontext zugeordnet, wo sie Objekte anlegen. Besitzerschaft werden wir später in diesem Kapitel behandeln.

Authentifiziert (Authenticated)
: Diese Rolle wird automatisch Benutzern zugeordnet, die gültige Authentifizierungsdaten übermittelt haben. Diese Rolle bedeutet, daß Zope "weiß", wer der bestimmte Benutzer ist.

Für einfache Zope-Sites kommen Sie mit Manager und Anonym klar. Für komplexere Sites mögen Sie ihre eigenen Rollen anlegen wollen, um ihre Benutzer in verschiedene Gruppen zu ordnen.

Rollen definieren

Um eine neue Rolle festzulegen, gehen Sie zur Registerkarte *Security* und scrollen bis zum Fuß des Bildschirms. Tippen Sie den Namen der neuen Rolle in das Feld *User defined role* und klicken Sie *Add Role*. Rollennamen sollten kurze, ein- oder zweiwortige Beschreibungen eines Benutzertyps sein wie etwa "Autor", "Site-Architekt" oder "Designer". Sie sollten Rollennamen aussuchen, die für ihre Anwendung relevant sind.

Sie sehen, daß ihre Rolle angelegt worden ist, wenn Sie beachten, daß es nun eine Rollen-Spalte für ihre neue Rolle am Kopf des Bildschirms gibt. Sie können auch eine Rolle löschen, indem sie die Rolle in der Auswahlliste am Fuß der Sicherheits-Seite markieren und dann den *Delete Role*-Button anklicken. Sie können nur ihre zusätzlich angelegten Rollen löschen und nicht die "Stamm"-Rollen, die Zope mitbringt.

Sie sollten beachten, daß Rollen auf der Ebene benutzt werden können, wo sie definiert sind und in der Objekthierarchie darunter. Wenn Sie also eine Rolle anlegen wollen, die auf ihrer gesamten Site verwendet werden soll, legen Sie sie im Root-Ordner an.

Im Allgemeinen sollten Rollen für große Abschnitte ihrer Site anwendbar sein. Wenn Sie dabei sind, Rollen anzulegen, um den Zugang zu Teilen ihrer Site zu begrenzen, gibt es möglicherweise bessere Wege, dasselbe Ziel zu erreichen. Zum Beispiel könnten Sie einfach die Sicherheitseinstellungen für die existierenden Rollen in dem Ordner ändern, den Sie schützen wollen oder Sie könnten Benutzer tiefer in der Objekthierarchie definieren, um ihren Zugang zu beschränken. Später im Kapitel werden wir mehr Beispiele dazu betrachten, wie Sicherheitsregeln definiert werden.

Lokale Rollen (local roles) verstehen

Lokale Rollen (lokal roles) sind eine fortgeschrittene Eigenschaft der Zope-Sicherheit. Benutzer können Extra-Rollen zugeordnet bekommen, wenn sie an einem bestimmten Objekt arbeiten. Wenn ein Objekt lokale Rollen mit einem Benutzer assoziiert hat, bekommt dieser Benutzer jene zusätzlichen Rollenm während er an dem Objekt arbeitet.

Wenn zum Beispiel ein Benutzer ein Objekt besitzt, ist ihm gewöhnlich auch die zusätzliche lokale Rolle des *Besitzers* (Owner) zugeordnet, während er mit dem Objekt arbeitet. Ein Benutzer mag generell nicht die Fähigkeit haben, DTML-Skripte zu bearbeiten, aber für DTML-Skripte, die er besitzt, könnte der Benutzer Zugang zum Bearbeiten des DTML-Skripts über die lokale Rolle des *Besitzers* haben.

Lokale Rollen sind ziemlich fortgeschrittene Sicherheitsmechanismen und werden nicht sehr oft gebraucht. Zopes automatische Kontrolle der lokalen Rolle *Besitzer* ist wahrscheinlich die einzige Stelle, wo sie lokalen Rollen begegnen.

Der Hauptgrund, aus dem Sie möglicherweise lokale Rollen manuell kontrollieren möchten, ist der Wunsch, einem bestimmten Benutzer besonderen Zugang zu einem Objekt zu gewähren. Sie sollten es nach Möglichkeit generell vermeiden, besondere Sicherheitseinstellungen für einzelne Benutzer vorzunehmen. Es ist einfacher, Sicherheitseinstellungen zu verwalten, die Gruppen von Benutzern betreffen als einzelne.

Berechtigungen verstehen

Berechtigungen definieren, welche Aktionen mit Zope-Objekten durchgeführt werden können. Genau wie Benutzer durch Rollen abstrahiert werden, werden Objekte durch Berechtigungen abstrahiert. Viele Zope-Objekte - einschließlich DTML-Skripten und DTML-Dokumenten - können beispielsweise betrachtet werden. Diese Aktion ist durch die Berechtigung *View* geschützt.

Manche Berechtigungen sind nur für einen Objekt-Typ relevant, wie zum Beispiel die Berechtigung zum ändern von DTML-Skripten, *Change DTML Methods*, die ausschließlich DTML-Skripte schützt. Andere Berechtigungen schützen viele Objekt-Typen wie etwa die Berechtigungen *FTP access* und *WebDAV access*, die kontrollieren, welche Objekte per FTP oder WebDAV zugänglich sind.

Sie können herausfinden, welche Berechtigungen bei einem bestimmten Objeklt zur Verfügung stehen, indem Sie zur Registerkarte *Security* gehen.

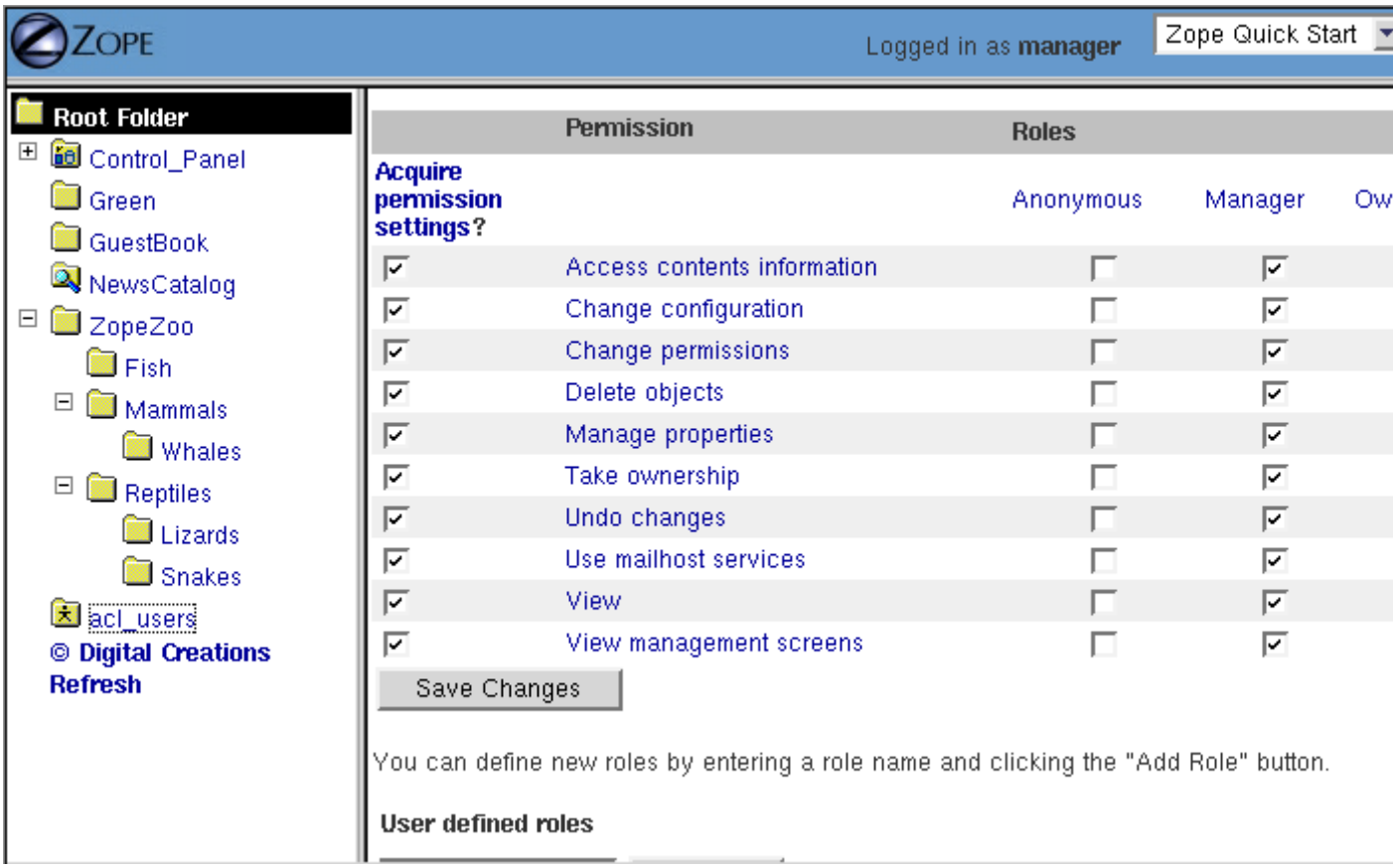


Bild 6.3 Sicherheitseinstellungen für ein Mail Host-Objekt.

Wie Sie in Bild 6.3 sehen können, verfügt ein Mail Host über eine eingeschränkte Berechtigungspalette. Vergleichen Sie das mit den vielen Berechtigungen, die Sie beim Festlegen der Sicherheitseinstellungen für einen Oredner sehen.

Sicherheitsregeln definieren

Sicherheitsregeln sind die Stelle, an der Rollen auf Berechtigungen treffen. Sicherheitsregeln definieren, wer was in einem bestimmten Teil der Site tun kann.

Sie können Sicherheitsregeln für fast jedes Zope-Objekt festlegen. Um eine Sicherheitsregel festzulegen, gehen Sie zur Registerkarte *Security*. Klicken Sie zum Beispiel auf die Sicherheits-Registerkarte des Root-Ordners.

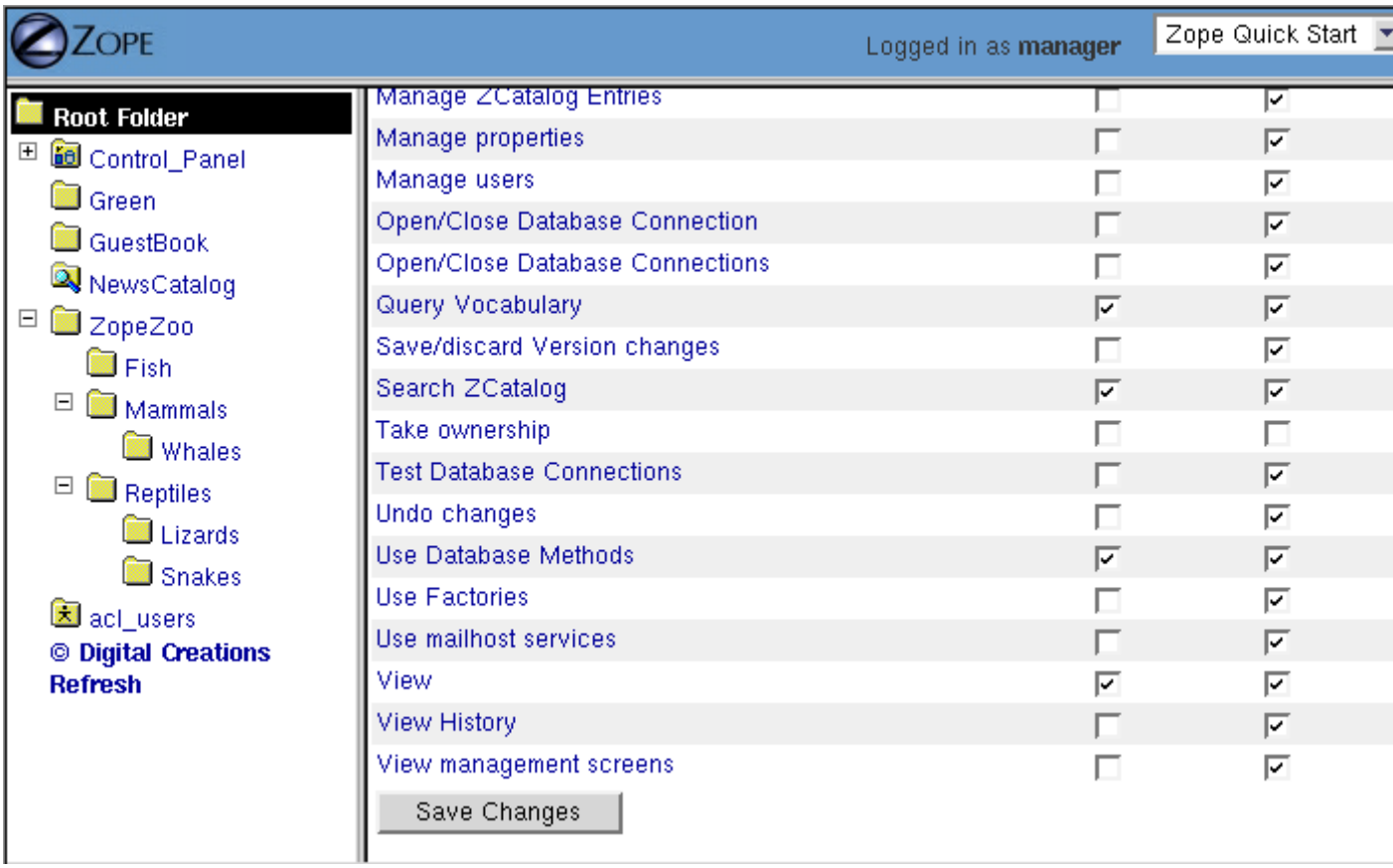


Bild 6.4 Sicherheitsregeln für den Root-Ordner.

Es geschieht eine Menge in Bild 6.4. Im Zentrum des Bildschirms befindet sich ein Gitter von Kontrollkästchen. Die vertikale Spalte des Gitters steht für Rollen, die horizontalen Reihen des Gitters für Berechtigungen. Das Anklicken des Kästchens an der Kreuzung zwischen einer Berechtigung und einer Rolle ordnet dem Benutzer mit dieser Rolle die Fähigkeit zu, Aktionen vorzunehmen, die von dieser Berechtigung geschützt werden.

Sie werden bemerken, daß Zope voreingestellte Sicherheitsregeln hat, die dem Manager erlauben, die meisten Aufgaben auszuführen und anonymen Benutzern nur erlauben, ein paar auszuführen. Sie können diese Regeln bearbeiten, um sie ihren Anforderungen anzupassen, indem Sie die Sicherheitseinstaellungen im Root-Ordner ändern.

Sie können beispielsweise ihre Site "privatisieren", indem Sie anonymen Benutzern verbieten, irgendwelche Webseiten zu sehen. Um das zu tun, verweigern Sie allen anonymen Benutzern den "View"-Zugang, indem Sie die *View*-Berechtigung abschalten, wo sie die Rolle *Anonymous* trifft. Sie können ihre gesamte Site privatisieren, indem Sie diese änderung der Sicherheitsregeln im Root-Ordner durchführen. Wenn Sie einen Teil ihrer Site privatisieren wollen, können Sie diese änderung in dem Ordner vornehmen, den Sie privatisieren wollen.

Dieses Beispiel weist auf einen sehr wichtigen Punkt von Sicherheitsregeln hin: Sie kontrollieren Sicherheit nur für einen bestimmten Teil der Site. Die einzige generelle Sicherheitsregel ist die des Root-Ordners.

Erwerben von Sicherheitsregeln

Wie interagieren verschiedene Sicherheitsregeln? Wir haben gesehen, daß man Sicherheitsregeln für verschiedene Objekte einrichten kann, aber was betsimmt, welche Regeln welches Objekt kontrollieren? Die Antwort ist, daß Objekte ihre eigenen Regeln beutzen, wenn sie welche haben, zusätzlich erwerben sie die Sicherheitsregeln von ihren Haupt-Dokumenten durch einen Prozeß, der *Erwerben*

Erwerben ist in Zope ein Menschanismus zum Teilen von Informationen zwischen Objekten, die in einem Ordner und seinen Unterordnern enthalten sind. Das Zope-Sicherheitssystem benutzt Erwerben, um Sicherheitsregeln zu teilen, damit der Zugang von Ordnern höherer Ebene aus kontrolliert werden kann.

Sie können das Erwerben von Sicherheitsregeln über die Registerkarte *Security* kontrollieren. Beachten Sie, daß es dort eine Spalte von Auswahlkästchen auf der linken Seite des Bildschirms gibt, die mit *Acquire permission settings* beschriftet ist. Jedes Auswahlfeld in dieser Spalte ist als Voreinstellung markiert. Das bedeutet, daß die Sicherheitsregel ergänzend zu allen Einstellungen in diesem Bildschirm die Einstellungen des Hauptdokuments für jede Berechtigung zu den Rollen-Einstellungen erwerben wird. Behalten Sie im Kopf, daß es im Root-Ordner (der kein übergeordnetes Hauptdokument hat) die äußerste, linke Spalte nicht gibt.

Nehmen Sie beispielsweise an, Sie würden diesen Ordner privat machen wollen. Wie wir schon gesehen haben, erfordert das bloß, der Rolle *Anonymous* die Berechtigung *View* zu verweigern. Wie Sie aber auf diesem Bildschirm sehen, hat die Rolle Anonymous hier keine View-Berechtigung und dennoch ist der Ordner nicht privat. Wie kommt das? Die Antwort ist, daß die Einstellung *Acquire permission settings* für die Berechtigung View aktiviert ist. Das bedeutet, daß die gegenwärtigen Einstellungen von den Sicherheitseinstellungen der übergeordneten Dokumente dieses Ordners übernommen worden sind. Irgendwo oberhalb dieses Ordners muß also der Rolle *Anonymous* die Berechtigung *View* zugewiesen sein. Siue können das überprüfen, indem Sie die Sicherheitsregeln der Haupt-Dokumente dieses Ordners untersuchen. Um den Ordner zu privatisieren, müssen wir die Markierung der Berechtigung *Acquire permission settings* aufheben. Das wird sicherstellen, daß ausschließlich die Einstellungen wirksam sind, die in dieser expliziten Sicherheitsregel gelten.

Im Allgmeinen sollten Sie immer Sicherheitseinstellungen übernehmen, wenn Sie nicht gerade einen besonderen Grund haben, das nicht zu tun. Das wird die Verwaltung ihrer Sicherheitseinstellungen viel einfacher machen, weil vieles vom Root-Ordner aus getan werden kann.

Als nächstes beschäftigen wir uns mit ein paar Beispielen, wie wirkungsvolle Sicherheitsregeln mit den Werkzeugen aufgestellt werden können, die Sie in diesem Kapitel bisher kennengelernt haben.

Anwendungsmuster Sicherheit

Die grundlegeneden Konzepte von Zope sind simpel: Rollen und Berechtigungen werden kombiniert, um Sicherheitsregeln zu schaffen und das Verhalten der Benutzer wird von diesen Regeln kontrolliert. Diese einfachen Werkzeuge können dennoch auf viele verschiedene Weise zusammengestellt werden. Das kann die Verwaltung von Sicherheitsaspekten komplex machen. Lassen Sie uns ein paar grundsätzliche Muster für die Verwaltung von Sicherheitsaspekten betrachten, die gute Beispiele für das Anlegen einer wirksamen und leicht zu verwaltenden Sicherheitsarchitektur liefern

Sicherheit: Faustregeln

Hier sind ein paar schlichte Leitlinien für die Sicherheitsverwaltung unter Zope. Die folgenden Sicherheitsmuster bieten spezifischere Rezepte, aber diese Leitlinien geben Ihnen etwas Orientierung, wenn Sie vor unerforschtem Gelände stehen.

1. Definieren Sie Benutzer auf iher höchsten Kontrollebene, aber nicht höher.
2. Gruppieren Sie Objekte, die von denselben Leuten verwaltet werden sollen, gemeinsam in denselben Ordnern.
3. Halten Sie die Dinge schlicht.

Die Regeln Eins und Zwei sind eng verwandt. Beide sind Teil einer eher generellen Regel für die Site-Architektur von Zope. Im Allgemeinen sollten Sie ihre Site überarbeiten, um verwandte Ressourcen und Benutzer nah beieinander anzuordnen. Natürlich ist es nie möglich, Ressourcen und Benutzer in eine strikte Hierarchie zu zwingen. Dennoch hilft eine wohlüberlegte Anordnung von Ressourcen und Benutzern in Ordnern und Unterordnern erheblich.

Versuchen Sie die Dinge schlicht zu halten, unabhängig von ihrer Site-Architektur. Je mehr Sie ihre Sicherheitseinstellungen komplizieren, umso schwieriger wird es, sie zu verstehen, zu verwalten und sicherzustellen, daß sie wirksam sind. Begrenzen Sie zum Beispiel die Zahl der neuen Rollen, die Sie anlegen und versuchen Sie, durch Erwerben von Sicherheitsregeln die Zahl der Orte zu begrenzen, wo Sie konkret Sicherheitseinstellungen definieren müssen. Wenn Sie bemerken, daß ihre Sicherheitsregeln, Benutzer und Rollen zu einem komplexen Dickicht verwachsen, sollten Sie überdenken, was Sie tun; es gibt vielleicht einen einfacheren Weg.

Allgemeine und lokale Regeln

Das grundlegende Sicherheitsmuster von Zope besteht darin, eine allgemeine Sicherheitsregel im Root-Ordner festzuilegen und diese Regel überall erwerben zu lassen. Dann können Sie - wo es nötig ist - zusätzliche Regeln tiefer in der Objekthierarchie festlegen, um die allgemeinen Regeln zu erweitern. Bemühen Sie sich, die Zahl der Orte zu begrenzen, an denen Sie die allgemeinen Sicherheitsregeln außer kraft setzen. Wenn Sie feststellen, daß Sie an vielen Stellen änderungen vornehmen müssen, bedenken Sie die Möglichkeit, die betreffenden Objekte aus den verschiedenen Ablageorten in einem Ordner zusammenzufassen, damit Sie die Sicherheitsregeln nur einmal bearbeiten müssen.

Sie sollten das Erwerben von Berechtigungsregeln in den Unter-Regeln übernehmen, es sei denn, ihre Unter-Regeln wären restriktiver als die allgemeinen Regeln. In diesem Fall sollten Sie die Markierung dieser Option für die Berechtigung aufheben, die nicht übernommen werden soll.

Dieses einfache Muster wird viele ihrer Sicherheitsbedürfnisse erfüllen. Sein Vorteil ist, daß es einfach zu verwalten und zu verstehen ist. Es sind extrem wichtige Charakteristika für jede Sicherheitsarchitektur.

Kontrolle an lokale Manager deligieren

Dieses Sicherheitsmuster ist für Zope sehr zentral und Teil dessen, was Zope seinen einzigartigen Geschmack gibt. Zope unterstützt Sie dabei, etwa Ressourcen in Ordnern zusammenzufassen und und dann Benutzer-Accounts in diesen Ordnern anzulegen, um ihren Inhalt zu verwalten.

Sagen wir, Sie wollen die Verwaltung des Ordners *Verkauf* auf ihrer Zope-Site dem neuen Web-Verkaufsmanager Stefan überlassen. Als erstes wollen Sie nicht, daß Stefan irgendwoanders dran herumfummelt als am Ordner *Verkauf*, also brauchen Sie ihn auch nicht dem acl-user-Ordner im Root-Verzeichnis hinzuzufügen. Stattdessen legen Sie einen neuen Benutzer-Ordner im Ordner *Verkauf* an.

Nun können Sie Steve zum Benutzer-Ordner im Ordner *Verkauf* hinzufügen und ihm die Rolle *Manager* zuweisen. Stefan kann sich nun direkt im Ordner *Verkauf* einloggen, um seinen Bereich zu verwalten, indem er mit seinem Browser *http://www.zopezoo.org/Verkauf/manage* aufruft.

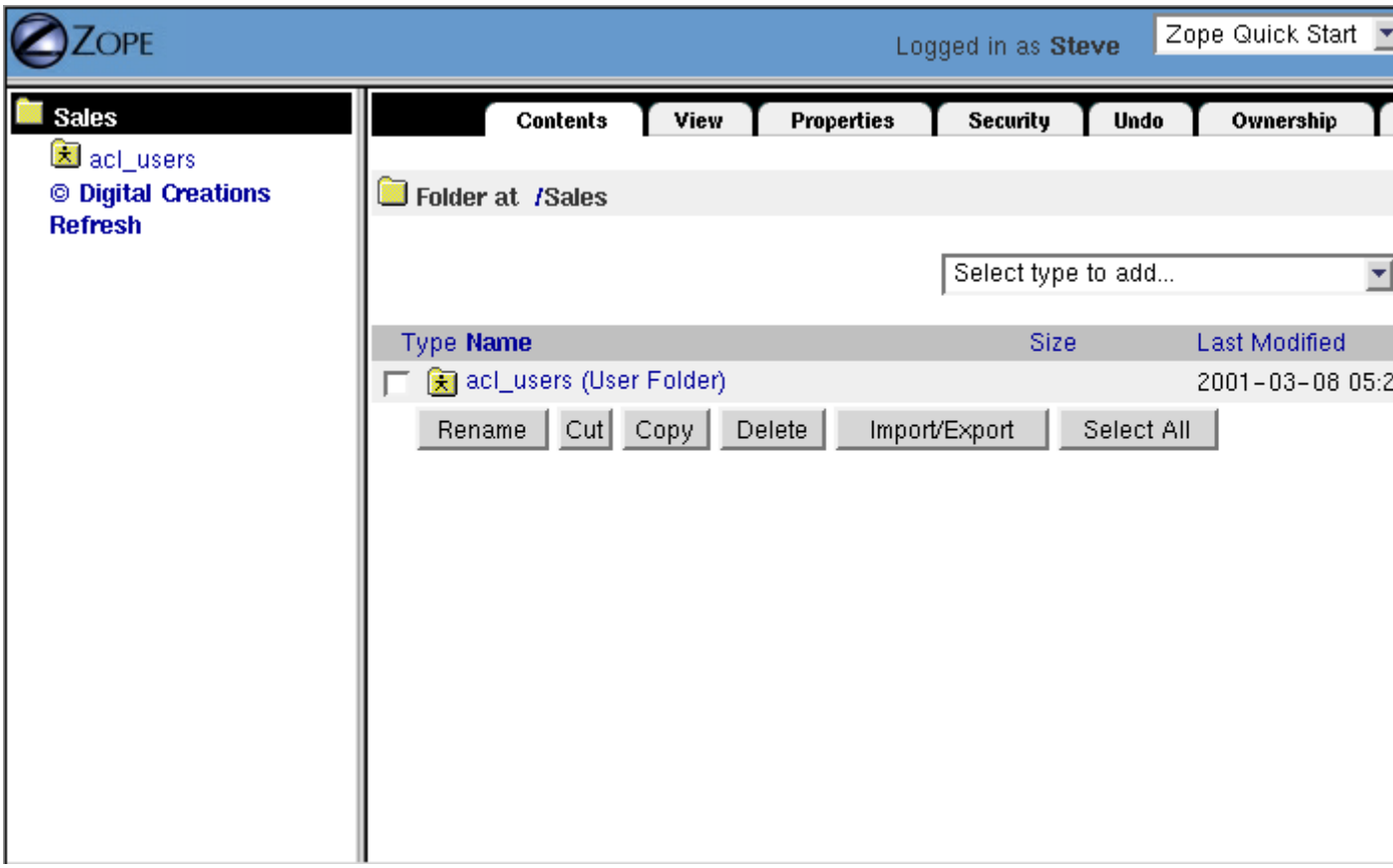


Figure 6-5 Den Ordner "Verkauf" verwalten.

Beachten Sie, daß in Bild 6.5 der Navigationsbaum auf der linken Seite den Ordner *Verkauf* als Root-Ordner anzeigt. Der lokale Manager, der für diesen Ordner defiiniert ist, wird niemals die Fähigkeit haben, sich in einen Ordner über *Verkauf* einzuloggen, daher wird er als oberster Ordner angezeigt.

Dieses Muster ist sehr mächtig, denn es kann rekursiv angewandt werden. Stefan kann beispielsweise einen Unterordner für Multi-Level-Marketing-Verkäufe anlegen. Dann kann er

einen Benutzer-Ordner anlegen und in diesem Unterordner anlegen, um die Kontrolle über diesen Ordner an den Multi-Level-Marketing-Manager zu delgieren. Und so weiter.as ist ein Rezept für *große* Websites, die von tausenden von Menschen verwaltet werden.

Die Schönheit dioeses Musters besteht darin, daß Manager in höheren Ebenen sich nicht so sehr damit beschäftigen müssen, was ihre Unterlinge tun. Wenn sie wollen, können sie von Nahem aufmerksam zusehen, aber sie können problemlos Einzelheiten ignorieren, denn sie wissen, daß die Untergebenen keinerlei änderungen außerhalb ihres Kontrollbereichs vornehmen können und sie wissen, daß ihre Sicherheitseinstellungen übernommen werden.

Verschiedene Zugangsebenen durch Rollen

Das Muster der lokalen Manager ist mächtig und skalierbar, aber es bietet einen eher groben Blick auf die Sicherheit. Entweder Sie haben Zugang oder eben nicht. Manchmal brauchen Sie eher feinere Kontrollmechanismen. Oft werden Sie Ressourcen haben, die von mehr als einem Benutzer-Typus benötigt werden. Rollen verschaffen Ihnen eine nLösung für dieses Problem. Rollen erlauben Ihnen, Benutzerklassen zu definieren und Sicherheitsregeln für sie festzulegen.

Bevor Sie neue Rollen anlegen, stellen Sie sicher, daß Sie sie wirklich brauchen. Angenommen, Sie haben eine Website, auf der Artikel veröffentlicht werden. Die öffentlichkeit liest Artikel und Manager bearbeiten und veröffentlichen Artikel, aber es gibt mit Autoren eine dritte Klasse von Benutzern, die Artikel schreiben, aber sie nicht bearbeiten oder veröffentlichen können.

Eine Lösung wäre, einen Autoren-Ordner anzulegen, wo Autoren-Accounts angelegt und ihnen *Manager*-Rollen zugewiesen werden. Dieser Ordner wäre privat, könnte also nur von Managern gesehen werden. Die Artikel könnten in diesem Ordner geschrieben werden und dann könnten Manager diese Artikel aus diesem Ordner herausbewegen, um sie zu veröffentlichen. Das ist eine angemessene Lösung, aber sie verlangt, daß Autoren nur in einem Teil der Site aus dem Autoren-Ordner heraus arbeiten und sie verlangt Extra-Arbeit von Managern, um die Artikel der Autoren aus dem Autoren-Ordner hinauszubewegen. Bedenken Sie auch die Schwierigkeiten, die es macht, wenn ein Autor einen Artikel überarbeiten will, der aus dem Autoren-Ordner entfernt wurde.

Eine bessere Lösung ist es, eine Rolle *Autor* hinzuzufügen. Eine Rolle hinzuzufügen, hilft uns, weil darüber ortsungebundene Zugangskontrollen möglich werden. Daher ermöglichen wir in unserem Beispiel mit dem Hinzufügen einer Autoren-Rolle das Schreiben, bearbeiten und Veröffentlichen von Artikeln überall auf der Site. Wir können eine allgemeine Sicherheitsregel anlegen, die Autoren die Berechtigung gibt, Artikel anzulegen und zu schreiben, aber ihnen die Berechtigung verweigert, sie zu veröffentlichen oder zu bearbeiten.

Rollen erlauben Ihnen, Zugang aufgrund dessen zu kontrollieren, wer ein Benutzer ist, nicht nur, wo er definiert ist.

Zugang zu Orten über Rollen kontrollieren

Rollen können helfen, mit einem subtilen Problem des Musters der lokalen Manager fertigzuwerden. Das Problem bestehtr darin, daß das Lokale-Manager-Muster eine strenge Kontrollhierarchie verlangt. Es gibt keine Möglichkeit, zwei verschiedenen Gruppen von Leuten den Zugang zu denselben Ressourcen zu gewähren, ohne daß die eine Gruppe

Manager der anderen Gruppe ist. Anders gesagt, gibt es keinen Weg für Benutzer, die im einen Teil der Site definiert sind, im anderen Teil Ressourcen zu verwalten.

Lassen Sie uns ein Beispiel nehmen, um die zweite Begrenzung des Lokal-Manager-Musters zu illustrieren. Stellen Sie sich vor, Sie unterhalten eine große Site für eine pharmazeutische Gesellschaft. Sie haben zwei Klassen von Benutzern: Wissenschatfler und Verkäufer. In der Hauptsache haben Wissenschaftler und Verkäufer verschiedene Web-Ressourcen. Nehmen Sie dennoch an, daß es einige Dinge gibt, die beide Sorten von Menschen verwalten müssen, wie etwa Anzeigen, die komplexe, wissenschaftliche Warnungen enthalten müssen. Wenn wir unsere Wissenschaftler im Ordner *Wissenschaft* definieren und die Verkäufer im Ordner *Verkauf*, wohin soll.en wir dann den Ordner *AnzeigenmitkomplexenWarnungen* stellen? Wenn nicht der Ordner *Wissenschaft* im Ordner *Verkauf* liegt oder umgekehrt, gint es keinen Ort, wo wir den Ordner *AnzeigenmitkomplexenWarnungen* anlegen können, und sowohl Wissenschaftler als auch Verkäufer ihn verwalten können. Es ist weder theoretisch noch praktisch eine gute Lösung, die Verkäufer von den Wissenschaftlern verwalten zu lassen oder umgekehrt; was ist möglich?

Die Lösung ist, Rollen zu verwenden. Sie wollten zwei Rollen auf einer Ebene über den Ordner von Wissenschaft und Verkauf anlegen, vielleicht *Wissenschaftler* und *Verkäufer*. Dann definieren Sie die Wissenschaftler und Verkäufer nicht innerhalb ihrer eigenen Ordner, sondern höher in der Objekthierarchie, sodaß beide Zugang zum Ordner *AnzeigenmitkomplexenWarnungen* haben.

Wenn Sie Benutzer auf dieser höheren Ebene anlegen, sollten Sie ihnen keine *Manager*-Rolle zuweisen, sondern ordnen Sie ihnen "Wissenschaftler" oder "Verkäufer" zuordnen. Dann sollten Sie die Sicherheitsregeln festlegen. Im Ordner *Wissenschaft* sollte die *Wissenschaftler*-Rolle das äquivalent der *Manager*-Kontrolle haben. Im Ordner *Verkauf* sollte die *Verkäufer*-Rolle dieselben Berechtigungen haben wie der *Manager*. Schließlich sollten Sie sowohl *Wissenschaftler* als auch *Verkäufer* im Ordner *AnzeigenmitkomplexenWarnungen* adäquate Berechtigungen geben. Auf diese Weise werden Rollen nicht benutzt, um verschiedenme Zugangsebenen zu schaffen, sondern Zugang zu verschiedenen Orten, indem sie darauf beruhen, wer Sie sind.

Eine andere häufige Situation beim Verwenden dieses Musters kann entstehen, wenn Sie ihre Benutzer nicht lokal definieren können. Sie mögen beispielsweise einen alternativen Benutzerordner verwenden, der die Definition aller Benutzer im Root-Ordner erzwingt. In diesem Fall würden Sie umfassenden Gebrauch von Rollen machen wollen, um den Zugang zu verschiedenen Orten auf Grundlage der Rollen zu beschränken.

Das schließt unsere Diskussion von Sicherheits-Mustern ab. Inzwischen sollten Sie eine angemessene Vorstellung davon haben, wie Benutzer-Ordner, Rollen und Sicherheitsregeln angewendet werden, um eine angemessene Sicherheitsarchitektur für ihre Anwendung anzulegen. Als nächstes werden wir zwei fortgeschrittene Sicherheitsaspekte behandeln, nämlich: Wie Sicherheitstests durchgeführt und ausführbare Inmhalte gesichert werden.

Sicherheitstests durchführen

Während der meisten Zeit brauchen Sie überhaupt keine Sicherheitstests durchzuführen. Wenn ein Benutzer versucht, eine gesicherte Operation durchzuführen, wird Zope ihn auffordern, sich einzuloggen. Wenn der Benutzer keine angemessenen Berechtigungen zum Zugang auf die geschützte Ressource besutzt, wird Zope ihm den Zugang verweigern.

Dennoch mögen Sie gelegentlich manuelle Sicherheitstests durchführen wollen. Der Hauptgrund, das zu tun, liegt darin, die Auswahl zu begrenzen, die einem Benutzer angeboten wird, wenn er sich authentifizieren soll. Das hindert einen raffinierten Benutzer nicht daran, auf geschützte Aktionen zugreifen zu wollen, aber es reduziert die Frustration von Benutzern, ihnen etwas, das nicht funktioniert, auch nicht anzubieten.

Die meistverbreitete Sicherheitsanfrage klärt, ob der gegenwärtige Benutzer eine bestimmte Berechtigung hat. Nehmen Sie zum Beispiel an, ihre Anwendung erlaubt Benutzern, Dateien hochzuladen. Diese Aktion wird durch die Zope-Standard-Berechtigung "Add Documents, Images, and Files" kontrolliert. Sie können mit DTML prüfen, ob der gegenwärtige Benutzer diese Berechtigung hat:

```
      <dtml-if expr="_.SecurityCheckPermission('Add Documents, Images, and
Files', this())">

        <form action="upload">
        ...
        </form>

      </dtml-if>
```

Die Funktion *SecurityCheckPermission* braucht zwei Argumente, nämlich einen Berechtigungsnamen und ein Objekt. In diesem Fall übergeben wir `this()` als das Objekt, was eine Refernz auf das gegenwärtige Objekt darstellt. Durch übergeben des gegenwärtigen Objekts stellen wir sicher, daß lokale Rollen einbezogen werden, wenn wir prüfen, ob der gegenwärtige Benutzer eine bestimmte Berechtigung hat.

Sie können etwas über den aktuellen Benutzer herausfinden, wenn Sie in DTML auf den Benutzer zugreifen. Der aktuelle Benutzer ist ein Zope-Objekt wie jedes andere und Sie können Aktionen mit ihm durchführen, indem Sie Skripte aus der API-Dokumentation benutzen.

Nehmen Sie an, Sie wollen den gegenwärtigen Benutzernamen auf einer Web-Seite darstellen, um die Seite zu personalisieren. Das können Sie ganz leicht in DTML:

```
      <dtml-var expr="_.SecurityGetUser().getUserName()">
```

Sie können den aktuell eingeloggten Benutzer über die DTML-Funktion *SecurityGetUser* auslesen. Dieses DTML-Fragment prüft den gegenwärtigen Benutzer, indem das Skript *getUserName* über das gegenwärtige Benutzer-Objekt aufgerufen wird. Wenn der Benutzer nicht eingeloggt ist, werden Sie den Namen des anonymen Benutzers bekommen, der *Anonymous User* lautet.

Als Nächstes sehen wir uns einen anderen fortgeschrittenen Aspekt ansehen, der die Sicherheit von DTML- und Skripten betrifft.

Fortgeschrittene Sicherheitsaspekte: Besitzerschaft und ausführbare Inhalte

Sie haben nun alle Grundlagen der Zope-Sicherheit behandelt. Was bleibt, sind die fortgeschrittenen Konzepte von *Besitzerschaft* (ownership) und *ausführbaren Inhalten*. Zope benutzt Besitzerschaft, um Objekte mit den Benutzern zu assoziieren, die sie anlegen und ausführbare Inhalte verweist auf auf Objekte wie Skripte und Dokumente, die Benutzer-Code ausführen.

Für kleine Sites mit vertrauenswürdigen Benutzern können Sie diese fortgeschrittenen Aspekte ignorieren. Wo Sie allerdings auf großen Sites nicht als vertrauenswürdig bekannten Benutzern erlauben, Zope-Objekte anzulegen oder zu bearbeiten, ist es wichtig, Besitzerschaft zu verstehen und ausführbare Inhalte abzusichern.

Das Problem: Angriffe mit Trojanern

Das grundlegende Szenario, daß ebenso Besitzerschaft wie ausführbare Inhalte interessant macht, ist ein *Trojaner*-Angriff. Ein Trojaner ist ein Angriff auf ein System, der einen Benutzer dergestalt austrickst, daß er eine potentiell schädliche Aktion durchführt. Ein typischer Trojaner maskiert sich als freundliches Programm, das Schaden anrichtet, wenn Sie es unbeabsichtigt ausführen.

Alle Web-basierten Plattformen - eingeschlossen Zope und viele andere - sind Gegenstand dieser Art von Angriff. Dazu ist es nur nötig, jemanden dazu zu bringen, einen URL aufzurufen, der eine schädliche Aktion durchführt.

Es ist sehr schwierig, sich gegen diese Art von Angriff zu schützen. Sie können sehr leicht jemanden dazu bringen, auf einen Link zu klicken oder Sie können fortgeschrittene Techniken wie JavaScript nutzen, um einen Benutzer auf einen schädlichen URL zu leiten.

Zope bietet einigen Schutz vor dieser Art von Trojanern. Zope hilft, ihre Site von der Server-Seite aus gegen Trojaner zu schützen, indem die Macht der Web-Ressourcen darauf gründet, wer ihr Autor ist. Wenn ein nicht-vertrauter Benutzer eine Web-Seite anlegt, ist die Macht, vertrauensseelige Benutzer zu schädigen, beschränkt. Nehmen Sie beispielsweise an, ein nicht-vertrauter Benutzer schreibt ein DTML-Dokument oder Python-Skript, das alle Seiten auf ihrer Site löschen soll. Wenn er versucht, die Seite aufzurufen, wird das scheitern, denn er hat die entsprechenden Berechtigungen nicht. Wenn ein Manager diese Seite aufruft, wird es auch scheitern, obwohl der Manager die entsprechenden Berechtigungen zur Durchführung der gefährlichen Aktion hat.

Zope benutzt die Besitzer-Informationen und die Kontrollen über ausführbare Inhalte, um diesen begrenzten Schutz zu gewähren.

Besitzerschaft verwalten

Wenn ein Benutzer ein Zope-Objekt anlegt, *besitzt* (own) er das Objekt. Ein Objekt, daß keinen Besitzer hat, wird als *unbesessen* (unowned) bezeichnet. Informationen zur Besitzerschaft werden im Objekt selbst gespeichert. Das ist so ähnlich, wie UNIX den Besitzer einer Datei behält.

Sie können herausfinden, wem ein Objekt gehört, indem Sie die Rewgisterkarte *Ownership* aufrufen, wie in Bild 6.6 zu sehen.

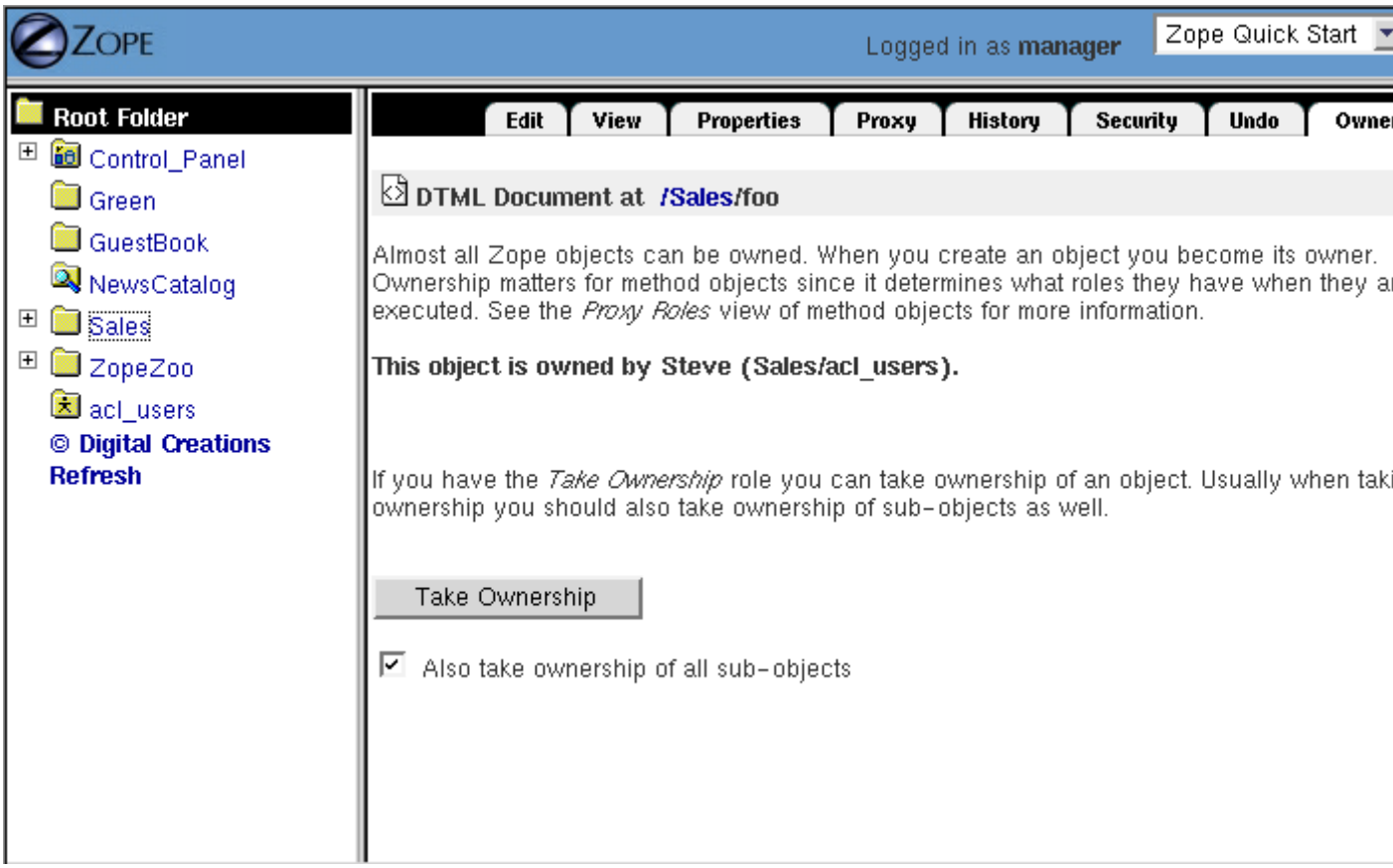


Bild 6.6 Besitzereinstellungen verwalten.

Dieser Bildschirm sagt Ihnen, ob das Objekt besessen wird und wenn ja, von wem. Wenn das Objekt von jemand anderem besessen wird und Sie haben die Berechtigung zur *Besitzübernahme* (Take ownership), können Sie die Besitzerschaft an einem Objekt übernehmen. Sie haben auch die Option, alle Unter-Objekte zu übernehmen, wenn Sie die Box *Take ownership of all sub-objects* anklicken. Besitzerschaft zu übernehmen ist meist nützlich, wenn der Besitzer-Account gelöscht worden ist oder wenn Objekte zur Bearbeitung an Sie weitergeleitet worden sind.

Wie wir oben erwähnt haben, hat Besitzerschaft Auswirkungen auf Sicherheitsregeln, weil ein Benutzer die lokale Rolle *Besitzer* (Owner) bei besessenen Objekten hat. Dennoch betrifft Besitzerschaft die Sicherheit, weil sie den ausführbaren Inhalt der Rolle kontrolliert.

Rollen ausführbarer Inhalte

Sie können manche Skripte in mancher Art Zope-Objekte durch das Web bearbeiten. Diese Objekte beinhalten DTML-Dokumente, DTML-Skripte, SQL-Skripte, Python-basierte Skripte und Perl-basierte Skripte. Diese Objekte werden als *ausführbar* bezeichnet, weil sie Skripte aufrufen, die durch das Web bearbeitet werden können.

Wenn Sie ein ausführbares Objekt besuchen, indem Sie zu seinem URL gehen oder es über DTML oder ein Skript aufrufen , führt Zope das Skript des Objekts aus. Dieses Skript wird beschränkt durch die Rollen des Besitzers dieses Objektes und ihren eigenen Rollen. Mit anderen Worten: Ein ausführbares Objekt kann nur Aktionen durchführen, für die *sowohl* der

Besitzer als auch der Betrachter berechtigt sind. Das hält einen unprivilegierten Benutzer davon ab, ein schädliches Skript zu schreiben und dann einen mächtigen Benutzer dazu zu bringen, es auszuführen. Sie können niemand anders dazu bringen, etwas zu tun, wozu Sie selbst nicht berechtigt sind. So nutzt Zope serverseitig Besitzerschaft als Schutz gegen Trojaner-Angriffe.

Proxy Roles

Manchmal ist Zopes System von begrenztem Zugang zu ausführbaren Objekten nicht genau das, was Sie wollen. Manchmal mögen Sie Sicherheit an einem Objekt festklammern wollen, egal wer auch immer es besitzt oder ausführt, als eine Form von Extra-Sicherheit. Zu anderen Zeiten werden Sie ein ausführbares Objekt mit besonderem Zugang bereitstellen wollen, um einem unprivilegierten Betrachter zu erlauben, geschützte Aktionen durchzuführen. *Proxy-Rollen* (Proxy roles) ermöglichen Ihnen, die Rollen eines ausführbarenm Objektes zu bearbeiten.

Nehmen wir an, Sie wollen eine Mail-Form anlegen, die es anonymen Benutzern erlaubt, dem Webmaster ihrer Site eMails zu schicken. Das Senden von eMails wird durch die Berechtigung `Use mailhost services` geschützt. Anonyme Benutzer dürfen diese Berechtigung normalerweise aus gutem Grund nicht haben. Sie wollen nicht, daß absolut jeder anonyme eMails über ihren Zope-Server versenden darf.

Das Problem mit dieser Einstellung ist, daß ihr DTML-Skript zum Senden von eMails für anonyme Benutzer nicht funktionieren wird. Wie können sie dieses Problem umgehen? Die Antwort ist, die Proxy-Rollen für ihr DTML-Skript zum Senden von eMails so zu setzen werden, daß es bei der Ausführung die "Manager"-Rolle hat. Gehen Sie zur Registerkarte Proxy Management ihres DTML-Skripts wie in Bild 6.7 gezeigt.

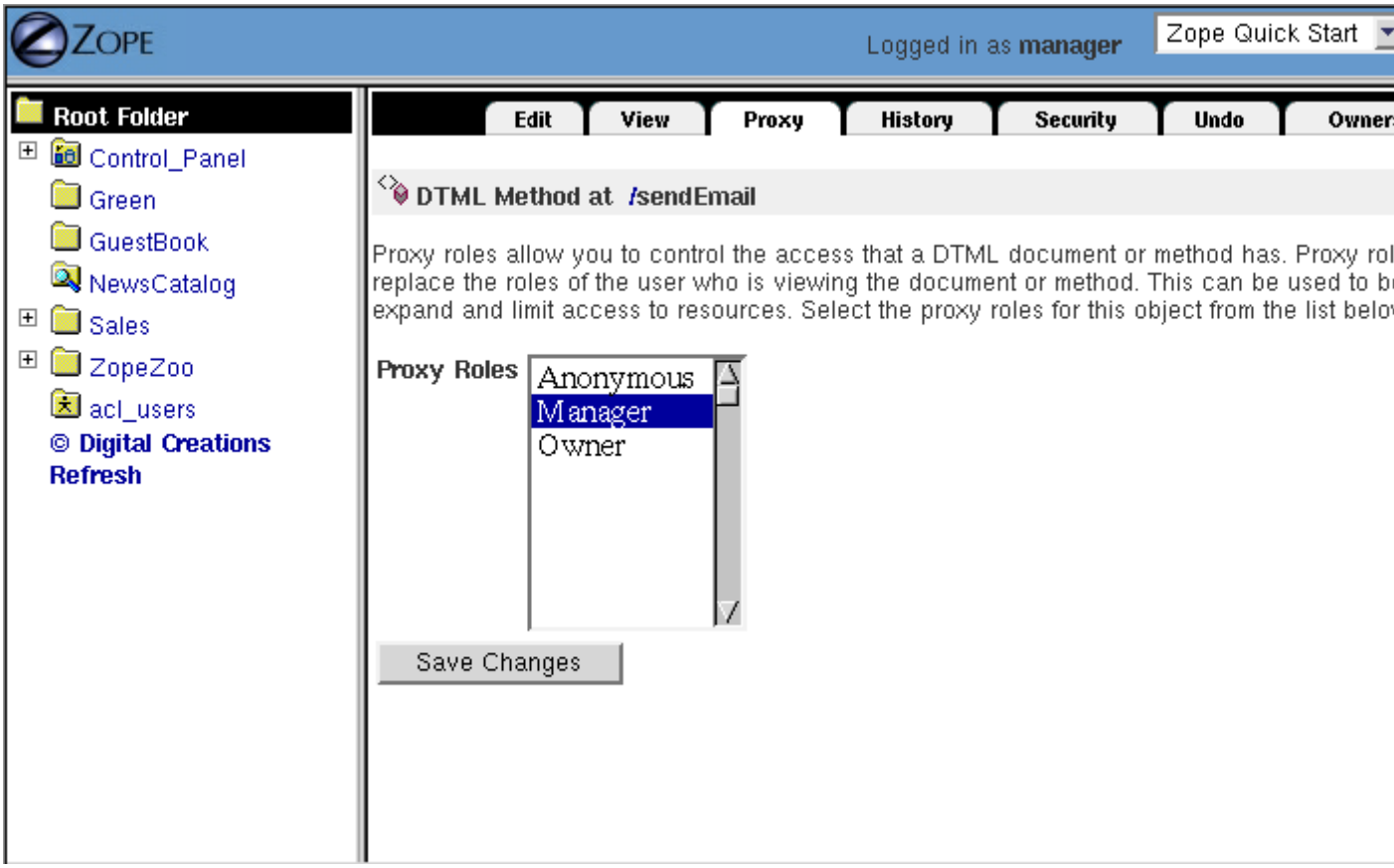


Figure 6.7 Verwaltung von Proxy-Rollen.

Wählen Sie *Manager* und klicken auf den *Change*-Button. Das wird die Proxy-Rollen des eMail-sendenden Skripts auf *Manager* setzen. Beachten Sie, daß Sie selbst die *Manager*-Rolle haben müssen, um sie als Proxy-Rolle festzulegen. Wenn nun irgendjemand - anonym oder nicht - ihr eMail-sendendes Skript aufruft, wird es mit der *Manager*-Rolle ausgeführt und daher autorisiert sein, eMails zu senden.

Proxy-Rollen definieren ein festgelegtes Set von Berechtigungen für ausführbare Inhalte. Daher können Sie sie genausogut benutzen, um Sicherheit einzuschränken. Wenn Sie zum Beispiel die Proxy-Rollen eines Skripts auf *Anonymous* setzen, wird das Skript nie irgendeine andere Rolle ausführen als *Anonymous*, ungeachtet der Rollen don Besitzer oder Betrachter.

Benutzen Sie Proxy-Rollen vorsichtig, da sie an den voreingestellten Sicherheitseinschränkungen vorbei benutzt werden können.

Zusammenfassung

Sicherheit besteht aus zwei Prozessen: Authentifizierung und Autorisierung. Benutzer-Ordner kontrollieren Authentifizierung, Sicherheitsregeln kontrollieren Autorisierung. Zope-Sicherheit ist eng an das Konzept des Ortes gebunden; Benutzer haben einen Ort, Sicherheitsregeln haben einen Ort, sogar Rollen können einen Ort haben. Der Aufbau einer wirksamen Sicherheitsarchitektur verlangt Aufmerksamkeit für den Ort. Im Zweifel halten Sie sich an die Anmwendungsmuster für Sicherheit, die in diesem Kapitel diskutiert worden sind.

Im nächsten Kapitel werden wird hochschalten und fortgeschrittene DTML untersuchen. DTML kann ein sehr mächtiges Werkzeug für Präsentation und Scripting sein. Sie werden etwas über viele neue Tags herausfinden und einen Blick auf einige DTML-spezifische Sicherheitskontrollen herausfinden, die in diesem Kapitel nicht behandelt worden sind.

Zopebuch: Inhaltsverzeichnis

Dieses Dokument ist momentan leider noch nicht vollständig übersetzt.

Kapitel 8: Variablen und DTML für Fortgeschrittene

DTML ist eine Sprache, die in der Regel "tut, was man meint". Das ist vor allem dann gut, wenn alles so läuft, wie man es sich vorstellt. Ist das aber nicht der Fall, so ist dies eher eine schlechte Eigenschaft. Dieses Kapitel soll Ihnen zeigen, wie Sie DTML dazu bringen, das zu tun, was Sie *wirklich* meinen. DTML hat einen gewissen Ruf, sehr komplex zu sein, und zu einem Teil stimmt dies auch. DTML ist sehr einfach, wenn es um einfache Layout-Aufgaben geht, die wir bisher kennen gelernt haben. Will man allerdings DTML für komplexere Aufgaben benutzen, so müssen wir ein gutes Verständnis der Herkunft von Variablen und deren Inhalten haben.

Nun folgt ein wirklich schwieriges Beispiel, über das fast alle "Neulinge" einmal stolpern. Stellen wir uns vor, wir haben ein DTML-Dokument namens *zooName*. Der Inhalt sieht aus wie folgt:

```
<dtml-var standard_html_header>

  <dtml-if zooName>

    <p><dtml-var zooName></p>

  <dtml-else>

    <form action="<dtml-var URL>" method="GET">
      <input name="zooName">
      <input type="submit" value="What is zooName?">
    </form>

  </dtml-if>

<dtml-var standard_html_footer>
```

Sieht einfach aus, nicht? Diese DTML-Seite ruft sich selber auf, da die HTML-Aktion auf die *URL* Variable verweist, welches natürlich die URL des DTML Documentes ist.

Wenn nun also die `zooName` Variable existiert, wird unsere Seite diese ausgeben. Existiert die Variable hingegen nicht, so wird ein Formular angezeigt, in dem wir diese Variable eingeben können. Wenn wir auf "submit" klicken, wird der eingegebene Wert die "if"-Abfrage wahr machen, und der folgende Code sollte den eingegebenen Wert auf der Seite ausgeben.

Leider ist dies einer der Fälle, in dem DTML nicht das tut, was man möchte, weil der Name des Dokuments ebenfalls *zooName* ist, und so wird nicht die Variable aus dem Request-Objekt benutzt, sondern das Dokument selbst, welches sich wiederum selber aufruft und so weiter, bis Zope mit einem "exessive Recursion" Fehler den Löffel abgibt. In den folgenden Abschnitten soll gezeigt werden, wie sich dieser Fehler beseitigen lässt und wie man DTML dazu bringt, das zu tun, was wir meinen.

Wie Variablen nachgesehen werden

Genau genommen gibt es zwei Wege, den Fehler im *zooName* Dokument zu beheben. Der erste besteht darin, das Dokument in*zopeNameFormOrReply* umzubenennen, sich für den Rest seiner Tage an diesen Sachverhalt zu erinnern und diesen Fehler niemals wieder zu machen, ohne zu wissen, was genu eigentlich der Fehler ist. Der zweite Weg besteht darin, sich darüber klar zu werden, wie Variablen und deren Werte in Zope gesucht und gefunden werden. Dann können wir Zope genau sagen, *wo* im Namensraum ("Namespace") wir den Namen der Variablen aufgelöst haben möchten.

Der DTML-Namensraum gleicht einer Ansammlung von Objekten auf einem Stapel ("*Stack*"). Ein Stapel ist eine Liste von Objekten, die durch "Drauflegen" ("push") und "Herunternehmen" ("pop") von neuen oder vorhandenen Objekten verändert werden kann. Zope erzeugt immer dann einen DTML-Namensraum, wenn DTML Dokumente oder -Methoden ausgeführt werden. Dieser Namensraum dient dann dazu, DTML-Variablennamen aufzulösen.

Es ist sehr wichtig, die inneren Abläufe im Namensraum zu verstehen, damit sich akkurat voraussagen lässt, auf welche Weise Zope unsere Variablen findet. Einige der schwierigsten DTML-Probleme lassen sich durch genaues Verständnis des DTML-Namensraum lösen.

Wenn Zope Variablennamen im DTML-Namensraum stack sucht, so schaut es zuerst auf das Objekt, welches "oben" auf dem Stapel liegt. Kann der Name dort nicht gefunden werden, so ist das nächst tiefere Objekt auf dem Stack an der Reihe und so weiter. Auf diese Weise arbeitet sich Zope bis auf den "Boden" des Stapels vor und versucht, den Variablennamen aufzulösen. Bleibt diese Suche erfolglos, so wird ein Fehler generiert und angezeigt.

Suchen wir nach einem nicht-existenten Namen, z. B. *unicorn*:

```
<dtml-var unicorn>
```

Gibt es keine Variable dieses Namens (*unicorn*), so erhalten wir bei Ansehen dieses Dokumentes den in Abbildung 7-1 gezeigten Fehler..

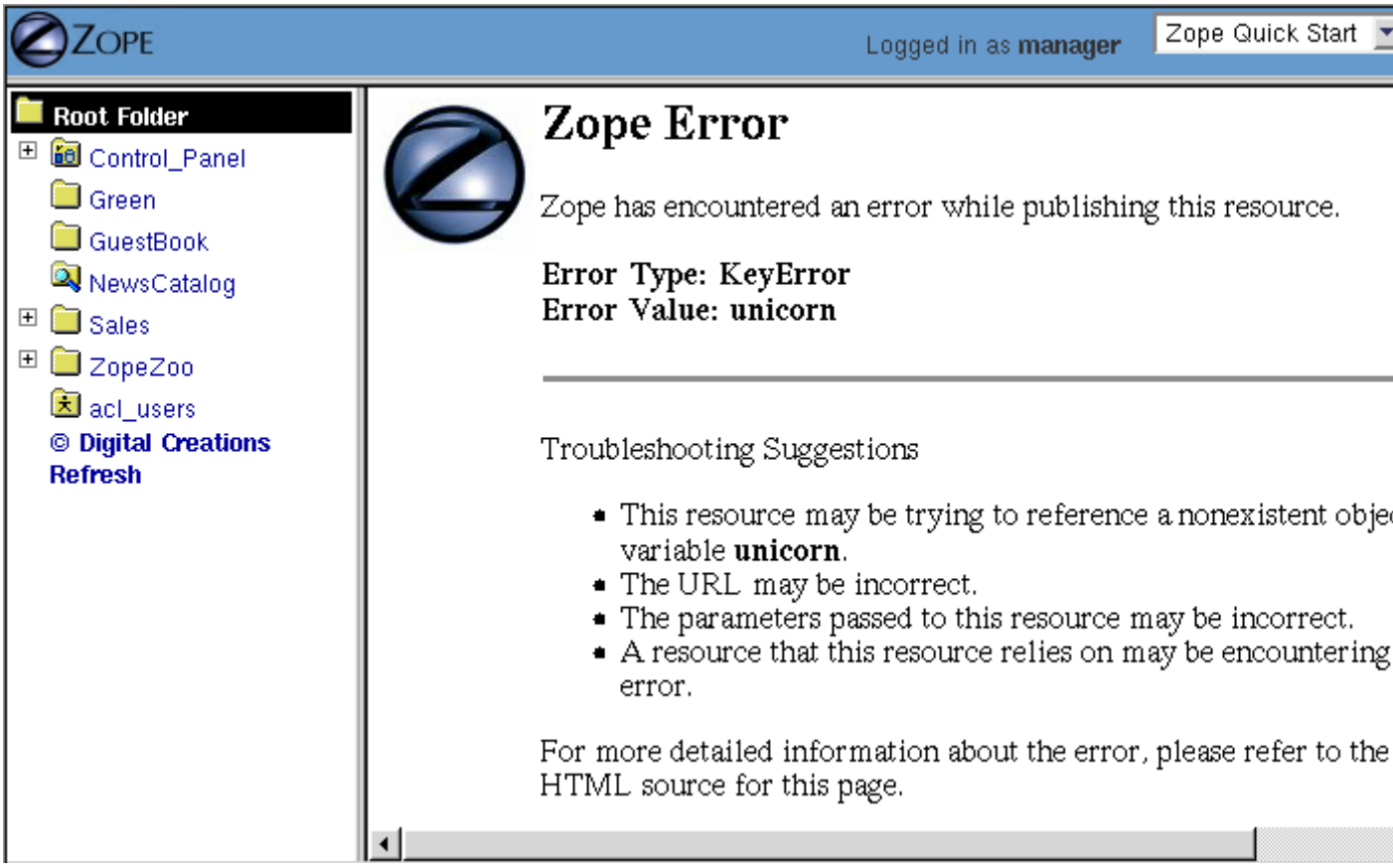


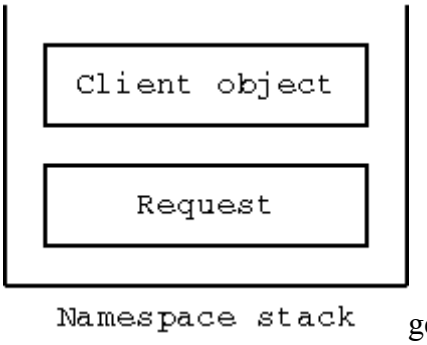
Abbildung 7-1 DTML Fehlermeldung über eine unbekannte Variable.

Der Namestack ist allerdings anfänglich nicht einfach leer, sondern er enthält bereits vor der Ausführung von DTML einige Objekte.

DTML-Namensräume

DTML-Namensräume werden dynamisch für jede Anforderung ("Request") erzeugt. Wird zum Beispiel eine DTML-Methode oder ein Dokument durch einen Web-Browser

angefordert, enthält der DTML-Namensraum zwei Objekte: Das client- und das Request-Objekt, wie in Abbildung 7-2



gezeigt.

Abbildung 7-2 Initialer DTML-Namensraum-Stack.

Das client-Object liegt oben auf dem Namensraum-Stack. Um welches Objekt es sich handelt, hängt davon ab, ob wir ein DTML-Dokument oder eine -Methode betrachten. In unserem obigen Beispiel heisst das client-Objekt *zooName*. Dies ist auch der Grund für den Fehler. Die Formular-Variable kommt eigentlich aus dem Request-Objekt, aber das Client-Objekt wird zuerst gefunden, weil es oben auf dem Stack liegt./p>

Der Namensraum des request-Objektes kommt immer an letzter Stelle im DTML-Namensraum, daher wird dieses Objekt auch als letztes nach Variablennamen durchsucht. Dies bedeutet, dass wir genau spezifieren müssen, welchen Namensraum wir verwenden möchten. Dies geschieht mit Hilfe des DTML-Tags `with`

```
<dtml-var standard_html_header>

  <dtml-with REQUEST only>
    <dtml-if zooName>
      <p><dtml-var zooName></p>
    <dtml-else>
      <form action="<dtml-var URL>" method="GET">
        <input name="zooName">
        <input type="submit" value="What is zooName?">
      </form>
    </dtml-if>
  </dtml-with>

<dtml-var standard_html_footer>
```

In diesem Fall gibt der "with" tag an, dass im `REQUEST`-Namensraum und *nur* im `REQUEST`-Namensraum, nach "zooName" gesucht werden soll.

Das DTML-Client-Objekt

Das Client-Objekt hängt ebenfalls davon ab, ob wir eine Methode oder ein Dokument ausführen. Handelt es sich um ein Dokument, so ist das Client-Objekt immer das Dokument selbst; in anderen Worten, das Client-Objekt eines DTML-Dokumentes ist immer das Dokument selbst.

Eine DTML-Methode dagegen kann verschiedene Client-Objekte haben, je nachdem, auf welche Weise die Methode aufgerufen wird. Nehmen wir z. B. eine Methode, welche den Inhalt eines Ordners ("Folder") anzeigt; in diesem Fall ist das Client-Objekt der Ordner, dessen Inhalt durch die Methode angezeigt wird. Das Client-Objekt kann sich ändern, je nachdem, welcher Folder angezeigt wird: Nehmen wir folgende TML Methode namens *list* im root-folder als Beispiel:

```
<dtml-var standard_html_header>

<ul>
<dtml-in objectValues>
  <li><dtml-var title_or_id></li>
</dtml-in>
</ul>

<dtml-var standard_html_footer>
```

Was diese Methode anzeigt, hängt davon ab, auf welche Art sie ausgeführt wird. Wenden wir diese Methode auf den *Reptiles* Folder mit der URL `http://localhost:8080/Reptiles/list` an, so werden wir etwas ähnliches wie in Abbildung 7-3 erhalten..

Abbildung 7-3 Anwenden der list-Methode auf den "Reptiles" folder.

Wenden wir die list-Methode dagegen auf den *Birds* folder mit der URL *http://localhost:8080/Birds/list* an, werden wir ein anderes Ergebnis erhalten, nämlich nur zwei Einträge in der Liste: *Parrot* und *Raptors*.

Gleiche DTML-Methode, unterschiedliche Ergebnisse. Im ersten Beispiel wurde die list-Methode auf den *Reptiles* folder angewendet; im zweiten Beispiel war das Client-Objekt der *Birds* folder. Bei der Auflösung der *objectValues* Variable fand Zope also einmal die *objectValues* Methode des *Reptiles* Ordners, im zweiten Fall dagegen die *objectValues* Methode des *Birds*-Ordners.

In anderen Worten: Im client-Objekt wird zuerst nach Properties und Methoden-Namen gesucht.

Wie wir in Kapitel 4 "Dynamischer Inhalt mit DTML" gesehen haben, durchsucht Zope die Container eines Objektes, wenn es die Variable nicht im client-Objekt findet. Zope benutzt dazu die Akquisition, um automatisch die Variablen des Containers zu erben, der das client-Objekt enthält. Wenn Zope also auf der Suche nach Variablen die Objekthierarchie hochklettert, sucht es immer zuerst im Client-Objekt und arbeitet sich von dort nach oben.

Das DTML-Request-Objekt

Das Request-Objekt befindet sich ganz unten im DTML-Namensraum-Stack und enthält alle Informationen, die den aktuellen Web-Request betreffen.

Genau wie das Client Objekt nutzt auch das Request-Objekt Akquisition, um in mehreren Namensräumen nach Variablen zu suchen. Im einzelnen sind dies die folgenden Objekte:

1. Die CGI-Umgebung. Das Common Gateway Interface (auch kurz CGI genannt) definiert eine Standardumgebung für dynamische Web-Scripte und Applikationen. Diese Variablen werden von Zope im REQUEST-Namensraum abgelegt.
2. Formular-Daten. Ist das aktuelle Objekt eine Form-Action (d.h. das Ziel eines &lquot;FORM>-Tags), so finden sich alle Eingabefelder des Formulars im REQUEST-Objekt wieder.
3. Cookies. Hat der aktuelle Client Cookie-Daten gesetzt, so finden sich diese im aktuellen REQUEST Objekt.
4. Weitere Variablen. Im REQUEST-Namensraum finden sich eine Menge weiterer nützlicher Variablen, so zum Beispiel die URL des aktuellen Objektes und auch die all seiner Parents.

Der Request-Namensraum ist in Zope sehr nützlich, da er die primäre Art und Weise darstellt, über die Zope mit dem Client (in der Regel der Benutzer eines Webbrowsers Formulardaten, Cookies und andere Informationen übermittelt.

Mehr Informationen über das Request-Objekt finden sich auch in Anhang B.

Wenn all diese Ausführung sehr theoretisch erscheinen, ist es meist hilfreich, auf irgendeiner Seite einmal den folgenden DTML-Schnippsel einzufügen, welcher alle Daten des aktuellen Request-Objektes in lesbarer Form darstellt:

```
<dtml-var standard_html_header>

<dtml-var REQUEST>

<dtml-var standard_html_footer>
```

Die Ausgabe dieser Seite sollte in etwa Abbildung 7-4 entsprechen.

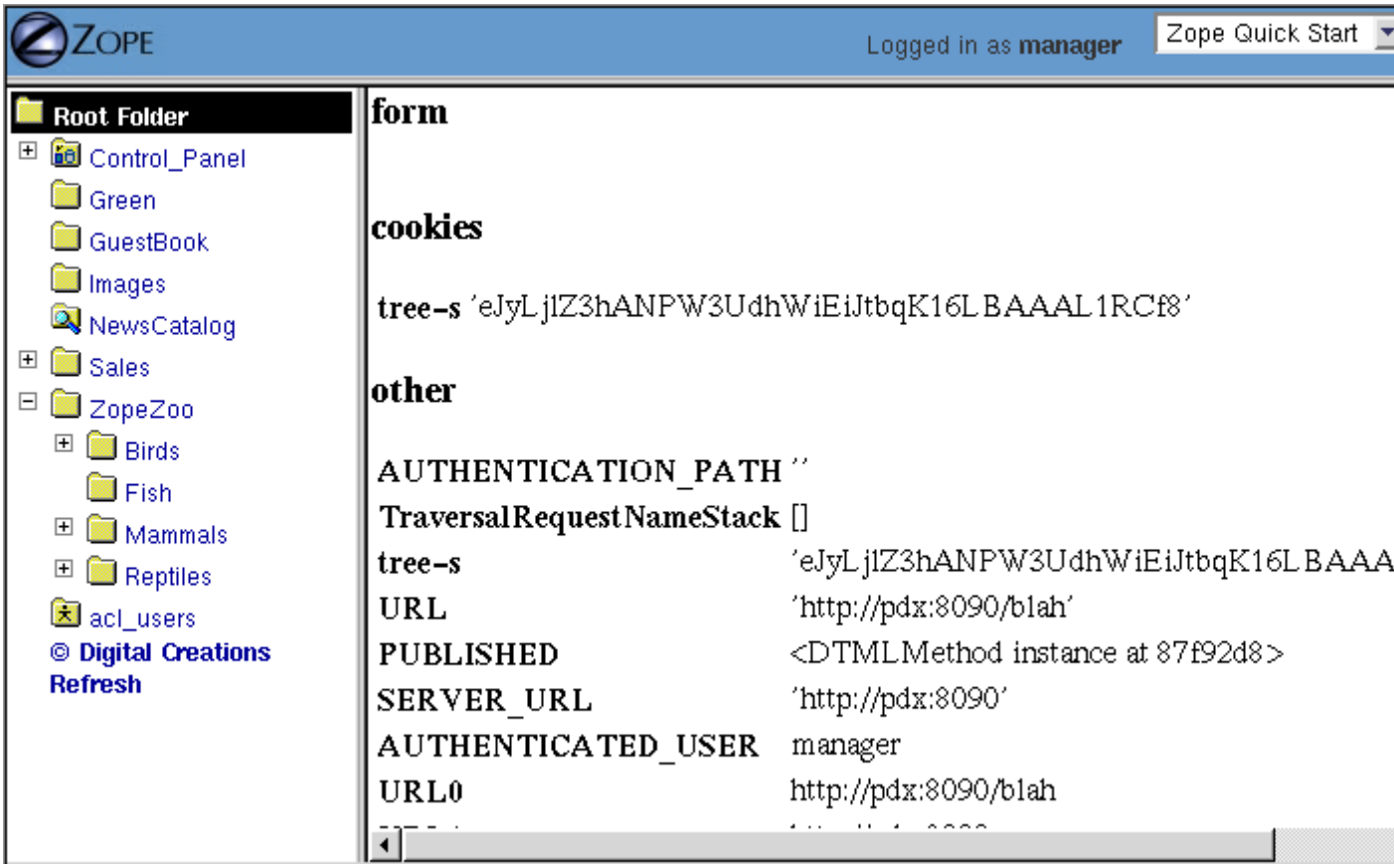


Abbildung 7-4 Das Request-Objekt.

Da das Request-Objekt im Stack nach dem Client-Ojekt folgt, wird bei einer Variablen, die in beiden Namensräumen existiert, das erste Vorkommen dieser Variablen aufgelöst, also im Client-Objekt. Dies kann zu einem Problem werden, und im folgenden wollen wir Wege aufzeigen, wie dieses Problem umgangen werden kann, indem wir mehr Einfluss auf die Art und Weise nehmen, in der DTML Variablennamen auflöst. controlling more directly how DTML looks up variables.

Variablen darstellen ("Rendering")

Wird in Zope eine Variable über den *var* tag eingefügt, so wird diese zunächst im DTML-Namensraum aufgelöst, dann "dargestellt" ("gerendert") und das Resultat eingefügt. "Rendern" bedeutet in diesem Fall die Transformation von einem Objekt in etwas, dass sich in der Ausgabe (in der Regel HTML, welches für einen Browser bestimmt ist) darstellen lässt.

Zope rendert einfache Variablen, indem es sich der Standard-Python-Methode zur Transformation von Objekten in Zeichenketten ("Strings") bedient. Für komplexere Objekte wie DTML- und SQL-Methoden ruft Zope das entsprechende Objekt auf und versucht nicht etwa, es einfach in einen String zu verwandeln. Dieser Mechanismus macht es sehr einfach, DTML-Methoden innerhalb von anderen DTML-Methoden zu verwenden und so komplexe Objekte zu erzeugen.

In der Regel rendert Zope Variablen so, wie "man es erwartet". Erst wenn wir fortgeschrittene Techniken verwenden, wird der eigentliche Rendering-Prozess deutlich. Später in diesem

Kapitel werden wir einige Beispiele betrachten, wie sich das Rendering durch die DTML-Hilfsmethode `getitem` steuern lässt.

Den DTML-Namensraum modifizieren

Jetzt, wo wir wissen, dass der DTML-Namensraum eine Stack-Struktur hat, fragen wir uns eventuell, warum das eigentlich so ist.

Einige DTML tags verändern den DTML-Namensraum während ihrer Ausführung. Ein Tag kann vielleicht ein neues Objekt auf den Stack legen wie z. B. das *in* tag, das *with* tag und auch das *let* tag.

In Namespace-Modifikationen durch DTML-Tags

Während einer *in* Tag Iteration über eine Sequenz schiebt dieses das aktuelle Objekt, über welches gerade iteriert wird, oben auf den Namensraum-Stack:

```
 <dtml-var getId> <!-- This is the id of the
client object -->

        <dtml-in objectValues>

          <dtml-var getId> <!-- this is the id of the current item in the
                               objectValues sequence -->
        </dtml-in>
```

Dieses Konstrukt ist uns bereits oft begegnet. Während der Iteration des *in* tags über eine Sequenz wird jeded Item oben auf den Namensraum-Stack geschoben. Ist der Block beendet, wird das aktuelle Item aus dem Namensraum-Stack entfernt und das nächste auf den Stack geschoben, bis die Sequenz abgearbeitet ist.

Das *With* Tag

Das *with* tag ermöglicht es, ein beliebiges, festgelegtes Objekt oben auf den Namensraum-Stack zu legen. Dies ermöglicht die genaue Definition des Objekts, in dem zuerst nach Variablen gesucht werden soll (für die gesamte Dauer des with-Blocks). Nach Beendigung des Blocks wird das Objekt aus dem Namensraum-Stack entfernt (jedenfalls on der "Spitze").

Denken wir uns einen Ordner, der diverse Methoden und Properties enthält, welche uns interessieren. Wir könnten den Inhalt z. B. mit Pyhon-Methoden wie im folgenden Beispiel ansprechen::

```
        <dtml-var standard_html_header>

        <dtml-var expr="Reptiles.getReptileInfo()">
        <dtml-var expr="Reptiles.reptileHouseMaintainer">

        <dtml-in expr="Reptiles.getReptiles()">
          <dtml-var species>
        </dtml-in>

        <dtml-var standard_html_footer>
```

Wie man sieht, wird die Sache sehr komplex, obwohl wir nur den Inhalt des *Reptiles* Ordners ansprechen wollen.. Mit Hilfe des *with* Tags lässt sich der Code stark vereinfachen:

```
<dtml-var standard_html_header>

        <dtml-with Reptiles>

          <dtml-var getReptileInfo>
          <dtml-var reptileHouseMaintainer>

          <dtml-in getReptiles>
            <dtml-var species>
          </dtml-in>

        </dtml-with>

        <dtml-var standard_html_footer>
```

Ein weiteres Anwendungsgebiet für den *with* Tag ist gegeben, wenn wir das Request-objekt (oder einen Teil dessen) oben auf die Spitze des Namensraum-Stacks befördern möchten. Nehmen wir als Beispiel ein HTML-Formular, welches ein Input-Feld mit dem Namen *id* benutzt. Versuchen wir wir nun, die *id* Variable so aufzulösen:

```
 <dtml-var id>
```

werden wir natürlich nicht die id aus dem Formular erhalten, sondern die id client-Objekt-Id, da diese ganz oben auf dem Namensraum-Stack liegt. Eine Lösung besteht darin, den Namensraum des Request-Objekts nach oben auf den Stack zu befördern (unter Verwendung des *with* tags:

```
 <dtml-with expr="REQUEST.form"> <dtml-var id>
</dtml-with>
```

Dieses Vorgehen stellt sicher, dass wir zuerst die "id" aus dem Formular erhalten. (Anhang B enthät die komplette API-Beschreibung des Request-Objektes)

Das auf dem Stack zu oberst liegende Formular ist jedoch nutzlos, wenn wir es abschicken ohne das *id*-Feld gefüllt zu haben. DTML kann die *id* dort nicht finden. Deshalb sucht es als nächstes im Client-Objekt danach, findet dort eine *id* und liefert sie uns. Sie ist jedoch die *id* des Client-Objekts. Der *with* Tag jedoch hat ein Attribut, welches es uns ermöglicht nur das gewünschte Objekt in den Namensraum einzufügen:

```
        <dtml-with expr="REQUEST.form" only>
          <dtml-if id>
            <dtml-var id>
          <dtml-else>
            <p>The form didn't contain an "id" variable.</p>
          </dtml-if>
        </dtml-with>
```

Durch die Verwendung des *only* Attributs können wir genau angeben, wo nach bestimmten Variablen gesucht werden soll.

Der *Let* Tag

Mit dem *let*-Tag kann ein neuer Namespace auf den Namespace-Stapel geschoben werden. Dieser Namespace wird durch die Tag-Attribute des *let*-Tags definiert:

```
<dtml-let person="'Bob'" relation="'uncle'">
  <p><dtml-var person>'s your <dtml-var relation>.</p>
</dtml-let>
```

Dies ergibt die Anzeige:

```
Bob's your uncle.
```

Der *let*-Tag erfüllt ähnliche Aufgaben wie der *with*-Tag. Der Hauptvorteil des *let*-Tags liegt in der Möglichkeit der Definition mehrerer Variablen, die in einem Block genutzt werden können. Der *let*-Tag erzeugt eine oder mehrerere Variablen mit ihren Werten und schiebt ein Namespace-Objekt, welches diese Variablen enthält auf die Spitze des DTML-Namespace-Stapels. Im Allgemeinen ist der *with*-Tag nützlicher, um bereits existierende Objekte auf den Namespace-Stapel zu schieben, *let*-Tag dagegen ist besser geeignet um neue Variablen für einen Block zu definieren.

Mit hoher Wahrscheinlichkeit ist es besser Python oder Perl zu verwenden, wenn man komplexe DTML-Skripte schreibt, die neue Variablen benötigen. Fortgeschrittenes Skriptprogrammieren wird im Kapitel 10, "Zope-Scripting für Fortgschrittene" beschrieben.

Der DTML-Namespace ist eine komplexe Sache, und seine Komplexität ist im Laufe der Zeit gewachsen. Trotzdem ist es hilfreich zu verstehen, wo die Variablen herkommen. Es ist noch hilfreicher, wenn immer genau angegeben wird, wo nach einer Variablen gesucht wird. Mit dem *with*-Tag und dem *let*-Tag kann der Namespace so kontrolliert werden, daß immer am richtigen Ort nach der gesuchten Variablen geschaut wird.

DTML Namensraumfunktionen

Wie fast alles in Zope ist auch der DTML Namensraum ein Objekt, auf welches über das _ (Unterstrich) Objekt zugegriffen werden kann. Der _ Namensraum wird auch häufig als "der unterstrich Namensraum" bezeichnet.

Der unterstrich Namensraum stellt dem Programmierer für seine Arbeiten viele nützlich Funktionen zur Verfügung. Einige wollen wir im Folgenden betrachten.

Nehmen wir einmal an, ein Name soll dreimal ausgegeben werden. Das kann mit dem *in* Tag realisiert werden. Aber wie bringt man dem *in* Tag bei, dass die Ausgabe genau dreimal erfolgen soll? Man muss einfach nur eine Sequenz aus drei Objekten übergeben:

```
<dtml-var standard_html_header>

<ul>
<dtml-in expr="_.range(3)">
```

```
  <li><dtml-var sequence-item>: Mein Name ist Bob.</li>
</dtml-in>
</ul>

<dtml-var standard_html_footer>
```

Die `_.range(3)` Pythonfunktion wird eine Sequenz bestehend aus den ersten drei Integervariablen 0,1 und 2 zurückgeben. Die *range* Funktion ist *standard Python built-in* und kann genauso wie die meisten standard built-in Funktionen von Python über den _ Namensraum angesprochen werden. Das Funktioniert auch z.B. mit folgenden Funktionen:

`range([start,], stop, [step])`
Gibt eine Liste von Integern von `start` bis `stop` mit `step` Sprüngen zurrück. Der Standardwert für `start` ist 0 und der Standardwert für `step` ist 1. Zum Beispiel gibt

```
'_.range(3,9,2)' folgende Liste zurück: '[3,5,7,9]'.

'len(sequenz)' -- 'len' Gibt die Anzahl der Werte die die Liste
enthält zurück.
```

Viele dieser Ausdrücke stammen aus der Programmiersprache Python, welche eine bestimmte Anzahl von speziellen Funktionen enthält, 'built-ins' genannt. Laut Python-Philosophie, soll es nur eine kleine Anzahl eingebauter Funktionen geben. Dagegen soll es nach der Zope-Philosophie eine große Menge komplexer, eingebauter Funktionen geben.

Mit dem 'Unterstrich-Namensraum' kann die Suche nach einer Variablen gesteuert werden. Dies ist eine der häufigsten Anwendungen dieser Syntax. Sie haben gesehen, daß das 'in-Tag' eine Anzahl spezieller Variablen definiert, so z.B. *sequence-item* und *sequence-key*, die Sie innerhalb einer Schleife nutzen können, um sie anzuzeigen und zu steuern. Wie sieht es wohl aus, wenn Sie eine dieser Variablen innerhalb eines Python-Ausdrucks verwenden wollen?:

```
<dtml-var standard_html_header>

<h1>The squares of the first three integers:</h1>
<ul>
<dtml-in expr="_.range(3)">
  <li>The square of <dtml-var sequence-item> is:
    <dtml-var expr="sequence-item * sequence-item">
  </li>
</dtml-in>
</ul>

<dtml-var standard_html_footer>
```

Try this, does it work? No! Why not? The problem lies in this var tag:

```
<dtml-var expr="sequence-item * sequence-item">
```

Remember, everything inside a Python expression attribute must be a *valid Python expression*. In DTML, *sequence-item* is the name of a variable, but in Python this means "The object *sequence* minus the object *item* ". This is not what you want.

What you really want is to look up the variable *sequence-item*. One way to solve this problem is to use the *in* tag *prefix* attribute. For example:

```
<dtml-var standard_html_header>

<h1>The squares of the first three integers:</h1>
<ul>
<dtml-in prefix="loop" expr="_.range(3)">
  <li>The square of <dtml-var loop_item> is:
    <dtml-var expr="loop_item * loop_item">
  </li>
</dtml-in>
</ul>

<dtml-var standard_html_footer>
```

The *prefix* attribute causes *in* tag variables to be renamed using the specified prefix and underscores, rather than using "sequence" and dashes. So in this example, "sequence-item" becomes "loop_item". See Appendix A for more information on the *prefix* attribute.

Another way to look up the variable *sequence-item* in a DTML expression is to use the *getitem* utility function to explicitly look up a variable:

```
The square of <dtml-var sequence-item> is:
<dtml-var expr="_.getitem('sequence-item') *
                _.getitem('sequence-item')">
```

The *getitem* function takes the name to look up as its first argument. Now, the DTML Method will correctly display the sum of the first three integers. The *getitem* method takes an optional second argument which specifies whether or not to render the variable. Recall that rendering a DTML variable means turning it into a string. By default the *getitem* function does not render a variable.

Here's how to insert a rendered variable named *myDoc*:

```
<dtml-var expr="_.getitem('myDoc', 1)">
```

This example is in some ways rather pointless, since it's the functional equivalent to:

```
<dtml-var myDoc>
```

However, suppose you had a form in which a user got to select which document they wanted to see from a list of choices. Suppose the form had an input named *selectedDoc* which contained the name of the document. You could then display the rendered document like so:

```
<dtml-var expr="_.getitem(selectedDoc, 1)">
```

Notice in the above example that *selectedDoc* is not in quotes. We don't want to insert the variable named *selectedDoc* we want to insert the variable *named by selectedDoc*. For example, the value of *selectedDoc* might be *chapterOne*. Using indirect variable insertion you

can insert the *chapterOne* variable. This way you can insert a variable whose name you don't know when you are authoring the DTML.

If you a python programmer and you begin using the more complex aspects of DTML, consider doing a lot of your work in Python scripts that you call *from* DTML. This is explained more in Chapter 10, "Advanced Zope Scripting". Using Python sidesteps many of the issues in DTML.

DTML Security

Zope can be used by many different kinds of users. For example, the Zope site, Zope.org, has over 11,000 community members at the time of this writing. Each member can log into Zope, add objects and news items, and manage their own personal area.

Because DTML is a scripting language, it is very flexible about working with objects and their properties. If there were no security system that constrained DTML then a user could potentially create malicious or privacy-invading DTML code.

DTML is restricted by standard Zope security settings. So if you don't have permission to access an object by going to its URL you also don't have permission to access it via DTML. You can't use DTML to trick the Zope security system.

For example, suppose you have a DTML Document named *Diary* which is private. Anonymous users can't access your diary via the web. If an anonymous user views DTML that tries to access your diary they will be denied:

```
<dtml-var Diary>
```

DTML verifies that the current user is authorized to access all DTML variables. If the user does not have authorization, than the security system will raise an *Unauthorized* error and the user will be asked to present more privileged authentication credentials.

In Chapter 7, "Users and Security" you read about security rules for executable content. There are ways to tailor the roles of a DTML Document or Method to allow it to access restricted variables regardless of the viewer's roles.

Safe Scripting Limits

DTML will not let you gobble up memory or execute infinite loops and recursions the restrictions on looping and memory are pretty tight, which makes DTML not the right language for complex, expensive programming logic. For example, you cannot create huge lists with the *_.range* utility function. You also have no way to access the filesystem directly in DTML.

Keep in mind however that these safety limits are simple and can be outsmarted by a determined user. It's generally not a good idea to let anyone you don't trust write DTML code on your site.

Advanced DTML Tags

In the rest of this chapter we'll look at the many advanced DTML tags. These tags are summarized in Appendix A. DTML has a set of built-in tags, as documented in this book, which can be counted on to be present in all Zope installations and perform the most common kinds of things. However, it is also possible to add new tags to a Zope installation. Instructions for doing this are provided at the Zope.org web site, along with an interesting set of contributed DTML tags.

This section covers what could be referred to as Zope *miscellaneous* tags. These tags don't really fit into any broad categories except for one group of tags, the *exception handling* DTML tags which are discussed at the end of this chapter.

The *Call* Tag

The *var* tag can call methods, but it also inserts the return value. Using the *call* tag you can call methods without inserting their return value into the output. This is useful if you are more interested in the effect of calling a method rather than its return value.

For example, when you want to change the value of a property, *animalName*, you are more interested in the effect of calling the *manage_changeProperties* method than the return value the method gives you. Here's an example:

```
      <dtml-if expr="REQUEST.has_key('animalName')">
        <dtml-call
expr="manage_changeProperties(animalName=REQUEST['animalName'])">
        <h1>The property 'animalName' has changed</h1>
      <dtml-else>
        <h1>No properties were changed</h1>
      </dtml-if>
```

In this example, the page will change a property depending on whether a certain name exists. The result of the *manage_changeProperties* method is not important and does not need to be shown to the user.

Another common usage of the *call* tag is calling methods that affect client behavior, like the `RESPONSE.redirect` method. In this example, you make the client redirect to a different page, to change the page that gets redirected, change the value for the "target" variable defined in the *let* tag:

```
      <dtml-var standard_html_header>

      <dtml-let target="'http://example.com/new_location.html'">

        <h1>This page has moved, you will now be redirected to the
        correct location.  If your browser does not redirect, click <a
        href="<dtml-var target>"><dtml-var target></a>.</h1>

        <dtml-call expr="RESPONSE.redirect(target)">

      </dtml-let>

      <dtml-var standard_html_footer>
```

In short, the *call* tag works exactly like the *var* tag with the exception that it doesn't insert the results of calling the variable.

The *Comment* Tag

DTML can be documented with comments using the *comment* tag:

```
<dtml-var standard_html_header>

<dtml-comment>

  This is a DTML comment and will be removed from the DTML code
  before it is returned to the client.  This is useful for
  documenting DTML code.  Unlike HTML comments, DTML comments
  are NEVER sent to the client.

</dtml-comment>

<!--

  This is an HTML comment, this is NOT DTML and will be treated
  as HTML and like any other HTML code will get sent to the
  client.  Although it is customary for an HTML browser to hide
  these comments from the end user, they still get sent to the
  client and can be easily seen by 'Viewing the Source' of a
  document.

-->

<dtml-var standard_html_footer>
```

The *comment* block is removed from DTML output.

In addition to documenting DTML you can use the *comment* tag to temporarily comment out other DTML tags. Later you can remove the *comment* tags to re-enable the DTML.

The *Tree* Tag

The *tree* tag lets you easily build dynamic trees in HTML to display hierarchical data. A *tree* is a graphical representation of data that starts with a "root" object that has objects underneath it often referred to as "branches". Branches can have their own branches, just like a real tree. This concept should be familiar to anyone who has used a file manager program like Microsoft Windows Explorer to navigate a file system. And, in fact, the left hand "navigation" view of the Zope management interface is created using the tree tag.

For example here's a tree that represents a collection of folders and sub-folders.

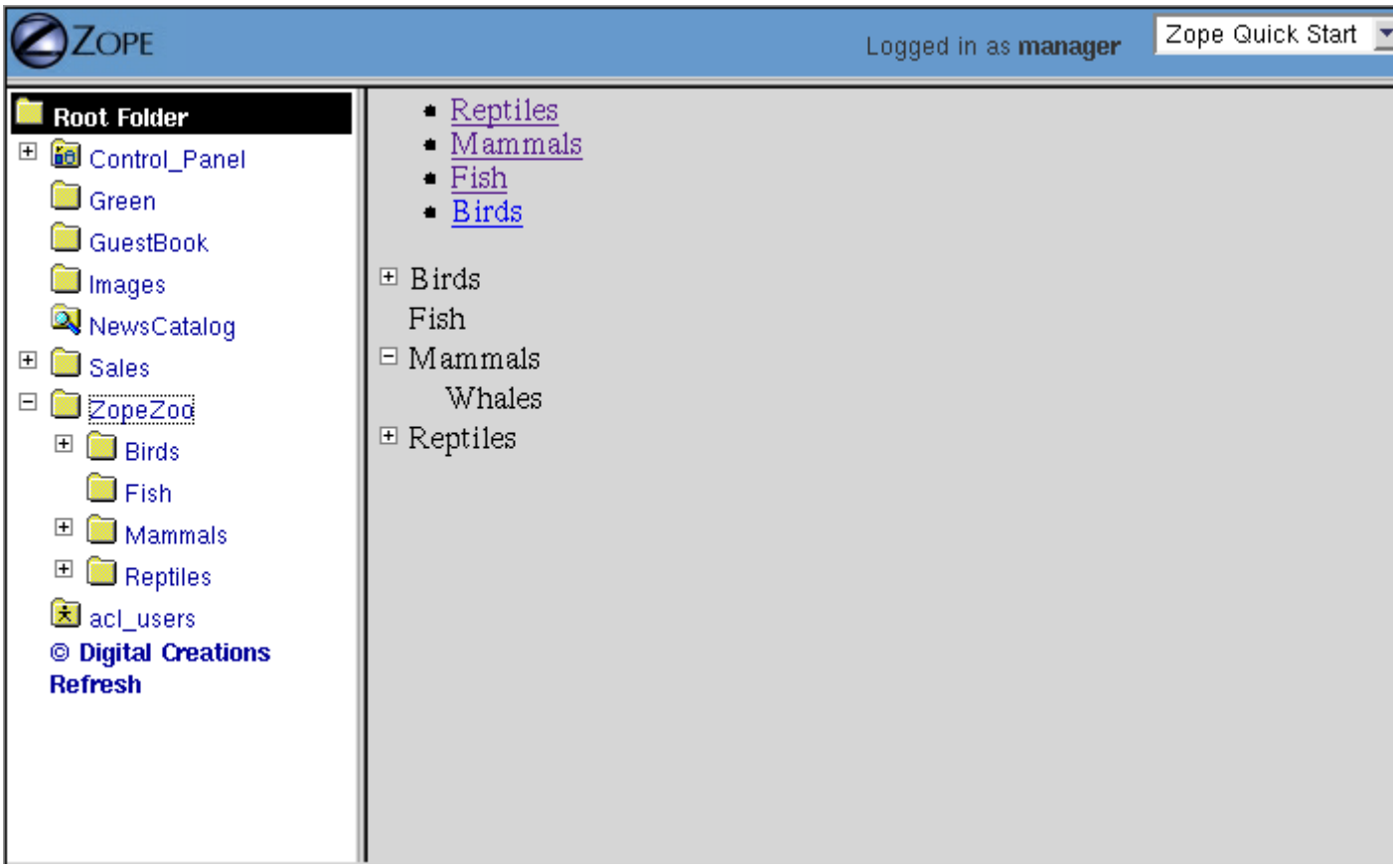


Abbildung 7-5 HTML tree generated by the tree tag.

Here's the DTML that generated this tree display:

```
<dtml-var standard_html_header>

<dtml-tree>

  <dtml-var getId>

</dtml-tree>

<dtml-var standard_html_footer>
```

The *tree* tag queries objects to find their sub-objects and takes care of displaying the results as a tree. The *tree* tag block works as a template to display nodes of the tree.

Now, since the basic protocol of the web, HTTP, is stateless, you need to somehow remember what state the tree is in every time you look at a page. To do this, Zope stores the state of the tree in a *cookie*. Because this tree state is stored in a cookie, only one tree can appear on a web page at a time, otherwise they will confusingly use the same cookie.

You can tailor the behavior of the *tree* tag quite a bit with *tree* tag attributes and special variables. Here is a sampling of *tree* tag attributes.

branches

The name of the method used to find sub-objects. This defaults to *tpValues*, which is a method defined by a number of standard Zope objects.

leaves
: The name of a method used to display objects that do not have sub-object branches.

nowrap
: Either 0 or 1. If 0, then branch text will wrap to fit in available space, otherwise, text may be truncated. The default value is 0.

sort
: Sort branches before text insertion is performed. The attribute value is the name of the attribute that items should be sorted on.

assume_children
: Either 0 or 1. If 1, then all objects are assumed to have sub-objects, and will therefore always have a plus sign in front of them when they are collapsed. Only when an item is expanded will sub-objects be looked for. This could be a good option when the retrieval of sub-objects is a costly process. The defalt value is 0.

single
: Either 0 or 1. If 1, then only one branch of the tree can be expanded. Any expanded branches will collapse when a new branch is expanded. The default value is 0.

skip_unauthorized
: Either 0 or 1. If 1, then no errors will be raised trying to display sub-objects for which the user does not have sufficient access. The protected sub-objects are not displayed. The default value is 0.

Suppose you want to use the *tree* tag to create a dynamic site map. You don't want every page to show up in the site map. Let's say that you put a property on folders and documents that you want to show up in the site map.

Let's first define a Script with the id of *publicObjects* that returns public objects:

```
## Script (Python) "publicObjects"
##
"""
Returns sub-folders and DTML documents that have a
true 'siteMap' property.
"""
results=[]
for object in context.objectValues(['Folder', 'DTML Document']):
    if object.hasProperty('siteMap') and object.siteMap:
        results.append(object)
return results
```

Now we can create a DTML Method that uses the *tree* tag and our Scripts to draw a site map:

```
<dtml-var standard_html_header>

<h1>Site Map</h1>

<p><a href="&dtml-URL0;?expand_all=1">Expand All</a> |
   <a href="&dtml-URL0;?collapse_all=1">Collapse All</a>
</p>

<dtml-tree branches="publicObjects" skip_unauthorized="1">
  <a href="&dtml-absolute_url;"><dtml-var title_or_id></a>
```

```
</dtml-tree>

<dtml-var standard_html_footer>
```

This DTML Method draws a link to all public resources and displays them in a tree. Here's what the resulting site map looks like.

Abbildung 7-6 Dynamic site map using the tree tag.

For a summary of the *tree* tag arguments and special variables see Appendix A.

Der *return* Tag

Meist wird DTML verwendet, um eine Textausgabe zu erzeugen. Es ist jedoch genauso möglich DTML beliebige Python Werte erzeugen zu lassen - genauso wie es z.B. Python Scripts machen.

Hier ein Beispiel:

```
<p>This text is ignored.</p>

<dtml-return expr="42">
```

Diese DTML Methode liefert die Zahl 42 zurück.

Ein Nebeneffekt des *return* Tag ist, dass die Ausführung des DTML Codes nach durch diesen Tag beendet wird - schliesslich ist der Rückgabewert jetzt ja gefunden.

Bei der Benutzung dieses Tags sollten Sie sich jedoch fragen, ob Sie ...

If you find yourself using the *return* tag, you almost certainly should be using a Script instead. The *return* tag was developed before Scripts, and is largely useless now that you can easily write scripts in Python and Perl.

The *Sendmail* Tag

The *sendmail* tag formats and sends a mail messages. You can use the *sendmail* tag to connect to an existing Mail Host, or you can manually specify your SMTP host.

Here's an example of how to send an email message with the *sendmail* tag:

```
<dtml-sendmail>
To: <dtml-var recipient>
Subject: Make Money Fast!!!!

Take advantage of our exciting offer now! Using our exclusive method
you can build unimaginable wealth very quickly. Act now!
</dtml-sendmail>
```

Notice that there is an extra blank line separating the mail headers from the body of the message.

A common use of the *sendmail* tag is to send an email message generated by a feedback form. The *sendmail* tag can contain any DTML tags you wish, so it's easy to tailor your message with form data.

The *Mime* Tag

The *mime* tag allows you to format data using MIME (Multipurpose Internet Mail Extensions). MIME is an Internet standard for encoding data in email message. Using the *mime* tag you can use Zope to send emails with attachments.

Suppose you'd like to upload your resume to Zope and then have Zope email this file to a list of potential employers.

Here's the upload form:

```
<dtml-var standard_html_header>

<p>Send you resume to potential employers</p>

<form method=post action="sendresume" ENCTYPE="multipart/form-data">
<p>Resume file: <input type="file" name="resume_file"></p>
<p>Send to:</p>
<p>
<input type="checkbox" name="send_to:list" value="jobs@yahoo.com">
```

```
        Yahoo<br>

      <input type="checkbox" name="send_to:list"
value="jobs@microsoft.com">
        Microsoft<br>

      <input type="checkbox" name="send_to:list"
value="jobs@mcdonalds.com">
        McDonalds</p>

      <input type=submit value="Send Resume">
      </form>

      <dtml-var standard_html_footer>
```

Create another DTML Method called *sendresume* to process the form and send the resume file:

```
      <dtml-var standard_html_header>

      <dtml-if send_to>

        <dtml-in send_to>

          <dtml-sendmail smtphost="my.mailserver.com">
          To: <dtml-var sequence-item>
          Subject: Resume
          <dtml-mime type=text/plain encode=7bit>

          Hi, please take a look at my resume.

          <dtml-boundary type=application/octet-stream
disposition=attachment
          encode=base64><dtml-var expr="resume_file.read()"></dtml-mime>
          </dtml-sendmail>

        </dtml-in>

        <p>Your resume was sent.</p>

      <dtml-else>

        <p>You didn't select any recipients.</p>

      </dtml-if>

      <dtml-var standard_html_footer>
```

This method iterates over the *sendto* variable and sends one email for each item.

Notice that there is no blank line between the `To:` header and the starting *mime* tag. If a blank line is inserted between them then the message will not be interpreted as a *multipart* message by the receiving mail reader.

Also notice that there is no newline between the *boundary* tag and the *var* tag, or the end of the *var* tag and the closing *mime* tag. This is important, if you break the tags up with newlines then they will be encoded and included in the MIME part, which is probably not what you're after.

As per the MIME spec, *mime* tags may be nested within *mime* tags arbitrarily.

Der *Unless* Tag

Der *unless* Tag führt den eingeschlossenen Code aus, wenn eine angegebenen Bedingung *nicht* zutrifft. Er ist damit das Gegenteil des *if* Tags. Folgender DTML Code:

```
<dtml-if expr="not butter">
   Es ist wirklich keine Butter!
 </dtml-if>
```

bewirkt das Gleiche wie:

```
<dtml-unless expr="butter">
  Es ist wirklich keine Butter!
</dtml-unless>
```

Was ist dann überhaupt der Zweck des *unless* Tag? Er dient einfach nur der Bequemlichkeit. Auch ist er weniger ausdrucksstark als der *if* Tag, denn ihm kann kein *else* oder *elif* Tag folgen.

Genauso wie der *if* Tag, bewirkt die Verwendung des *unless* Tag mit dem *name* Attribut die Überprüfung auf Existenz:

```
<dtml-unless name="the_easter_bunny">
       The Easter Bunny does not exist or is not true.
     </dtml-unless>
```

Überprüft, ob *the_easter_bunny* existiert und ob es wahr ist. Folgendes Beispiel hingegen benutzt das *expr* Attribut und prüft daher ausschliesslich, ob das Objekt wahr ist:

```
<dtml-unless expr="the_easter_bunny">
       The Easter Bunny is not true.
     </dtml-unless>
```

Daher wird eine Ausnahme gemeldet, wenn *the_easter_bunny* nicht existiert.

Jeder *unless* Tags kann durch einen äquivalenten *if* Tag ersetzt werden, der dann die inverse Bedingung angibt. Seine Verwendung ist daher optional und eine Frage des Stils.

Batch Verarbeitung mit dem *In* Tag

Häufig müssen Nutzern Listen von Einträgen wie z.B. Suchresultaten oder Ergebnissen einer Datenbankabfrage präsentiert werden. Werden diese Listen zu umfangreich, ist es nicht praktikabel diese alle auf einer Seite aufzulisten. Stattdessen ist eine Aufteilung der Liste in mehrere kleinere Listen wünschenswert. So wird dies z.B. von Suchmaschinen gehandhabt, um dem Nutzer eine Vielzahl von Suchresultaten etappenweise anzeigen können.

Dieser Prozess wird im Englischen mit "batching" bezeichnet. In Ermangelung einer angemessenen Übersetzung wird dieser Begriff übernommen.

Batching hat folgende Vorteile:

- Die geringere Größe eines Batches verringert die Ladezeit eines Dokumentes.
- Zope benötigt weniger Speicher
- Navigationshilfen erleichtern die Orientierung innerhalb großer Listen.

Der *in* Tag stellt mehrere Variablen zur Verfügung, um Batching zu ermöglichen. Schauen wir uns ein Beispiel an, dass zeigt, wie 100 Einträge in Batches von 10 am Stück angezeigt werden:

```
<dtml-var standard_html_header>

  <dtml-in expr="_.range(100)" size=10 start=query_start>

    <dtml-if sequence-start>

      <dtml-if previous-sequence>
        <a href="<dtml-var URL><dtml-var sequence-query
           >query_start=<dtml-var previous-sequence-start-number>">
           (Vorherige <dtml-var previous-sequence-size> Resultate)
        </a>
      </dtml-if>

      <h1>Dies wird über dem Batch angezeigt:</h1>
      <ul>

    </dtml-if>

      <li>Iteration number: <dtml-var sequence-item></li>

    <dtml-if sequence-end>

      </ul>
      <h4>Dies wird unter dem Batch angezeigt.</h4>

      <dtml-if next-sequence>
         <a href="<dtml-var URL><dtml-var sequence-query
            >query_start=<dtml-var
            next-sequence-start-number>">
         (Nächste <dtml-var next-sequence-size> Resultate)
         </a>

      </dtml-if>

    </dtml-if>

  </dtml-in>

<dtml-var standard_html_footer>
```

Was genau geht hier vor? Zuerst ist da ein *in* Tag, der über 100 Nummern iteriert, die von der *range* Funktion erzeugt werden. Das *size* Attribute lässt den *in* Tag nur 10 Nummern auf einmal anzeigen. Das *start* Attribute gibt an, an welcher Stelle innerhalb der Liste der Batch beginnt.

Der *in* Tags hat zwei *if* Tags als direkte Kindelemente. Der erste überprüft die Variable `sequence-start`. Diese ist nur bei dem ersten Durchlauf des *in* Blocks wahr. Daher wird der Inhalt des Tags nur einmal am Anfang der Schleife ausgeführt. Der zweite *if* Tag überprüft die Variable `sequence-end`. Diese wiederum ist nur beim letzten Durchlauf des *in* Blocks wahr. Entsprechend wird dieser *if* Block nur einmal am Ende der Schleife ausgeführt. Nur der Teil zwischen den beiden *if* Tags wird bei jedem Schleifendurchlauf ausgeführt.

Innerhalb von beiden *if* Tags befindet sich ein weiterer *if* Tag der die Variablen `previous-sequence` bzw. `next-sequence` überprüft. Diese Variablen sind wahr, wenn der aktuelle Batch einen Vorgänger bzw. einen Folgebatch hat. `previous-sequence` ist also für alle Batches bis auf den ersten wahr, und `next-sequence` für alle bis auf den letzten. Wenn wahr, so wird ein Link zum Vorgänger- oder Nachfolgebatch erzeugt.

Diese Links referenzieren das eigene DTML Dokument, nur der *query_start* Query Parameter wird auf den Wert gesetzt, an dem der verlinkte Batch starten soll. Um ein Gefühl dafür zu bekommen, wie das funktioniert, sollte man diese Art der Navigation einmal benutzen und darauf achten, wie sich die URL dieser Navigationslinks in Abhängigkeit von der Position des Batches innerhalb der Gesamtliste ändert.

Schließlich werden in unserem Beispiel noch einige Kennzahlen der Batches mit Hilfe der `next-sequence-size` und `previous-sequence-size` Variablen ausgegeben. Diese ganze Logik gibt schließlich folgenden HTML Code aus:

```
<html><head><title>Zope</title></head><body bgcolor="#FFFFFF">

  <h1>Dies wird über dem Batch angezeigt:</h1>
  <ul>
    <li>Iteration number: 0</li>
    <li>Iteration number: 1</li>
    <li>Iteration number: 2</li>
    <li>Iteration number: 3</li>
    <li>Iteration number: 4</li>
    <li>Iteration number: 5</li>
    <li>Iteration number: 6</li>
    <li>Iteration number: 7</li>
    <li>Iteration number: 8</li>
    <li>Iteration number: 9</li>
  </ul>
  <h4>Dies wird unter dem Batch angezeigt.</h4>

     <a href="http://pdx:8090/batch?query_start=11">
       (Nächste 10 Resultate)
     </a>

</body></html>
```

Batch Verarbeitung kann sehr komplex sein. Eine gute Methode mit Batches zu arbeiten, ist die Benutzung des *Searchable* Interface Objektes, um eine Batch Ausgabe zu erhalten. Dieses DTML kann dann für die eigenen Bedürfnisse angepasst werden. Dies wird näher in Kapitel 11 "Inhalt durchsuchen und kategorisieren" beschrieben.

Tags für die Behandlung von Ausnahmen

Zope hat ausgefeilte Mechanismen zur Behandlung von Ausnahmen. Auf diese kann von DTML aus mit den *raise* und *try* Tags zugegriffen werden. Mehr über Ausnahmen und wie sie ausgelöst und behandelt werden, erfahren Sie in einführenden Büchern über Python oder auch online im Python Tutorial.

Der *Raise* Tag

Mit dem *raise* Tag können Ausnahmen ausgelöst werden, um Fehler zu signalisieren. Es könnte zum Beispiel mit dem *if* Tag auf einen Fehler geprüft werden, um diesen dann mit dem *raise* Tag bekannt zu machen.

Das *type* Attribut des *raise* Tags spezifiziert die Art des Fehlers. Dies ist ein kurzer beschreibender Name. Es gibt einige standardisierte Fehlernamen, z.B. *Unauthorized* und *Redirect*, welche eine HTTP Antwort mit einer Fehlernummer verursachen. *Unauthorized* Fehler führen dazu, dass der Browser einen Login Prompt anzeigt. Auch können HTTP Fehlernummern im *type* Attribut verwendet werden, die Zope in der HTTP Antwort verwendet. Zum Beispiel:

```
<dtml-raise type="404">Nicht gefunden</dtml-raise>
```

Dies verursacht das Setzen des HTTP 404 (Not Found) Fehler in der HTTP Anwort an den Browser.

Der *raise* Tag ist ein Block Tag. Der eingeschlossene Text bestimmt die Art der Fehlermeldung. Enthält er HTML Elemente, so wird er als Fehlermeldung gesendet, andererseits wird eine generische Fehlerseite erzeugt innerhalb derer der Text als Fehlerwert angezeigt wird.

Hier ein Beispiel für den *raise* Tag:

```
<dtml-if expr="kontostand >= betrag">

  <dtml-call expr="belaste(konto, betrag)">

  <p><dtml-var betrag> Euro wurden von Ihrem Konto
       <dtml-var account> abgebucht.</p>

<dtml-else>

  <dtml-raise type="Ueberziehung">

    <p>Sie haben nicht genügend Geld auf Ihrem Konto <dtml-
account>.</p>

  </dtml-raise>

</dtml-if>
```

Ein wichtiger Seiteneffekt beim Auslösen von Ausnahmen ist der Rollback der aktuellen Transaktion. Das heißt, dass alle Änderungen, die durch die aktuelle Anfrage ausgelöst wurden, zurückgenommen werden. Neben der Möglichkeit dem Nutzer eine Fehlermeldung anzuzeigen, erlauben es Ausnahmen also, beim Auftreten von Problemen evtl. bereits vorgenommene Änderungen rückgängig zu machen.

Der *Try* Tag

Wird eine Ausnahme entweder manuell mit dem *raise* Tag oder intern durch Zope ausgelöst, so kann diese mit dem *try* Tag abgefangen werden.

Ausnahmen sind unerwartete Fehler, die während der Ausführung von DTML Code entstehen. Wird eine Ausnahme entdeckt, stoppt Zope die weitere Ausführung. Sehen Sie sich dazu folgendes Beispiel an:

```
        Kosten pro Stück:
             <dtml-var expr="_.float(total_cost/total_units)" fmt=dollars-
and-cents>
```

Dieser DTML Code funktioniert tadellos solange *total_units* nicht Null ist. Ist dies jedoch der Fall, zeigt eine *ZeroDivisionError* Ausnahme diese ungültige Operation an. Anstelle der erwarteten Ausgabe wird jetzt eine Fehlermeldung angezeigt.

Der *try* Tag kann benutzt werden, um dies zu verhindern. Mit ihm kann die Fehlerbehandlung in die eigene Hand genommen werden.

Der *try* Tag hat zwei Funktionen. Erstens kann er die Ausgabe einer Fehlermeldung verhindern. Im Falle einer Ausnahme kann eingeschlossener Code auf den Fehler reagieren. Zweitens verhindert er den Abbruch der Ausführung des DTML Codes.

Innerhalb des *try* Tags gibt es einen oder mehrere *except* Tags die auf jeweils unterschiedliche Ausnahmen reagieren. Wird eine Ausnahme ausgelöst, so wird jeder *except* Tag auf seine Zuständigkeit für die Ausnahme hin überprüft. Der erste gefundene *except* Tag behandelt die Ausnahme. Wird in einem *except* Tag kein spezifischer Ausnahmetyp angegeben, so ist er für alle Ausnahmen zuständig.

Folgendermaßen müsste der *try* Tag verwendet werden, um die Ausgabe einer Fehlermeldung im vorangehenden Beispiel zu vermeiden:

```
        <dtml-try>
              Kosten pro Stück:
                  <dtml-var expr="_.float(total_cost/total_units)"
fmt=dollars-and-cents>

        <dtml-except ZeroDivisionError>

              Kosten pro Stück: N/A

        </dtml-try>
```

Wird ein *ZeroDivisionError* ausgelöst, geht die Kontrolle auf den *except* Tag über, und "Kosten pro Stück: N/A" wird ausgegeben. Nach der Ausführung des *except* Blocks wird die Ausführung des DTML Codes nach dem *try* Block wieder aufgenommen.

DTML's *except* Tags funktionieren mit Python's klassenbasierten Ausnahmen. Wie in Python werden dabei auch Unterklassen der spezifizierten Ausnahmeklassen behandelt. Wird z.B. *ArithmeticError* in einem *except* Tag angegeben, so werden von diesem sowohl Ausnahmen der Klasse *ArithmeticError* als auch Ausnahmen von dessen Unterklassen, z.B. auch

ZeroDivisionError behandelt. Eine Auflistung aller Ausnahmen der Python Standardbibliothek und ihrer Vererbungshierarchie findet sich z.B. in der Python Library Reference. Auch kann ein *except* Tag mehrere Ausnahmen spezifizieren und behandeln.

Innerhalb eines *except* Tags kann auf Informationen über die behandelte Ausnahme über verschiedene durch Zope bereitgestellte Variablen zugegriffen werden.

error_type
: Der Typ der behandelten Ausnahme.

error_value
: Der Wert der Ausnahme.

error_tb
: Der Traceback der behandelten Ausnahem.

Diese Variablen können benutzt werden, um Fehlermeldungen für Nutzer zu erzeugen oder in Abhängigkeit von ihrem Wert - z.B. dem Fehlertyp - entweder dem Webmaster eine Email zu senden oder den Fehler zu loggen.

Der optionale *Else* Block

Im Anschluss an den *try* Tag kann ein *else* Block verwendet werden, der ausgeführt wird, wenn keine Ausnahme ausgelöst wurde. Hier ein Beispiel, das zeigt, wieso das sinnvoll sein kann:

```
<dtml-try>

  <dtml-call fütterKrokodile>

<dtml-except NichtGenugFutter FalschesFutter>

  <p>Wir haben nicht genug Futter!</p>

<dtml-except NotHungry>

  <p>Die Krokos sind noch nicht hungrig.</p>

<dtml-except>

  <p>Fütterungsversuch nicht erfolgreich.<p>
  <p>Error type: <dtml-var error_type></p>
  <p>Error value: <dtml-var error_value></p>

<dtml-else>

  <p>Die Krokodile sind gefüttert worden!</p>

</dtml-try>
```

Der erste *except* Block der für den auftretenden Fehler zuständig ist, wird ausgeführt. Wenn ein *except* Block keinen Namen hat, kann er alle Ausnahmen behandeln. Der optionale *else* Block wird ausgeführt, wenn keine Ausnahme innerhalb des *try* Blocks auftritt. Ausnahmen innerhalb des *else* Block werden nicht von den vorangehenden *except* Blocks behandelt.

Der optionale *Finally* Block

Der *try* Tag kann auch noch auf eine etwas andere Weise verwendet werden. Anstatt Ausnahmen zu behandeln, kann er benutzt werden, um nach ihrem Auftreten &aufzuräumen&.

Der *finally* Tag innerhalb des *try* Tags enthält dazu Code, der immer ausgeführt wird, selbst wenn eine unbehandelte Ausnahme auftritt.

Der *finally* Block ist dann nützlich, wenn sichergestellt werden muss, dass Code ausgeführt wird, egal ob eine Ausnahme auftritt oder nicht. Selbst wenn vorher ein *return* Tag wurde wird der *finally* Block ausgeführt. In diesem Fall wird jedoch jegliche Ausgabe des *finally* Blocks verworfen. Hier ein Beispiel dafür wann der *finally* Tag nützlich sein könnte:

```
          <dtml-call sperreRessource>
          <dtml-try>
              <dtml-call verwendeRessource>
          <dtml-finally>
              <!-- dies wird immer ausgeführt, mit oder ohne Auftreten
einer Ausnahme -->
              <dtml-call entsperreRessource>
          </dtml-try>
```

In diesem Beispiel wird die Sperrung einer Ressource erlangt, mit der gearbeitet werden soll. Wird eine Ausnahme bei ihrer darauffolgenden Verwendung ausgelöst, wird diese zwar nicht behandelt, aber es wird sichergestellt, dass die Sperrung wieder aufgehoben wird. Auch wenn keine Ausnahme auftritt, wird der *finally* Block ausgeführt und die Sperrung nach Abarbeitung des *try* Blocks aufgehoben.

Der *finally* Tag wird in Zope selten verwendet, da solch komplizierte Programmabläufe besser in Skripten realisiert werden.

Zusammenfassung

DTML stellt umfangreiche Funktionalität zur Entwicklung von Webanwendungen zur Verfügung. Wir haben uns in diesem Kapitel einige der mächtigeren DTML Tags und ihre Optionen angeschaut. Eine vollständigere Referenz zu DTML findet sich in Anhang A.

Das nächste Kapitel zeigt Ihnen, wie Sie Experte für Page Templates werden. Während DTML ein ausdrucksstarkes Allzweckwerkzeug ist, sind Page Templates ein eleganterer Ansatz zur Erzeugung von HTML und anderen Markupsprachen.

Zopebuch: Inhaltsverzeichnis

Kapitel 9: Zope-Seitenvorlagen für Fortgeschrittene

In Kapitel 5, "Zope-Seitenvorlagen verwenden", haben Sie die grundlegenden Merkmale von Seitenvorlagen kennen gelernt. Dieses Kapitel behandelt komplexere Verfahren, darunter auch neue Arten von Ausdrücken und Makros.

TAL für Fortgeschrittene

Sie haben bereits einiges über TAL-Anweisungen (Template Attribute Language: Auszeichnungssprache für Seitenvorlagen) erfahren. In diesem Abschnitt gehen wir nun detailliert auf alle TAL-Anweisungen und ihre Optionen ein. Eine Zusammenstellung dieses Materials finden Sie in Anhang C, "Zope-Page-Templates-Referenz".

Inhaltseingabe

Nachdem Sie in Kapitel 5, "Zope-Seitenvorlagen verwenden", bereits gesehen haben, wie `tal:content` und `tal:replace` funktionieren, werden Sie in diesem Abschnitt einige darauf aufbauende Tricks zum Einfügen von Inhalt kennen lernen.

Strukturen einbetten

Normalerweise setzen die Anweisungen `tal:replace` und `tal:content` HTML-Tags und -Entitäten im einzufügenden Text außer Kraft, d. h. sie konvertieren HTML-Auszeichnungen in Text. So wandeln sie zum Beispiel < in `<` um. Möchten Sie diese Umwandlung vermeiden, den Text also als Teil der HTML-Struktur eingeben, so stellen Sie dem Ausdruck das Schlüsselwort `structure` voran. Beispiel:

```
<p replace="structure here/geschichte">
    die <b>Geschichte</b>
</p>
```

Diese Funktion ist nützlich, wenn Sie ein HTML- oder XML-Fragment einfügen, das in einer Eigenschaft gespeichert ist oder von einem anderen Zope-Objekt erzeugt wurde. Nehmen wir zum Beispiel an, Sie haben Nachrichtenmeldungen, die einfache HTML-Auszeichnungen, z. B. für Fett- und Kursivschrift, enthalten, und diese Auszeichnungen möchten Sie auch beibehalten, wenn Sie die Meldungen in eine "Top-News"-Seite einfügen. Dafür schreiben Sie:

```
<p tal:repeat="newsItem here/topNews"
        tal:content="structure newsItem">
    Eine Meldung mit <code>HTML</code>-Auszeichnungen.
</p>
```

Dieser Code fügt die HTML-Auszeichnungen der Nachrichtenmeldungen in eine Reihe von Absätzen ein.

Blindelemente

Seitenelemente, die zwar in der Vorlage, nicht aber im erzeugten Text sichtbar sind, fügen Sie mit der integrierten Variable `nothing` ein. Hierzu ein Beispiel:

```
<tr tal:replace="nothing">
    <td>10213</td><td>Beispielelement</td><td>$15.34</td>
</tr>
```

Dieses Verfahren ist insbesondere dann nützlich, wenn Sie Seitenbereiche ausfüllen möchten, die mit dynamischem Inhalt gefüllt werden sollen. So hat zum Beispiel eine Tabelle, die normalerweise zehn Zeilen enthält, in der Seitenvorlage nur eine Zeile. Fügt man nun aber 9 Blindzeilen ein, so hat das Layout der Vorlage weit größere Ähnlichkeit mit dem Endergebnis.

Nicht immer müssen Sie auf `tal:replace="nothing"` zurückgreifen, um Blindinhalt in Ihre Seitenvorlagen einzufügen. So haben Sie zum Beispiel bereits gesehen, dass alles innerhalb der Elemente `tal:content` und `tal:replace` normalerweise entfernt wird, wenn die Seitenvorlage umgesetzt wird. Bei diesen Elementen sind also keine zusätzlichen Verfahren nötig, um den Blindinhalt zu entfernen.

Standardinhalt

Sie können den Inhalt eines Elements bestehen lassen, indem Sie den Ausdruck `default` zusammen mit `tal:content` oder `tal:replace` verwenden. Hierzu ein Beispiel:

```
<p tal:content="default">Spam<p>
```

Dieser Code wird folgendermaßen umgesetzt:

```
<p>Spam</p>
```

Meist möchte man Standardinhalt nicht generell einfügen, sondern nur, wenn bestimmte Bedingungen erfüllt sind. Ein Beispiel:

```
<p tal:content="python:here.getEssen() or
default">Dosenfleisch</p>
```

Hinweis: Python-Ausdrücke werden weiter unten in diesem Kapitel erklärt. Falls die Methode `holeEssen` einen wahren Wert zurückgibt, wird ihr Ergebnis in den Absatz eingefügt; ansonsten ist das Ergebnis Dosenfleisch zum Abendessen.

Wiederholung

Die meisten Dinge, die Sie mit der Anweisung `tal:repeat` machen können, haben Sie bereits in Kapitel 5, "Zope-Seitenvorlagen verwenden", kennen gelernt. Dieser Abschnitt behandelt einige weiterführende Funktionen der Anweisung `tal:repeat`.

Wiederholungsvariablen

Die Wiederholungsvariablen gehören zu denjenigen Themen, die eventuell einer ausführlicheren Erläuterung bedürfen. Sie stellen Informationen über die aktuelle Wiederholung bereit. Für `repeat`-Variablen stehen die folgenden Attribute zur Verfügung:

- *index* - Nummer der Wiederholung, beginnend bei null
- *number* - Nummer der Wiederholung, beginnend bei eins
- *even* - wahr für geradzahlig indizierte Wiederholungen (0, 2, 4, ...)
- *odd* - wahr für ungeradzahlig indizierte Wiederholungen (1, 3, 5, ...)
- *start* - wahr für die Anfangswiederholung (Index 0)
- *end* - wahr für die abschließende (letzte) Wiederholung
- *length* - Länge der Folge, also die Gesamtzahl der Wiederholungen
- *letter* - Anzahl der Wiederholungen mit Kleinbuchstaben: "a" - "z", "aa" - "az", "ba" - "bz", ..., "za" - "zz", "aaa" - "aaz", und so weiter
- *Letter* - wie *letter*, aber für Großbuchstaben

Auf den Inhalt einer Wiederholungsvariablen können Sie mit Pfadausdrücken oder Python-Ausdrücken zugreifen. Ein Pfadausdruck ist die dreiteilige Angabe eines Pfades: Name der Variablen `repeat`, Name der Anweisungsvariablen, Name der gewünschten Information; also zum Beispiel `repeat/item/start`. In Python-Ausdrücken rufen Sie zuerst mit Hilfe der normalen Wörterbuchnotation die Wiederholungsvariable ab, und dann über den Attributzugriff die gewünschte Information. Zum Beispiel: 'python:repeat['item'].start'.

Tipps für die Wiederholung

Der folgende Abschnitt erläutert einige nützliche Tipps. Manchmal möchte man ein Tag wiederholen, allerdings ohne umschließendes Tag. Nehmen wir zum Beispiel an, Sie möchten eine Reihe von Absatz-Tags wiederholen, ohne sie in ein anderes Tag einzuschließen. Dafür verwenden Sie die Anweisung `tal:omit-tag`: pre> Zitat

Die Anweisung `tal:omit-tag` wird weiter unten in diesem Kapitel erklärt.

Der folgende Tipp wurde bereits einmal erwähnt: `tal:repeat`-Anweisungen dürfen geschachtelt werden. Dabei muss jedes `tal:repeat` einen eindeutigen Wiederholungs-Variablennamen haben. Das folgende Beispiel zeigt eine Multiplikationstabelle:

```
<table border="1">
  <tr tal:repeat="x python:range(1, 13)">
    <div tal:repeat="y python:range(1, 13)"
         tal:omit-tag="">
      <td tal:content="python:'%d x %d = %d' % (x, y, x*y)">
        X x Y = Z
      </td>
    </div>
  </tr>
</table>
```

Dieses Beispiel verwendet Python-Ausdrücke sowie die Anweisung `tal:omit-tag`. Beide werden weiter unten in diesem Kapitel näher erklärt.

Falls Sie bereits mit der DTML-Wiederholungsanweisung `dtml-in` gearbeitet haben, ist Ihnen der Begriff "Stapelung" (Batching) sicher schon untergekommen. Als Stapelung wird die Aufteilung einer großen Liste in mehrere kleinere Listen bezeichnet. In der Regel

verwendet man dieses Verfahren, um auf einer Web-Seite nur einen Teil der Elemente einer umfangreichen Liste anzuzeigen. Ein anschauliches Beispiel sind Suchmaschinen, die die große Menge der Suchergebnisse in kleinere Gruppen zerlegen. Die Anweisung `tal:repeat` unterstützt die Stapelung zwar nicht, doch Zope enthält ein Stapelprogramm. Weitere Informationen dazu finden Sie im Abschnitt "Stapelung" weiter unten in diesem Kapitel.

Eine weitere nützliche Funktion, die von `tal:repeat` nicht bereitgestellt wird, ist das Sortieren. Möchten Sie eine Liste sortieren, so können Sie entweder ein eigenes Sortierskript schreiben (in Python ist das ganz einfach) oder die Funktion `sequence.sort` verwenden. Das folgende Beispiel zeigt, wie man eine Liste von Objekten auf dem Titel und dem Änderungsdatum sortiert.

```
          <table tal:define="objects here/objectValues;
                             sort_on python:(('title', 'nocase', 'asc'),
                                             ('bobobase_modification_time',
'cmp', 'desc'));
                             sorted_objects python:sequence.sort(objects,
sort_on)">
            <tr tal:repeat="item sorted_objects">
              <td tal:content="item/title">Titel</td>
              <td tal:content="item/bobobase_modification_time">
                Änderungsdatum</td>
            </tr>
          </table>
```

Dieses Beispiel ist sehr übersichtlich, da es die Argumente für die Sortierung außerhalb der Funktion `sort` definiert. Die Funktion `sequence.sort` nimmt eine Folge sowie eine Beschreibung, wie diese Folge sortiert werden soll, entgegen. In diesem Beispiel ist die Beschreibung des Sortiervorgangs in der Variablen `sort_on` definiert. Weitere Informationen zu der leistungsstarken Funktion `sequence.sort` finden sie im Anhang B, "API-Referenz".

Attributsteuerung

Die Anweisung `tal:attributes` kennen Sie bereits. Mit Hilfe dieser Anweisung können Sie Tag-Attribute wie z. B. das Attribut `href` eines Elementes `a` dynamisch ersetzen. Auch das Ersetzen mehrerer Attribute eines Tags ist möglich. Dabei werden die Attribute durch Semikola voneinander getrennt:

```
        <a href="link"
           tal:attributes="href here/getLink;
                           class here/getClass">Link</a>
```

Auch mit XML-Namensräumen können Sie Attribute definieren:

```
        <Description
            dc:Creator="creator name"
            tal:attributes="dc:Creator here/owner/getUserName">
          Beschreibung</Description>
```

Um Attribute mit XML-Namensräumen zu erstellen, setzen Sie einfach das XML-Namensraumpräfix vor den Attributnamen.

Variablen definieren

Mit dem Attribut `tal:define` können Sie eigene Variablen definieren. Dies kann aus mehreren Gründen wünschenswert sein, zum Beispiel wenn ein langer Ausdruck innerhalb einer Vorlage immer wieder vorkommt oder eine aufwändige Methode häufig aufgerufen wird. Eine Variable wird nur einmal definiert, und kann dann in einer Vorlage beliebig oft verwendet werden. Die folgende Liste definiert eine Variable , prüft sie und wiederholt auf ihr:

```
<ul tal:define="items container/objectIds"
    tal:condition="items">
  <li tal:repeat="item items">
    <p tal:content="item">ID</p>
  </li>
</ul>
```

Die Anweisung `tal:define` erstellt die Variable `items`, die an beliebiger Stelle im Tag `ul` verwendet werden kann. Achten Sie in diesem Beispiel auch darauf, dass zwei TAL-Anweisungen innerhalb desselben `ul`-Tags eingesetzt werden. Im Abschnitt "Interaktionen zwischen TAL-Anweisungen" weiter unten in diesem Kapitel finden Sie weitere Informationen zur Verwendung mehrerer Anweisungen innerhalb eines Tags. In diesem Fall weist die erste Anweisung die Variable `items` zu, während die zweite Anweisung in einer Bedingung prüft, ob `items` wahr oder falsch (eine leere Folge) ist. Ist die Variable `items` falsch, so wird das Tag `ul` nicht angezeigt.

Gehen wir nun davon aus, dass Sie die Liste, falls sie keine Elemente enthält, nicht einfach entfernen, sondern stattdessen eine Meldung ausgeben möchten. Hierfür fügen Sie vor der Liste den folgenden Code ein:

```
<h4 tal:condition="not:container/objectIds">Keine Elemente
vorhanden</h4>
```

Der Ausdruck `not:container/objectIds` ist dann wahr, wenn `container/objectIds` falsch ist, und umgekehrt. Weitere Informationen dazu finden Sie im Abschnitt "not-Ausdrücke" weiter unten in diesem Kapitel.

Hier können Sie die Variable `items` nicht verwenden, da sie an dieser Stelle noch nicht definiert ist. Verschieben Sie die Definition von `items` jedoch in das Tag `h4`, so können Sie sie im Tag `ul` nicht mehr verwenden, da sie in diesem Fall eine *lokale* Variable von `h4` ist. Sie könnten die Definition zwar in ein Tag einfügen, das sowohl `h4` als auch `ul` einschließt, doch es gibt auch eine einfachere Möglichkeit. Stellen Sie dem Variablennamen das Schlüsselwort `global` voran, so ist die Definition vom Tag `h4` bis zum Ende der Vorlage gültig:

```
<h4 tal:define="global items container/objectIds"
    tal:condition="not:items">Keine Elemente vorhanden</h4>
```

Mit `tal:define` können Sie auch mehrere Variablen gleichzeitig definieren, indem Sie diese durch Semikola voneinander trennen:

```
<p tal:define="ids container/objectIds;
               title container/title">
```

Auf diese Art können Sie beliebig viele Variablen definieren. Jede dieser Variablen kann einen eigenen globalen oder lokalen Gültigkeitsbereich haben. Außerdem können Sie in einer Definition auf zuvor bereits definierte Variablen verweisen:

```
<p tal:define="title template/title;
               tlen python:len(title);">
```

Wie Sie sehen, kann der gezielte Einsatz von `tal:define` die Effizienz und Lesbarkeit einer Vorlage wesentlich verbessern.

Tags entfernen

Mit der Anweisung `tal:omit-tag` können Sie Tags entfernen. Diese TAL-Anweisung kommt nur selten zum Einsatz, kann gelegentlich aber durchaus nützlich sein. Das Attribut `omit-tag` entfernt ein Tag, hat aber keinerlei Auswirkungen auf den Inhalt dieses Tags. Hierzu ein Beispiel:

```
<b tal:omit-tag=""><i>this</i> bleibt</b>
```

Dieser Code wird umgesetzt als:

```
<i>dies</i> bleibt
```

Wird `tal:omit-tag`auf diese Art eingesetzt, so funktioniert es fast wie `tal:replace="default"`. Allerdings ist `tal:omit-tag` nützlicher, wenn es zusammen mit anderen TAL-Anweisungen wie z. B. `tal:repeat` verwendet wird. Das folgende Beispiel zeigt eine Möglichkeit, mit `tal:repeat` zehn Absatz-Tags zu erstellen:

```
<span tal:repeat="n python:range(10)"
      tal:omit-tag="">
  <p tal:content="n">1</p>
</span>
```

Dieser Code erstellt zehn Absatz-Tags, wobei das Tag `span` nicht mit ausgegeben wird.

Das Attribut `tal:omit-tag` nimmt einen Ausdruck entgegen, wobei in der Regel allerdings ein leerer Ausdruck verwendet wird. Ist der Ausdruck wahr oder gibt es keinen Ausdruck, so wird das Ausdrucks-Tag entfernt. Ist der Ausdruck falsch, so wird das Tag nicht entfernt. Dieses Attribut gibt Ihnen also die Möglichkeit, Tags nur unter bestimmten Umständen zu entfernen.

Fehlerbehandlung

Falls in Ihrer Seitenvorlage ein Fehler auftrifft, können Sie diesen abfangen und eine sinnvolle Fehlermeldung ausgeben. Nehmen wir zum Beispiel an, dass die Vorlage anhand von Formulardaten eine Variable definiert:

```
...
<span tal:define="global prefs request/form/prefs"
      tal:omit-tag="" />
...
```

Stößt Zope auf ein Problem, kann es zum Beispiel die Variable `prefs` in den Formulardaten nicht finden, so wird der gesamte Seitenaufbau abgebrochen und stattdessen eine Fehlerseite ausgegeben. Zum Glück lässt sich dies durch eine eingeschränkte Fehlerbehandlung mit der Anweisung `tal:on-error` vermeiden:

```
...
<span tal:define="global prefs here/scriptToGetPreferences"
      tal:omit-tag=""
      tal:on-error="string:Ein Fehler ist aufgetreten.">
...
```

Tritt während der Umsetzung einer Vorlage ein Fehler auf, so sucht Zope nach einer `tal:on-error`-Anweisung, um den Fehler zu behandeln. Zuerst sucht es im aktuellen Tag, dann im umschließenden Tag und so weiter, bis es schließlich die oberste Ebene erreicht. Findet Zope einen Fehlerbehandler, so ersetzt es den Inhalt des betroffenen Tags durch diesen Ausdruck für die Fehlerbehandlung. In diesem Fall enthält das Tag `span` dann eine Fehlermeldung.

Normalerweise wird ein Fehlerbehandler auf einem Tag definiert, das ein logisches Seitenelement umschließt, also z. B. auf einer Tabelle. Falls ein Fehler auftritt, während die Tabelle gezeichnet wird, kann der Fehlerbehandler die Tabelle einfach aus der Seite entfernen oder sie durch eine Fehlermeldung ersetzen.

Eine flexiblere Fehlerbehandlung wird durch den Aufruf eines Skripts ermöglicht:

```
<div tal:on-error="structure here/handleError">
...
</div>
```

Jeder Fehler, der innerhalb von `div` auftritt, ruft das Skript `handleError` auf. Beachten Sie, dass es die Option `structure` dem Skript ermöglicht, zu HTML zurückzukehren. Das Skript

zur Fehlerbehandlung kann den Fehler untersuchen und je nachdem, um welchen Fehler es sich handelt, unterschiedliche Maßnahmen ergreifen. Das Skript greift über die Variable `error` im Namensraum auf den Fehler zu. Hierzu ein Beispiel:

```
## Script (Python) "handleError"
##binde Namensraum=_
##
error=_['error']
if error.type==ZeroDivisionError:
    return "<p>Kann nicht durch null teilen.</p>"
else
    return """<p>Ein Fehler ist aufgetreten.</p>
              <p>Fehlertyp: %s</p>
              <p>Fehlerwert: %s</p>""" % (error.type,
                                          error.value)
```

Das Skript zur Fehlerbehandlung kann beliebige Aktionen durchführen. So kann es zum Beispiel ein Fehlerprotokoll per E-Mail abschicken.

Die Anweisung `tal:on-error` ist nicht auf eine allgemeine Ausnahmebehandlung angelegt. Zum Beispiel ist sie nicht für die Validierung von Formulareingaben geeignet. Diese sollte mit einem Skript durchgeführt werden, da Skripte eine leistungsstarke Ausnahmebehandlung ermöglichen. Die Anweisung `tal:on-error` dient lediglich der Behandlung ungewöhnlicher Probleme, die bei der Umsetzung von Vorlagen auftreten können.

Interaktionen zwischen TAL-Anweisungen

Gibt es für jedes Element nur eine einzige TAL-Anweisung, so folgt deren Ausführung einer einfachen Logik: Zuerst wird die Anweisung des Wurzelelementes ausgeführt, dann werden der Reihe nach alle Kindelemente geprüft und ihre Anweisungen ausgeführt.

Es ist allerdings auch möglich, dass es auf einem Element mehr als eine TAL-Anweisung gibt. Dabei ist mit Ausnahme der Anweisungen `tal:content` und `tal:replace`, die nicht gemeinsam verwendet werden dürfen, jede Kombination von Anweisungen möglich.

Verfügt ein Element über mehrere Anweisungen, so werden diese in der folgenden Reihenfolge ausgeführt:

1. define
2. condition
3. repeat
4. content oder replace
5. attributes
6. omit-tag

Die Anweisung `tal:on-error` wird hier nicht genannt, da sie nur aufgerufen wird, wenn ein Fehler auftritt.

Diese Reihenfolge ist natürlich nicht willkürlich festgelegt. Häufig definiert man Variablen, um sie dann in anderen Anweisungen zu verwenden. Daher steht `define` an erster Stelle. Anschließend muss man entscheiden, ob dieses Element überhaupt aufgenommen wird, wofür `condition` zuständig ist. Da die jeweilige Bedingung von gerade definierten Variablen abhängen kann, folgt `condition` direkt hinter `define`. Es ist nützlich, wenn man bestimmte Teile eines Elements bei jeder Iteration von `repeat` durch andere Werte ersetzen kann. Daher steht `repeat` vor `content`, `replace` und `attributes`. Da `content` und `replace` nicht auf demselben Element ausgeführt werden können, stehen sie nebeneinander. Die Anweisung `omit-tag` steht an letzter Stelle, da es höchst unwahrscheinlich ist, dass irgendeine andere Anweisung von ihr abhängt, und sie nach `repeat` aufgeführt werden sollte.

Das folgende Beispiel-Tag enthält mehrere TAL-Anweisungen:

```
<p tal:define="x /root/a/long/path/x | nothing"
   tal:condition="x"
   tal:content="x/txt"
   tal:attributes="class x/class">Ex Text</p>
```

Beachten Sie, dass die Anweisung `tal:define` zuerst ausgeführt wird und die anderen Anweisungen von ihrem Ergebnis abhängen.

Wenn Sie TAL-Anweisungen auf Elementen kombinieren, sollten Sie die folgenden drei Einschränkungen beachten:

1. Da HTML die Ausführung mehrerer Attribute mit demselben Namen verbietet, kann auf einem bestimmten Tag nur eine Instanz eines Anweisungstyps ausgeführt werden. So ist es zum Beispiel nicht möglich, zwei Instanzen von `tal:define` auf demselben Tag auszuführen.
2. Da die Funktionen von `tal:content` und `tal:replace` nicht miteinander vereinbar sind, kann auf jedem Tag nur eine dieser beiden Anweisungen ausgeführt werden.
3. Die Reihenfolge, in der TAL-Attribute auf einem Tag notiert werden, hat keinerlei Einfluss auf die Reihenfolge ihrer Ausführung. Unabhängig von ihrer Anordnung werden die TAL-Anweisungen auf einem Tag immer in der oben genannten Reihenfolge ausgeführt.

Möchten Sie die Reihenfolge von TAL-Anweisungen überschreiben, so müssen Sie das Element in ein anderes Element einschließen, z. B. in `div` oder `span`, und einige der Anweisungen in dieses neue Element einfügen. Nehmen wir z. B. an, Sie möchten eine Folge von Objekten mit einer Schleife durchlaufen, dabei aber einige der Objekte überspringen. Eine Vorlage, die die Zahlen 0 bis 9 mit einer Schleife durchläuft und dabei die 3 überspringt, könnte folgendermaßen aussehen:

```
<!-- broken template -->
<ul>
  <li tal:repeat="n python:range(10)"
      tal:condition="python:n != 3"
      tal:content="n">
    1
  </li>
</ul>
```

Diese Vorlage funktioniert nicht, da die Bedingung vor der Ausführung von `repeat` geprüft wird. Anders ausgedrückt: Die Variable `n` soll zuerst geprüft und danach erst definiert werden. Dieses Problem lässt sich folgendermaßen umgehen:

```
<ul>
  <div tal:repeat="n python:range(10)"
       tal:omit-tag="">
    <li tal:condition="python:n != 3"
        tal:content="n">
      1
    </li>
  </div>
</ul>
```

Diese Vorlage löst das Problem, indem sie die Variable `n` auf einem umschließenden `div`-Tag definiert. Beachten Sie, dass das Tag `div` nicht in die Ausgabe übernommen wird, da es die Anweisung `tal:omit-tag` enthält. Dieser Code sieht zwar nicht schön aus, funktioniert aber. Möglicherweise werden zukünftige Seitenvorlagen-Versionen eine attraktivere Lösung dieses Problems anbieten.

Formularverarbeitung

In DTML können Sie Formulare mit einem einfachen Muster verarbeiten, dem so genannten "Formular-/Aktionspaar". Ein Formular-/Aktionspaar besteht aus zwei DTML-Methoden oder Dokumenten: Die bzw. das erste enthält ein Formular, das eine Eingabe des Anwenders entgegennimmt, und die bzw. das zweite enthält eine Aktion, die auf dieser Eingabe ausgeführt wird, und gibt eine Antwort aus. Das Formular ruft die Aktion auf. Weitere Informationen zum Formular-/Aktionsmuster finden Sie in Kapitel 4, "Dynamischer Inhalt mit DTML".

In Zope-Seitenvorlagen lässt sich das Formular-/Aktionsmuster nicht sonderlich gut einsetzen, da es davon ausgeht, dass die Verarbeitung der Eingabe und die Darstellung der Antwort von demselben Objekt (der Aktion) gehandhabt werden. Anstelle des Formular-/Aktionsmusters empfiehlt sich daher die Verwendung eines Formular-/Aktions-/Antwortmusters. Dabei sollten das Formular und die Antwort Seitenvorlagen sein, die Aktion hingegen ein Skript, das die Eingabe verarbeitet und eine Antwortvorlage zurückgibt. Dieses Muster ist flexibler als das Formular-/Aktionsmuster, da das Skript ein beliebiges Objekt aus einer Gruppe von Antwortobjekten zurückgeben kann.

Das folgende Beispiel veranschaulicht eine Formularvorlage:

```
...
<form action="action">
  <input type="text" name="name">
  <input type="text" name="age:int">
  <input type="submit">
</form>
...
```

Für die Verarbeitung des Formulars könnte ein Skript wie das folgende zuständig sein:

```
## Script (Python) "action"
##Parameter=name, age
##
container.addPerson(name, age)
return container.responseTemplate()
```

Dieses Skript ruft eine Methode für die Verarbeitung der Eingabe auf und gibt eine weitere Vorlage zurück, die die Antwort enthält. Durch den Aufruf dieses Skripts können Sie eine Seitenvorlage aus Python umsetzen. Die Antwortvorlage enthält normalerweise die Bestätigung, dass das Formular richtig verarbeitet wurde.

Das Aktionsskript kann alle möglichen Aktionen durchführen. Es kann eine Eingabe prüfen, Fehler behandeln, E-Mails verschicken usw. Das folgende Beispiel zeigt, wie eine Eingabe mit einem Skript geprüft wird:

```
## Script (Python) "action"
##
if not context.validateData(request):
    # Tritt ein Problem auf, so gib die Formularseitenvorlage
    # zusammen mit einer Fehlermeldung zurück
    return context.formTemplate(error_message='Invalid data')

# Ansonsten gib die Dankesseite zurück
return context.responseTemplate()
```

Dieses Skript prüft die Formulareingabe und gibt die Formularvorlage mit einer Fehlermeldung zurück, falls ein Problem aufgetreten ist. An Seitenvorlagen können Sie Zusatzinformationen mit Schlüsselwortargumenten übergeben. Die Schlüsselwortargumente stehen der Vorlage über die integrierte Variable `options` zur Verfügung. Die Formularvorlage kann also einen Abschnitt wie den folgenden enthalten:

```
<b tal:condition="options/error_message | nothing"
   tal:content="options/error_message">
  Hier folgt die Fehlermeldung.
</b>
```

Dieses Beispiel zeigt, wie Sie eine Fehlermeldung ausgeben können, die über Schlüsselwortargumente an die Vorlage übergeben werden.

Je nach Anwendung können Sie den Anwender auch an eine Antwortseitenvorlage weiterleiten, statt diese direkt zurückzugeben. Obwohl dies die Netzwerkaktivität verdoppelt, ist es durchaus sinnvoll, da hierbei im Browser des Anwenders nicht der URL des Aktionsskripts angezeigt wird, sondern der der Seitenvorlage.

Falls Sie keinen Wert auf saubere Formularverarbeitung legen, können Sie mit Ihren Seitenvorlagen auch eine eingeschränkte Version des Formular-/Aktionspaares verwenden. Dies sollten Sie allerdings wirklich nur dann machen, wenn Ihnen an einer Fehlerbehandlung

nicht gelegen ist und die Antwort in jedem Fall, unabhängig von der jeweiligen Eingabe des Anwenders, dieselbe ist. Da Seitenvorlagen keine Entsprechung zu `dtml-call` haben, können Sie einen Trick anwenden, um eine Methode zur Eingabeverarbeitung aufzurufen, ohne deren Ergebnisse einzufügen. Hierzu ein Beispiel:

```
<span tal:define="unused here/processInputs"
      tal:omit-tag=""/>
```

Dieses Beispiel ruft die Methode `processInputs` auf und weist ihr Ergebnis der Variable `unused` zu.

Ausdrücke

Seitenvorlagenausdrücke haben Sie bereits kennen gelernt. Ausdrücke stellen Werte für Vorlagenanweisungen bereit. Ein Beispiel sind Pfadausdrücke, die Objekte durch einen Pfad wie `request/form/age` oder `user/getUserName` beschreiben. Dieser Abschnitt erklärt die verschiedenen Arten von Ausdrücken und Variablen.

Integrierte Variablen

Variablen sind Namen, die in Ausdrücken verwendet werden können. Sie haben bereits einige Beispiele für integrierte Variablen kennen gelernt, z. B. `template`, `user`, `repeat` und `request`. Die folgende Liste nennt alle weiteren integrierten Variablen und ihre Verwendungsweise.

`nothing`
: Ein false-Wert, ähnlich einer leeren Zeichenkette, den Sie in `tal:replace` oder `tal:content` verwenden können, um ein Tag oder seinen Inhalt zu löschen. Es besteht allerdings ein Unterschied zu leeren Zeichenketten: Wenn Sie ein Attribut auf `nothing` setzen, wird dieses Attribut aus dem Tag gelöscht (oder nicht eingefügt).

`default`
: Ein besonderer Wert, der nichts ändert, wenn er in `tal:replace`, `tal:content` oder `tal:attributes` verwendet wird. Er beeinflusst den Vorlagentext nicht.

`options`
: Die an die Vorlage übergebenen Schlüsselwortargumente, sofern vorhanden. Hinweis: `options` steht nur zur Verfügung, wenn eine Vorlage aus Python ausgerufen wird. Wird sie aus dem Internet umgesetzt, so gibt es keine Optionen.

`attrs`
: Ein Attributwörterbuch des aktuellen Tags in der Vorlage. Als Schlüssel dienen die Attributnamen und als Werte die ursprünglichen Werte der Attribute in der Vorlage. Diese Variable kommt selten zum Einsatz.

`root`
: Das Zope-Wurzelobjekt. Damit werden Zope-Objekte von festen Speicherorten geholt, unabhängig davon, wo die Vorlage platziert oder aufgerufen wird.

`here`
: Das Objekt, auf dem die Vorlage aufgerufen wird. Häufig ist es mit dem *Behälter* identisch; falls Sie mit Akquisition arbeiten, kann `here` allerdings auch ein anderes Objekt bezeichnen. Mit dieser Variable werden Zope-Objekte abgerufen, die sich je nachdem, wie die Vorlage aufgerufen wird, an verschiedenen Stellen befinden können. `here` entspricht der Variable `context` in Python-basierten Skripten.

`container`
: Der Behälter (normalerweise ein Ordner), in dem sich die Vorlage befindet. Mit dieser Vorlage werden Zope-Objekte von Speicherorten abgerufen, der relativ zum dauerhaften Speicherort der Vorlage angegeben ist. Die Variablen `container` und `here` verweisen auf dasselbe Objekte, falls eine Schablone von ihrem dauerhaften Speicherort aufgerufen wird. Wird eine Vorlage allerdings auf ein anderes Objekt (z. B. eine ZSQL-Methode) angewandt, so verweisen `container` und `here` nicht auf dasselbe Objekt.

`modules`
: Die Sammlung von Python-Modulen, die Vorlagen zur Verfügung stehen. Weitere Informationen dazu finden Sie im Abschnitt über Python-Ausdrücke.

Dieses Kapitel enthält eine Reihe von Beispielen für die Verwendung dieser Variablen.

Zeichenkettenausdrücke

Zeichenkettenausdrücke ermöglichen die problemlose Verbindung von Pfadausdrücken und Text. Der gesamte Text nach dem öffnenden Tag `string:` wird auf Pfadausdrücke hin durchsucht. Jedem Pfadausdruck muss ein Dollar-Zeichen ('$') vorangestellt sein:

```
"string:Nur Text. Kein Pfad."
"string:copyright $year, ich."
```

Besteht der Pfadausdruck aus mehreren Teilen oder muss er vom danach folgenden Text abgetrennt werden, so wird er in geschweifte Klammern ('{}') gesetzt:

```
"string:Drei ${Birne}n, bitte."
"string:Ihr Name ist ${user/getUserName}!"
```

Achten Sie bei diesem Beispiel darauf, dass der Pfad `Birne` in geschweifte Klammern gesetzt werden muss, damit Zope ihn nicht mit `Birnen` verwechselt.

Da sich der Text innerhalb eines Attributwerts befindet, können Sie doppelte Anführungszeichen nur mit Hilfe der Entitätssyntax `&quot;` einfügen. Kommt im Text ein Dollar-Zeichen vor, so muss es verdoppelt werden ('$$'), weil das einfache Dollar-Zeichen Pfadausdrücke kennzeichnet:

```
"string:Bitte bezahlen Sie $$$dollars_owed"
"string:Sie sagte: &quot;Hallo Welt.&quot;"
```

Einige komplexe Formatierungsoptionen für Zeichenketten (z. B. Suchen und Ersetzen oder Umwandlung in Großbuchstaben) können mit Zeichenkettenausdrücken problemlos umgesetzt werden. Am besten sind hierfür Python-Ausdrücke oder Skripte geeignet.

Pfadausdrücke

Pfadausdrücke verweisen mit einem URL-ähnlichen Pfad auf Objekte. Ein Pfad beschreibt den Weg von einem Objekt zu einem anderen Objekt. Alle Pfade beginnen mit einem bekannten Objekt (z. B. einer integrierten Variable, einer Wiederholungsvariable oder einer benutzerdefinierten Variable) und gehen von dort aus zum gewünschten Objekt. Einige Beispiele für Pfadausdrücke:

```
template/title
container/files/objectValues
user/getUserName
container/master.html/macros/header
request/form/address
root/standard_look_and_feel.html
```

Pfadausdrücke ermöglichen den Wechsel von einem Objekt zu seinen Unterobjekten einschließlich der Eigenschaften und Methoden. Auch die Akquisition kann in Pfadausdrücken verwendet werden. Weitere Informationen zur Akquisition und Pfaddurchquerung finden Sie im Abschnitt "Skripte aus dem Internet aufrufen" in Kapitel 10, "Zope-Scripting für Fortgeschrittene"; und in Kapitel 7, "Benutzer und Sicherheit", erfahren Sie mehr über die Objektzugriffssteuerung.

Alternative Pfade

Der Pfad `template/title` besteht immer, wenn die Vorlage verwendet wird, kann allerdings eine leere Zeichenkette sein. Andere Pfade, z. B. `request/form/x`, gibt es bei bestimmten Umsetzungen der Schablone nicht. In diesem Fall tritt ein Fehler auf, wenn Zope den Pfadausdruck auswertet.

Gibt es einen bestimmten Pfad nicht, so können Sie einen Ersatzpfad oder -wert einrichten, der statt dieses Pfades verwendet werden soll. Existiert z. B. `request/form/x` nicht, so können Sie `here/x` als Alternative festlegen. Um einen alternativen Pfad zu definieren, geben Sie eine Liste von Pfaden an, die nach Priorität geordnet und durch senkrechte Striche ('|') voneinander abgetrennt werden:

```
<h4 tal:content="request/form/x | here/x">Header</h4>
```

Als letztes Element einer Liste alternativer Pfade sind besonders die Variablen `nothing` und `default` sinnvoll. In diesem Fall teilt z. B. `default` der Anweisung `tal:content` mit, dass der Blindinhalt unverändert bleiben soll. Andere TAL-Anweisungen interpretieren `default` und `nothing` anders. Weitere Informationen hierzu finden Sie in Anhang C, "Zope-Page-Templates-Referenz".

Der letzte Teil eines Alternativpfadausdrucks kann auch ein Nicht-Pfadausdruck sein, zum Beispiel:

```
<p tal:content="request/form/age|python:18">Alter</p>
```

Existiert in diesem Beispiel der Pfad `request/form/age` nicht, so ist der Wert die Zahl 18. Dieses Formular ermöglicht die Angabe von Standardwerten, die nicht als Pfade ausgedrückt werden können. Beachten Sie, dass Sie Nicht-Pfadausdrücke nur als letzte Alternative verwenden können.

Mit dem Ausdruckstyppräfix *exists* kann direkt geprüft werden, ob ein Pfad existiert. Weitere Informationen zu `exists`-Ausdrücken finden Sie im Abschnitt "exists-Ausdrücke" weiter unten in diesem Kapitel.

not-Ausdrücke

Mit not-Ausdrücken können Sie den Wert anderer Ausdrücke negieren. Hierzu ein Beispiel:

```
<p tal:condition="not:here/objectIds">
  Hier sind keine Objekte enthalten.
</p>
```

Ist der Ausdruck, auf den ein not-Ausdruck angewendet wird, falsch, so gibt der not-Ausdruck `true` zurück, und umgekehrt. In Zope werden nicht vorhandene Variablen, null, leere Zeichenketten, leere Folgen, nothing und None als falsch gewertet, alles andere als wahr.

In Verbindung mit Python-Ausdrücken wird normalerweise kein not-Ausdruck verwendet, sondern das Python-Schlüsselwort `not`.

nocall-Ausdrücke

Ein gewöhnlicher Pfadausdruck versucht das Objekt, das er abruft, umzusetzen. Falls das Objekt eine Funktion, ein Skript, eine Methode oder ein anderes ausführbares Objekt ist, bedeutet dies, dass der Ausdruck als das Ergebnis des Objektaufrufs ausgewertet wird. Dies ist zwar meist das gewünschte Ergebnis, aber nicht immer. Möchte man z. B. ein DTML-Dokument in eine Variable einfügen, um auf seine Eigenschaften verweisen zu können, so kann man dafür keinen normalen Pfadausdruck verwenden, da dieser das Dokument in eine Zeichenkette umwandelt.

Setzt man einem Pfad das Ausdruckstyppräfix `nocall:` voran, so wird die Umsetzung unterbunden und nur das Objekt abgerufen. Hierzu ein Beispiel:

```
<span tal:define="doc nocall:here/aDoc"
      tal:content="string:${doc/getId}: ${doc/title}">
ID: Titel</span>
```

Dieser Ausdruckstyp ist auch nützlich, wenn Sie eine Variable definieren möchten, die zur Weiterverwendung in einem Python-Ausdruck eine Funktion oder Klasse aus einem Modul enthalten soll.

Nicht nur auf Objekten, sondern auch auf Funktionen können nocall-Ausdrücke verwendet werden:

```
<p tal:define="join nocall:modules/string/join">
```

Dieser Ausdruck definiert die Variable `join` nicht als Ergebnis eines Funktionsaufrufs, sondern als Funktion ('string.join').

exists-Ausdrücke

Ein exists-Ausdruck ist dann wahr, wenn sein Pfad existiert; ansonsten ist er falsch. Das folgende Beispiel zeigt eine Möglichkeit, eine Fehlermeldung nur dann auszugeben, wenn sie in der Anfrage übergeben wird:

```
<h4 tal:define="err request/form/errmsg | nothing"
    tal:condition="err"
    tal:content="err">Fehler!</h4>
```

Mit einem exists-Ausdruck erzielen Sie dasselbe Ergebnis auf einfachere Art:

```
<h4 tal:condition="exists:request/form/errmsg"
    tal:content="request/form/errmsg">Fehler!</h4>
```

Auch eine Kombination von not- und exists-Ausdrücken ist möglich:

```
<p tal:condition="not:exists:request/form/number">Geben Sie bitte
eine Zahl zwischen 0 und 5 ein.</p>
```

Beachten Sie, dass der Ausdruck "not:request/form/number" in diesem Beispiel nicht verwendet werden kann, da er wahr ist, falls die Variable `number` existiert und den Wert null hat.

Python-Ausdrücke

Python ist eine einfache und ausdrucksstarke Programmiersprache. Falls Sie bisher noch nicht damit gearbeitet haben, sollten Sie eine der hervorragenden Einführungen lesen, die Sie auf der Python-Website finden.

Ein Python-Ausdruck für Seitenvorlagen kann alles enthalten, was Python als einen Ausdruck erkennt. Nicht verwendet werden dürfen Anweisungen wie zum Beispiel `if` und `while`. Außerdem erzwingt Zope einige sicherheitsrelevante Einschränkungen, um den Zugriff auf geschützte Informationen, die Änderung gesicherter Daten und die Einführung problematischer Verfahren (z. B. Endlosschleifen) zu verhindern. Weitere Informationen zu Sicherheitsrestriktionen für Python finden Sie in Kapitel 10, "Zope-Scripting für Fortgeschrittene".

Vergleich

In einigen Fällen sind Python-Ausdrücke fast unumgehbar. Hierzu gehört z. B. die Anweisung `tal:condition`. Normalerweise vergleicht man zwei Zeichenketten oder Zahlen, und dies ist nur mit Python-Ausdrücken möglich. Sie können die Vergleichsoperatoren `<` (kleiner als), `>` (größer als), `==` (ist gleich) und `!=` (ist nicht gleich) sowie die Booleschen Operatoren `and`, `not` und `or` verwenden. Beispiele:

```
<p tal:repeat="widget widgets">
  <span tal:condition="python:widget.type == 'gear'>
  Gang #<span tal:replace="repeat/widget/number>1</span>:
  <span tal:replace="widget/name">Name</span>
  </span>
</p>
```

Dieses Beispiel durchläuft eine Sammlung von Objekten mit einer Schleife und prüft dabei das Attribut `type` der einzelnen Objekte.

Manchmal möchte man innerhalb einer Anweisung anhand einer oder mehrerer Bedingungen verschiedene Werte wählen. Dies ist mit der Funktion `test` möglich:

```
Sie <span tal:define="name user/getUserName"
     tal:replace="python:test(name=='Anonymer Anwender',
                              'bitte anmelden', default)">
      sind angemeldet als
      <span tal:replace="name">Name</span>
    </span>
```

Die Funktion `test` funktioniert wie eine if/then/else-Anweisung. Weitere Informationen zu dieser Funktion finden Sie in Anhang A, "DTML-Referenz". Das folgende Beispiel veranschaulicht die Verwendung der Funktion `test`:

```
<tr tal:define="oddrow repeat/item/odd"
    tal:attributes="class python:test(oddrow, 'oddclass',
                                      'evenclass')">
```

Ohne die Funktion `test` müssten Sie hier zwei `tr`-Elemente mit jeweils unterschiedlichen Bedingungen schreiben: eine für gerade Zeilen und eine für ungerade Zeilen.

Weitere Ausdruckstypen

Auch andere Ausdruckstypen können in einem Python-Ausdruck verwendet werden. Jedem Ausdruckstyp entspricht eine gleichnamige Funktion: `path()`, `string()`, `exists()` und `nocall()`. Somit kann man Ausdrücke wie die folgenden schreiben:

```
"python:path('here/%s/thing' % foldername)"
"python:path(string('here/$foldername/thing'))"
"python:path('request/form/x') or default"
```

Das letzte Beispiel hat eine etwas andere Bedeutung als der Pfadausdruck "request/form/x | default", da es den Standardtext verwendet, falls "request/form/x" nicht existiert *oder* falsch ist.

Auf Zope-Objekte zugreifen

Zope ist vor allem auch deshalb so leistungsfähig, weil es spezialisierte Objekte miteinander verbindet. Seitenvorlagen können Skripte, SQL-Methoden, Kataloge und benutzerdefinierte Inhaltsobjekte verwenden. Um diese Objekte nutzen zu können, müssen Sie wissen, wie Sie in Seitenvorlagen auf sie zugreifen können.

Objekteigenschaften sind normalerweise Attribute. Daher ruft man den Titel einer Vorlage mit dem Ausdruck "template.title" ab. Die meisten Zope-Objekte unterstützen die Akquisition, die den Zugriff auf Attribute von Elternobjekten ermöglicht. Dies bedeutet, dass der Python-Ausdruck "here.Control_Panel" das Objekt Systemsteuerung aus dem Wurzelordner erwirbt. Objektmethoden sind Attribute, z. B. "here.objectIds" und "request.set". Auf Objekte in einem Ordner kann man als Attribute dieses Ordners zugreifen. Allerdings sind ihre Namen häufig keine zulässigen Python-Bezeichner, sodass man nicht die normale Notation verwenden kann. So ist z. B. dieser Python-Ausdruck nicht zulässig:

```
"python:here.pinguin.gif"'.
```

Stattdessen müssen Sie die folgende Notation verwenden:

```
"python:getattr(here, 'pinguin.gif')"
```

Dies liegt daran, dass Python Attributnamen, die Punkte enthalten, nicht unterstützt.

Einige Objekte, z. B. `request`, `modules` und Zope-Ordner, unterstützten den Python-Elementzugriff:

```
request['URL']
modules['math']
here['thing']
```

Wenn Sie den Elementzugriff auf einen Ordner anwenden, versucht er nicht, den Namen zu akquirieren, wird also nur dann erfolgreich durchgeführt, wenn der Ordner tatsächlich ein Objekt mit dem angegebenen Namen enthält.

In anderen Kapiteln wurde bereits gezeigt, dass Sie bei Pfadausdrücken Einzelheiten des Weges von einem Objekt zum nächsten ignorieren können. Zope versucht zuerst einen Attributzugriff, dann einen Elementzugriff. Anstatt

```
"python:getattr(here.images, 'pinguin.gif')"
```

können Sie auch die folgende Schreibweise verwenden:

```
"here/images/pinguin.gif"
```

ebenso auch

```
"request/form/x"
```

an Stelle von:

```
"python:request.form['x']"
```

Der Nachteil ist, dass die Angabe dieser Einzelheiten bei Pfadausdrücken nicht möglich ist. Haben Sie z. B. eine Formularvariable namens "get", so müssen Sie

```
"python:request.form['get']"
```

schreiben, da der Pfadausdruck

```
"request/form/get"
```

als die *Methode* "get" des Formularwörterbuches ausgewertet wird.

Mit der Funktion `path()` können Sie, wie oben beschrieben, Pfadausdrücke innerhalb von Python-Ausdrücken verwenden.

Mit Skripten arbeiten

Skriptobjekte werden häufig dafür verwendet, die Anwendungslogik und eine komplexe Datenbearbeitung zu kapseln. Schreiben Sie viele TAL-Anweisungen, die komplizierte Ausdrücke enthalten, so sollten Sie auf jeden Fall erwägen, stattdessen ein Skript zu verwenden. Bereitet es Ihnen Schwierigkeiten, Ihre Vorlagenanweisungen und -ausdrücke zu durchschauen, so sollten Sie die Vorlage vereinfachen und für die komplexen Bereiche Skripte verwenden.

Für jedes Skript gibt es eine Parameterliste, die dem Skript bei dessen Aufruf übergeben wird. Ist diese Liste leer, so können Sie das Skript dennoch verwenden, indem Sie einen Pfadausdruck angeben. Anderenfalls müssen Sie das Argument mit Hilfe eines Python-Ausdrucks bereitstellen:

```
"python:here.myscript(1, 2)"
"python:here.myscript('arg', foo=request.form['x'])"
```

Geben Sie aus einem Skript mehr als ein Datenelement an eine Seitenvorlage zurück, so ist die Rückgabe in einem Wörterbuch sinnvoll, denn dabei können Sie eine Variable definieren, die alle Daten enthält, und über Pfadausdrücke auf die einzelnen Daten verweisen. Nehmen wir z. B. an, dass das Skript `getPerson` ein Wörterbuch mit den Schlüsseln `name` und `age` zurückgibt:

```
<span tal:define="person here/getPerson"
      tal:replace="string:${person/name} is ${person/age}">
Name ist 30</span> Jahre alt.
```

Natürlich können auch Zope-Objekte und Python-Listen zurückgegeben werden.

DTML aufrufen

Im Gegensatz zu Skripten haben DTML-Methoden und -Dokumente keine ausdrückliche Parameterliste. Stattdessen müssen ihnen ein Client, eine Zuordnung und Schlüsselwortargumente übergeben werden. Anhand dieser Parameter erstellen sie dann einen Namensraum. Weitere Informationen zu ausdrücklichen DTML-Aufrufen finden Sie in Kapitel 7.

Wenn Zope ein DTML-Objekt über das Internet veröffentlicht, übergibt es den Inhalt des Objekts als Client und die Anfrage (REQUEST) als Zuordnung. Wenn ein DTML-Objekt ein anderes DTML-Objekt aufruft, übergibt es keinen Client und als Zuordnung seinen eigenen Namensraum.

Falls Sie ein DTML-Objekt über einen Pfadausdruck umsetzen, übergibt es einen Namensraum, der `request`, `here` und die Variablen der Vorlage bereits enthält. Dies bedeutet, dass das DTML-Objekt nicht nur dieselben Namen wie bei einer Veröffentlichung im Kontext der Vorlage verwenden kann, sondern auch die in der Vorlage definierten Variablennamen.

Python-Module

Die Programmiersprache Python enthält eine Vielzahl von Modulen, die Python-Programme flexibel und vielseitig machen. Jedes Modul ist eine Sammlung von Python-Funktionen, -Daten und -Klassen, die sich auf einen einzigen Zweck beziehen, z. B. auf mathematische Berechnungen oder reguläre Ausdrücke.

Einige Module, darunter "math" und "string", stehen in Python-Ausdrücken immer zur Verfügung. So können Sie zum Beispiel mit "python:math.pi" den Wert von pi aus dem Modul "math" abrufen. Um darauf aus einem Pfadausdruck heraus zuzugreifen, benötigen Sie hingegen die Variable `modules`: "modules/math/pi".

Da das Modul "string"in Python-Ausdrücken durch die Ausdruckstypfunktion "string" verborgen ist, benötigen Sie für den Zugriff auf das Modul die Variable `modules`. Dabei gibt es zwei Möglichkeiten, auf das Modul zuzugreifen: Entweder direkt in einem Ausdruck, in dem Sie es verwenden, oder, wie im folgenden Beispiel, über eine selbst definierte globale Variable.

```
          tal:define="global mstring modules/string"
          tal:replace="python:mstring.join(slist, ':')"
```

In der Praxis ist dies selten erforderlich. Meist braucht man sich nicht auf Funktionen im Modul "string" zu stützen, sondern kann stattdessen Zeichenkettenmethoden verwenden.

Module können zu Paketen zusammengefasst werden. Pakete stellen eine einfache Möglichkeit dar, verwandte Module zu organisieren und zu benennen. So werden z. B. die Python-basierten Skripte in Zope als eine Sammlung von Modulen im Unterpaket "PythonScripts" des Zope-Pakets "Products" bereitgestellt. Besonders das in diesem Paket enthaltene Modul "standard" bietet eine Reihe nützlicher Formatierungsfunktionen, die im DTML-Tag "var" regelmäßig eingesetzt werden. Der vollständige Name dieses Moduls ist "Products.PythonScripts.standard"; der Zugriff erfolgt demnach mit einer der folgenden Anweisungen:

```
          tal:define="global pps modules/Products/PythonScripts/standard"
          tal:define="global pps
python:modules['Products.PythonScripts.standard']"
```

Auf die meisten Python-Module kann man nur dann aus Seitenvorlagen, DTML oder Skripten heraus zugreifen, wenn man diese um Zope-spezifische Sicherheitsmaßnahmen erweitert. Diese Verfahren können im Rahmen dieses Buches nicht besprochen werden; sie werden im *Zope Developer's Guide* näher erläutert.

Makros

Bisher haben Sie gesehen, wie Sie einzelnen Web-Seiten mit Seitenvorlagen dynamisches Verhalten hinzufügen können. Darüber hinaus ermöglichen Seitenvorlagen die Wiederverwendung von Darstellungselementen auf mehreren Seiten.

Mit Seitenvorlagen können Sie zum Beispiel eine Site mit einem einheitlichen Look&Feel einrichten: Unabhängig vom Inhalt der einzelnen Seiten erhalten alle Seiten identische Header, Footer, Seitenleisten und/oder andere Seitenelemente. Dies ist eine Anforderung, die häufig an Websites gestellt wird.

Mit *Makros* können Sie Darstellungselemente auf mehreren Seiten wiederverwenden. Makros definieren einen Seitenbereich, der in anderen Seiten wiederverwendet werden kann. Ein Makro kann ebenso gut eine ganze Seite wie ein kleiner Seitenbereich sein, also z. B. ein Header oder Footer. Nachdem Sie in einer Seitenvorlage ein oder mehrere Makros definiert haben, können Sie diese auch in anderen Seitenvorlagen verwenden.

Makros verwenden

Makros werden mit TAG-Attributen ähnlich den TAL-Anweisungen definiert. Makro-Tag-Attribute werden als METAL-Anweisungen bezeichnet, wobei METAL für Macro Expansion Tag Attribute Language steht. Das folgende Beispiel zeigt die Definition eines Makros:

```
<p metal:define-macro="copyright">
  Copyright 2001, <em>Foo, Bar und Co.</em> KG
</p>
```

Diese `metal:define-macro`-Anweisung definiert ein Makro namens "copyright", das aus dem Tag `p` und seinem Inhalt besteht (einschließlich aller darin enthaltenen Tags).

In einer Seitenvorlage definierte Makros werden im Attribut `macro` der Vorlage gespeichert. Makros aus einer anderen Vorlage können Sie verwenden, indem Sie über das Attribut `macros` der Seitenvorlage, in der sie definiert sind, auf sie verweisen. Nehmen wir z. B. an, dass das Makro `copyright` in der Seitenvorlage "master_page" definiert ist. In einer anderen Seitenvorlage wird `copyright` nun mit dem folgenden Code aufgerufen:

```
<hr>
<b metal:use-macro="container/master_page/macros/copyright">
  Hier folgt das Makro
</b>
```

Wenn Zope diese Seitenvorlage umsetzt, wird das Tag `b` vollständig durch das Makro ersetzt:

```
<hr>
<p>
  Copyright 2001, <em>Foo, Bar und Co.</em> KG
</p>
```

Wird das Makro geändert (z. B. wegen Namensänderung des Urheberrechtsinhabers), so spiegelt sich diese Änderung in allen Seitenvorlagen wider, die dieses Makro verwenden.

Beachten Sie, dass das Makro durch einen Pfadausdruck mit der Anweisung `metal:use-macro` angegeben wird. Die Anweisung `metal:use-macro` ersetzt das Anweisungselement durch das genannte Makro.

Makros im Detail

Die Anweisungen `metal:define-macro` und `metal:use-macro` sind recht einfach. Allerdings gibt es einige Feinheiten, die erwähnt zu werden verdienen.

Der Name eines Makros muss innerhalb der Seitenvorlage, in der es definiert wird, eindeutig sein. Sie können in einer Vorlage mehrere Makros definieren, doch jedes dieser Makros muss einen anderen Namen haben.

Normalerweise verweist man auf Makros in einer `metal:use-macro`-Anweisung mit einem Pfadausdruck. Eigentlich kann man aber jeden beliebigen Ausdruckstyp verwenden, wichtig ist nur, dass er ein Makro zurückgibt. Hierzu ein Beispiel:

```
<p metal:use-macro="python:here.getMacro()">
  Wird durch ein dynamisch ermitteltes Makro ersetzt,
  das vom Skript getMacro gesucht wird.
```

```
</p>
```

Verweist man mit Python-Ausdrücken auf Makros, so kann man das Makro, das eine Vorlage verwenden soll, dynamisch wechseln.

Das folgende Beispiel veranschaulicht, wie die Anweisung `metal:use-macro` zusammen mit der Variable `default` verwendet wird:

```
<p metal:use-macro="default">
  Dieser Inhalt bleibt erhalten - es wird kein Makro verwendet.
</p>
```

Dieser Code führt zu demselben Ergebnis wie die Verwendung von `default` mit `tal:content` und `tal:replace`; das Anweisungselement bleibt unverändert.

Der Versuch, die Variable `nothing` mit `metal:use-macro` zu verwenden, löst einen Fehler aus, da `nothing` kein Makro ist. Falls Sie `nothing` verwenden möchten, um ein Makro bedingt einzufügen, sollten Sie stattdessen die Anweisung `metal:use-macro` in eine `tal:condition`-Anweisung einschließen.

Bei der Umsetzung von Vorlagen behandelt Zope zuerst Makros und wertet dann TAL-Ausdrücke aus. Betrachten Sie z. B. das folgende Makro:

```
<p metal:define-macro="title"
   tal:content="template/title">
  Titel der Vorlage
</p>
```

Dieses Makro fügt den Titel der Vorlage ein, in der das Makro verwendet wird, *nicht* den der Vorlage, in der es definiert wird. Anders ausgedrückt: Verwendet man ein Makro, so ist das dasselbe, wie wenn man das Makro in eine Vorlage kopiert und diese dann umsetzt.

Wenn Sie die Option *Expand macros when editing* in der Ansicht *Edit* einer Seitenvorlage aktivieren, werden alle Makros, die Sie verwenden, im Quellcode der Seitenvorlage vollständig angezeigt. Dies ist das Standardverhalten von Zope und in aller Regel auch das gewünschte Verhalten, da man so eine vollständige und gültige Seite bearbeiten kann. Manchmal ist es jedoch komfortabler, die Makros während der Bearbeitung nicht vollständig anzuzeigen. Dies ist insbesondere dann der Fall, wenn man keinen WYSISYG-Editor verwendet, sondern im ZMI arbeitet. Um Makros nicht vollständig anzuzeigen, deaktiviert man einfach die entsprechende Option.

Slots

Noch nützlicher werden Makros dadurch, dass man ein Makro bei seinem Einsatz teilweise überschreiben kann. Hierfür definieren Sie so genannte *Slots* im Makro, die Sie dann, wenn Sie die Vorlage verwenden, füllen können. Betrachten Sie z. B. das folgende Makro für eine Seitenleiste:

```
<p metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Produkte</a></li>
    <li><a href="/support">Kundendienst</a></li>
    <li><a href="/contact">Kontakt</a></li>
  </ul>
</p>
```

Dieses Makro ist an sich zwar gut, ermöglicht aber keine Abweichungen. Möchten Sie nun in der Seitenleiste zusätzliche Informationen bereitstellen, so besteht eine Lösungsmöglichkeit darin, Slots zu definieren:

```
<p metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Produkte</a></li>
    <li><a href="/support">Kundendienst</a></li>
    <li><a href="/contact">Kontakt</a></li>
  </ul>
  <span metal:define-slot="additional_info"></span>
</p>
```

Mit diesem Makro können Sie den Slot folgendermaßen füllen:

```
<p metal:fill-slot="container/master.html/macros/sidebar">
  <b metal:fill-slot="additional_info">
    Beachten Sie auch unsere <a
href="/specials">Sonderaktionen</a>.
  </b>
</p>
```

Wenn Sie diese Vorlage umsetzen, enthält die Seitenleiste die Zusatzinformation, die Sie im Slot bereitstellen:

```
<p>
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Produkte</a></li>
    <li><a href="/support">Kundendienst</a></li>
    <li><a href="/contact">Kontakt</a></li>
  </ul>
  <b>
    Beachten Sie auch unsere<a href="/specials">Sonderaktionen</a>.
  </b>
</p>
```

Achten Sie darauf, wie das Element `span`, das den Slot definiert, durch das Element `b` ersetzt wird, das den Slot füllt.

Anpassung der Standarddarstellung

Normalerweise dienen Slots dafür, eine Standarddarstellung bereitzustellen, die dann angepasst werden kann. Im obigen Beispiel besteht die Definition des Slots einfach aus einem leeren `span`-Element. Allerdings können Sie in einer Slot-Definition auch eine Standarddarstellung angeben. Betrachten Sie z. B. die folgende Version des Makros für die Seitenleiste:

```
<div metal:define-macro="sidebar">
  <p metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Produkte</a></li>
    <li><a href="/support">Kundendienst</a></li>
    <li><a href="/contact">Kontakt</a></li>
  </ul>
  </p>
  <span metal:define-slot="additional_info"></span>
</div>
```

Jetzt können alle Bestandteile der Seitenleiste individuell angepasst werden. Sie können den Slot `links` so ausfüllen, dass die Seitenleisten-Links neu definiert werden. Füllen Sie ihn jedoch nicht aus, so werden in diesem Slot die Standard-Links angezeigt.

Mit demselben Verfahren können Sie auch Slots in Slots definieren und damit die Standarddarstellung durch eine detailliertere Ausführung überschreiben. Hier sehen Sie noch einmal das Makro für die Seitenleiste, wobei diesmal jedoch innerhalb der Slots weitere Slots definiert werden:

```
<div metal:define-macro="sidebar">
  <p metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Produketes</a></li>
    <li><a href="/support">Kundendienst</a></li>
    <li><a href="/contact">Kontakt</a></li>
    <span metal:define-slot="additional_links"></span>
  </ul>
  </p>
  <span metal:define-slot="additional_info"></span>
</div>
```

Möchten Sie die Links in der Seitenleiste anpassen, so können Sie entweder den Slot `links` so füllen, dass er die Links vollständig überschreibt, oder den Slot `additional_links` so füllen, dass nach den Standard-Links noch weitere Links eingefügt werden. Slots können beliebig tief geschachtelt werden.

METAL und TAL kombinieren

METAL- und TAL-Anweisungen können auf dieselben Elemente angewandt werden:

```
<ul metal:define-macro="links"
    tal:repeat="link here/getLinks">
  <li>
    <a href="link url"
       tal:attributes="url link/url"
       tal:content="link/name">Link-Name</a>
  </li>
</ul>
```

Da METAL-Anweisungen vor TAL-Anweisungen ausgewertet werden, gibt es keine Konflikte. Dieses Beispiel ist auch insofern interessant, als es ein Makro ohne Slots anpasst. Das Makro ruft das Skript `getLinks` auf, um die Links zu ermitteln. Somit können Sie die Links Ihrer Site anpassen, indem Sie das Skript `getLinks` an verschiedenen Speicherorten Ihrer Web Site anpassen.

Nicht immer ist es einfach, das beste Verfahren für die Anpassung der Darstellung verschiedener Bereiche einer Site zu finden. Im Allgemeinen empfiehlt es sich, Darstellungselemente mit Slots zu überschreiben und dynamischen Inhalt mit Skripten bereitzustellen. In manchen Fällen, wie etwa dem oben gezeigten Link-Beispiel, ist nicht klar, ob es sich bei einem Seitenelement um Inhalt oder Darstellung handelt. Skripte sind in diesem Fall wahrscheinlich die flexiblere Lösung, besonders wenn die Site Link-Inhaltsobjekte enthält.

Eine Seite als Makro

Nicht nur einzelne Darstellungselemente, die von mehreren Seiten verwendet werden, können mit Makros definiert werden, sondern auch ganze Seiten. Ermöglicht wird dies durch Slots. Das folgende Beispiel zeigt ein Makro, das eine ganze Seite definiert:

```
<html metal:define-macro="page">
  <head>
    <title tal:content="here/title">Der Titel</title>
  </head>

  <body>
    <h1 metal:define-slot="headline"
        tal:content="here/title">Titel</h1>

    <p metal:define-slot="body">
      Dies ist der Rumpf.
    </p>

    <span metal:define-slot="footer">
      <p>Copyright 2001 Fluffy Enterprises</p>
    </span>

  </body>
</html>
```

Dieses Makro definiert eine Seite mit drei Slots: `headline`, `body` und `footer`. Achten Sie darauf, dass der Slot `headline` eine TAL-Anweisung enthält, die den Inhalt der Überschrift dynamisch ermittelt.

Dieses Makro können Sie in Vorlagen für verschiedene Arten von Inhalt oder für verschiedene Teile einer Site verwenden. In einer Vorlage für Nachrichten könnte das Makro zum Beispiel folgendermaßen eingesetzt werden:

```
<html metal:use-macro="container/master.html/macros/page">

  <h1 metal:fill-slot="headline">
    Pressemeldung:
    <span tal:replace="here/getHeadline">Überschrift</span>
  </h1>

  <p metal:fill-slot="body"
     tal:content="here/getBody">
    Hier folgt die Meldung selbst.
  </p>

</html>
```

Diese Vorlage definiert den Slot `headline` neu, und zwar so, dass er den Text "Pressemeldung" enthält und die Methode `getHeadline` auf dem aktuellen Objekt aufruft. Außerdem definiert sie den Slot `body` so, dass er die Methode `getBody` auf dem aktuellen Objekt aufruft.

Dieses Verfahren ist deshalb so leistungsfähig, weil damit bei einer Änderung des Makros `page` automatisch auch die Vorlage für die Pressemeldung aktualisiert wird. Sie können z. B. den `body` der Seite in eine Tabelle einfügen und links davon eine Seitenleiste einrichten; die Vorlage für die Pressemeldung verwendet dann automatisch diese neuen Darstellungselemente.

Damit bietet dieses Verfahren weitaus flexiblere Möglichkeiten, das Look&Feel einer Seite zu steuern, als die DTML-Lösung mit `standard_html_header` und `standard_html_footer`. Im Wurzelordner von Zope finden Sie eine Seitenvorlage namens `standard_template.pt`, die ein Seitenmakro mit einem `head`- und einem `body`-Slot enthält. Das folgende Beispiel veranschaulicht eine Möglichkeit, dieses Makro in einer Vorlage zu verwenden:

```
<html metal:use-macro="here/standard_template.pt/macros/page">
  <div metal:fill-slot="body">
    <h1 tal:content="here/title">Titel</h1>
    <p tal:content="here/getBody">Hier folgt der Textkörper.</p>
  </div>
</html>
```

Das Makro `standard_template.pt` wird ähnlich verwendet wie andere Seitenmakros. Eine Besonderheit stellt lediglich der Pfad zum Makro dar. Im obigen Beispiel beginnt der Pfad mit `here`. Dies bedeutet, dass Zope mit Hilfe der Akquisition nach dem Objekt `standard_template.pt` sucht, wobei es bei dem Objekt zu suchen beginnt, auf das die Vorlage angewandt wird. Dadurch können Sie die Darstellung von Vorlagen anpassen, indem

Sie an verschiedenen Speicherorten individuell angepasste `standard_template.pt`-Objekte erstellen. Dieses Verfahren entspricht also dem Anpassen der Darstellung durch Überschreiben von `standard_html_header` und `standard_html_footer` an einzelnen Stellen einer Web Site. Bei `standard_template.pt` stehen Ihnen allerdings mehr Möglichkeiten zur Verfügung. Sie können den Pfad zum Makro nicht nur mit `here` beginnen, sondern auch mit `root` oder `container`. Beginnt der Pfad mit `root`, so erhalten Sie immer die Standardvorlage, die sich im Wurzelordner befindet. Beginnt der Pfad mit `container`, so sucht Zope die Standardvorlage mit Hilfe der Akquisition und beginnt dabei in dem Ordner, in dem die Vorlage definiert ist. Dies erlaubt zwar die Anpassung des Look&Feel von Vorlagen, nicht aber die Anpassung des Look&Feel unterschiedlicher Objekte anhand ihres Speicherorts innerhalb der Site.

Vorlagen zwischenspeichern

Normalerweise werden Seitenvorlagen zwar relativ schnell umgesetzt, manchmal aber nicht schnell genug. Bei Seiten, auf die häufig zugegriffen wird oder bei denen der Seitenaufbau lange dauert, lohnt es sich, eine höhere Geschwindigkeit auf Kosten des dynamischen Verhaltens zu erzielen. Zu erreichen ist dieser Kompromiss durch Puffern, d. h. Zwischenspeichern im Cache. Weitere Informationen zum Puffern finden Sie im Abschnitt "Pufferung zur Leistungssteigerung" in Kapitel 3, "Verwendung grundlegender Zope-Objekte".

Seitenvorlagen können genau wie andere Objekte mit einem Cache-Manager gepuffert werden. Um eine Seitenvorlage zu puffern, müssen Sie sie mit einem Cache-Manager verbinden. Hierfür wählen Sie entweder in der Ansicht *Cache* der betreffenden Seitenvorlage den Cache-Manager oder in der Ansicht *Associate* des Cache-Managers die gewünschte Seitenvorlage.

Das folgende Beispiel veranschaulicht das Puffern einer Seitenvorlage. Zuerst erstellen Sie den Python-basierten Skriptnamen `long.py` mit dem folgenden Inhalt:

```
## Script (Python) "long.py"
##
for i in range(500):
  for j in range(500):
    for k in range(5):
      pass
return 'Done'
```

Der Zweck dieses Skript besteht darin, viel Ausführungszeit in Anspruch zu nehmen. Im nächsten Schritt legen Sie eine Seitenvorlage an, die dieses Skript verwendet; zum Beispiel:

```
<html>
  <body>
    <p tal:content="here/long.py">results</p>
  </body>
</html>
```

Wenn Sie sich diese Seite nun ansehen, werden Sie feststellen, dass ihre Umsetzung einige Zeit dauert. Als Nächstes werden wir die Umsetzung durch Pufferung erheblich beschleunigen. Falls Sie noch keinen Ram-Cache-Manager haben, erstellen Sie ihn jetzt, und zwar unbedingt entweder in demselben Ordner, in dem sich auch die Seitenvorlage befindet, oder auf einer höheren Ebene. Gehen Sie zur Ansicht *Cache* der Seitenvorlage, wählen Sie dort den Ram-Cache-Manager, den Sie gerade erstellt haben, und klicken Sie *Save Changes* an. Anschließend klicken Sie den Link *Cache Settings* an, um zu den Konfigurationseinstellungen des Ram-Cache-Managers zu gelangen. In der Standardeinstellung speichert der Cache Objekte für eine Stunde (3600 Sekunden). Diesen Zeitraum sollten Sie an Ihre Anwendung anpassen. Kehren Sie nun zur Seitenvorlage zurück. Nach wie vor dauert es einige Zeit, bis die Vorlage umgesetzt wird. Wenn Sie die Seite nun jedoch erneut laden, wird sie sofort umgesetzt. Sie können die Seite neu laden, so oft Sie möchten; sie wird immer sofort umgesetzt, da die Seite nun im Cache gespeichert ist.

Ändern Sie die Seitenvorlage, so wird sie aus dem Cache entfernt. Wenn Sie sie dann erneut aufrufen, dauert die Umsetzung wieder einige Zeit. Da sie dabei aber im Cache gespeichert wird, wird sie bei allen folgenden Aufrufen wieder schnell aufgebaut.

Das Puffern ist ein einfaches, aber leistungsstarkes Verfahren, um die Leistung zu steigern. Es ist keine Hexerei, steigert die Geschwindigkeit aber beträchtlich. Bei Anwendungen, bei denen die Leistung eine wichtige Rolle spielt, lohnt sich das Puffern auf jeden Fall.

Hilfprogramme für Seitenvorlagen

Zope-Seitenvorlagen sind leistungsstark, aber einfach. Im Gegensatz zu DTML bieten Seitenvorlagen kaum Hilfsmittel für die Stapelung, das Anlegen von Bäumen, Sortieren usw. Die Erfinder der Seitenvorlagen wollten diese einfach halten. Allerdings vermisst man doch einige der integrierten Funktionen, die DTML bietet. Um diese Lücke zu schließen, umfasst Zope Hilfsprogramme, die die Seitenvorlagen verbessern sollen.

Große Informationsmengen stapeln

Wenn ein Anwender eine Datenbank abfragt und Hunderte von Ergebnissen erhält, so ist es häufig sinnvoller, diese Ergebnisse auf mehrere Seiten zu verteilen (z. B. 20 Ergebnisse pro Seite), als alle Ergebnisse auf einer einzigen Seite auszugeben. Das Zerlegen großer Listen in mehrere kleine wird als *Stapelung* bezeichnet.

Im Gegensatz zu DTML, bei dem die Stapelung direkt in die Programmiersprache integriert ist, unterstützen Seitenvorlagen diese durch ein besonderes Objekt namens `Batch`, das sich im Hilfsmodul `ZTUtils` befindet. Weitere Informationen zu dem Python-Modul `ZTUtils` finden Sie im Anhang B, "API-Referenz".

Das folgende einfache Beispiel zeigt, wie ein `Batch`-Objekt erstellt wird:

```
<ul tal:define="lots python:range(100);
                batch python:modules['ZTUtils'].Batch(lots,
                                                      size=10,
                                                      start=0)">
  <li tal:repeat="num batch"
      tal:content="num">0
  </li>
```

```
        </ul>
```

Dieses Beispiel setzt eine Liste mit 10 Elementen um (in diesem Fall die Zahlen von 0 bis 9). Das Objekt `Batch` zerschneidet eine lange Liste in Gruppen oder Batches. In diesem Fall teilt es eine aus hundert Elementen bestehende Liste in Gruppen zu je zehn Elementen.

Um eine Gruppe von zehn Elementen anzuzeigen, übergeben Sie eine andere Anfangszahl:

```
        <ul tal:define="lots python:range(100);
                        batch python:modules['ZTUtils'].Batch(lots,
                                                              size=10,
                                                              start=13)">
```

Dieser Batch beginnt mit dem 14. und endet mit dem 23. Element. Anders ausgedrückt: Er zeigt die Zahlen zwischen 13 und 22 an. Beachten Sie, dass das Argument `start` des Batches der *Index* des ersten Elementes ist. Indizes beginnen bei 0 zu zählen, nicht bei 1. Der Index 13 ist demnach das 14. Element der Folge. Python verweist mit Indizes auf Listenelemente.

Arbeitet man mit Batches, so ist es sinnvoll, Navigationselemente in die Seite einzufügen, damit der Anwender von Batch zu Batch wechseln kann. Das folgende vollständige Batch-Beispiel veranschaulicht die Navigation zwischen einzelnen Gruppen.

```
        <html>
          <head>
            <title tal:content="template/title">Der Titel</title>
          </head>
          <body tal:define="employees  here/getEmployees;
                            start python:path('request/start') or 0;
                            batch
python:modules['ZTUtils'].Batch(employees,
                                                                  size=10,

start=start);
                            previous python:batch.previous;
                            next python:batch.next">

            <p>
              <a tal:condition="previous"
                 tal:attributes="href
string:${request/URL0}?start:int=${previous/first}"
                 href="previous_url">zurück</a>
              <a tal:condition="next"
                 tal:attributes="href
string:${request/URL0}?start:int=${next/first}"
                 href="next_url">weiter</a>
            </p>

            <ul tal:repeat="employee batch" >
              <li>
                <span tal:replace="employee/name">Hans Müller</span>
                verdient $<span tal:replace="employee/salary">100.000</span>
                im Jahr.
              </li>
            </ul>
```

```
  </body>
</html>
```

Dieses Beispiel iteriert über Ergebnisgruppen aus der ZSQL-Methode `getEmployees`. Es zeichnet die Links *zurück* und *weiter*, um das Blättern durch die Ergebnisgruppen zu ermöglichen.

Sehen Sie sich die Anweisung `tal:define` im Element `body` etwas genauer an. Sie definiert mehrere Batch-Variablen. Die Variable `employees` ist eine möglicherweise lange Liste von employee-Objekten, die von der ZSQL-Methode `getEmployees` zurückgegeben wird. Die zweite Variable, `start`, ist entweder auf den Wert von `request/start` oder, falls die Anfrage keine `start`-Variable enthält, auf null gesetzt. Die Variable `start` verfolgt, an welcher Stelle der Angestelltenliste Sie sich gerade befinden. Die Variable `batch` ist eine Gruppe von zehn Elementen aus der Angestelltenliste. Die Gruppe beginnt an der von der Variable `start` angegebenen Position. Die Variablen `previous` und `next` verweisen auf die vorhergehende und die folgende Gruppe (soweit vorhanden). Da alle diese Variablen im Element `body` definiert sind, stehen sie allen Elementen im Body zur Verfügung.

Sehen wir uns nun die Navigations-Links an. Diese erstellen Hyperlinks, die das Blättern zur vorhergehenden und zur folgenden Gruppe ermöglichen. Die Anweisung `tal:condition` prüft zuerst, ob es eine vorhergehende und eine folgende Gruppe gibt. Ist dies der Fall, so wird der Link dargestellt, anderenfalls nicht. Die Anweisung `tal:attributes` erstellt einen Link zur vorhergehenden und zur folgenden Gruppe. Der Link besteht einfach aus dem URL der aktuellen Seite ('request/URL0') und einer Anfragezeichenkette, die den Anfangsindex der Gruppe angibt. Beginnt die aktuelle Gruppe zum Beispiel mit dem Index 10, so muss die vorhergehende Gruppe mit dem Index 0 beginnen. Die Variable `first` einer Gruppe gibt deren Anfangsindex an, in diesem Fall ist `previous.start` also 0.

Sie brauchen nicht alle Einzelheiten dieses Beispiels zu verstehen. Kopieren Sie es einfach oder verwenden Sie ein vom *Z Search Interface* erstelltes Stapelungsbeispiel. Wenn Sie später komplexere Batch-Aufgaben lösen möchten, können Sie mit diesem Beispielcode experimentieren. In Anhang B, "API-Referenz" finden Sie weitere Informationen zum Modul `ZTUtils` sowie zu `Batch`-Objekten.

Verschiedene Hilfsprogramme

Zope bietet eine Reihe von Python-Modulen, die die Arbeit mit Seitenvorlagen erleichtern. Die Module `string`, `math` und `random` können in Python-Ausdrücken für die Formatierung von Zeichenketten, für mathematische Funktionen und für die Erzeugung von Zufallszahlen verwendet werden. Diese Module stehen in Python-basierten und DTML-Skripten gleichermaßen zur Verfügung. Weitere Informationen dazu finden Sie in Anhang B, "API-Referenz".

Das Modul `Products.PythonScripts.standard` soll Hilfsprogramme für Python-basierte Skripte bereitstellen, ist aber auch bei der Arbeit mit Seitenvorlagen sinnvoll. Es umfasst verschiedene Funktionen für die Formatierung von Zeichenketten und Zahlen. Weitere Informationen dazu finden Sie in Anhang B, "API-Referenz".

Wie in diesem Kapitel bereits erwähnt wurde, stellt das Modul `sequence` die nützliche Funktion `sort` bereit. Einzelheiten dazu finden Sie in Anhang B, "API-Referenz".

Das Modul `AccessControl` enthält eine Funktion und eine Klasse, die Sie für die Prüfung des Zugriffs sowie für das Abrufen berechtigter Anwender benötigen. Weitere Informationen dazu finden Sie in Anhang B, "API-Referenz".

Zusammenfassung

Dieses Kapitel behandelt die Einzelheiten von Seitenvorlagen, und stiftet zunächst vielleicht vor allem Verwirrung. Aber keine Sorge: Um Seitenvorlagen effizient einzusetzen, brauchen Sie nicht alles zu wissen, was hier angesprochen wurde. Wichtig sind die verschiedenen Pfadtypen und Makros. Die übrigen Informationen, die komplizierteren Aspekte, die Sie in diesem Kapitel kennen gelernt haben, können Sie auch später jeder Zeit nachlesen, wenn Sie sie tatsächlich benötigen. Doch es ist beruhigend, zu wissen, dass Ihnen dann, wenn Sie soweit sind, mit Seitenvorlagen beeindruckende Möglichkeiten offenstehen.

Zopebuch: Inhaltsverzeichnis

Kapitel 10: Zope-Scripting für Fortgeschrittene

Zope verwaltet ihre Präsentation, Logik und Daten durch Objekte. Bis hierher haben Sie gesehen, wie Zope Präsentation über DTML und Daten über Dateien und Bilder verwalten kann. Dieses Kapitel zeigt Ihnen, wie Sie Skript-Objekte hinzufügen können, was Ihnen erlaubt, mit ihrem Browser Skripte in Python und Perl zu schreiben.

Was ist *Logik* und wie unterscheidet sie sich von der Präsentation? Logik liefert die Aktionen, die Objekte ändern, Nachrichten senden, Bedingungen prüfen und auf Ereignisse antworten, während die Präsentation Informationen und Meldungen anzeigt und formatiert. Typischerweise werden Sie DTML benutzen, um ihre Präsention zu bearbeiten und Zope-Skripting mit Python und Perl, um Logik zu bearbeiten.

Zope-Skripte

Zope-*Skript*-Objekte sind Objekte, die ein kleines bißchen Code enthalten, der in einer Programmiersprache geschrieben ist. Gegenwärtig bietet Zope *Python-basierte Skripte*, die in der Programmiersprache Python geschrieben sind, und *Perl-Skripte*, die in der Programmiersprache Perl geschrieben sind. Skript-Objekte sind seit Zope 2.3 eingeführt worden und stellen die bevorzugte Weise dar, Programm-Logik in Zope zu schreiben.

Bisher haben sie in diesem Buch starken Gebrauch von DTML-Skripten und -Dokumenten gemacht, um einfache Anwendungen in Zope herzustellen. DTML erlaubt Ihnen, einfache Skript-Operationen wie String-Auswertungen durchzuführen. DTML-Skripte sollten jedoch hauptsächlich zur Präsentation angewendet werden. In Kapitel 4 (Dynamische Inhalte mit DTML) und Kapitel 8 (Variablen und fortgeschrittenes DTML) werden DTML-Skripte erklärt.

Es folgt ein Überblick über die Skripte von Zope:

Python-basierte Skripte
: Sie können Python - eine Allzweck-Skriptsprache - benutzen, um Zope-Objekte zu kontrollieren und andere Aufgaben durchzuführen. Diese Skripte geben Ihnen universelle Programmierungsmöglichkeiten innerhalb von Zope.

Perl-basierte Skripte
: Sie können Perl - eine mächtige textverarbeitende Programmiersprache - benutzen, um Zope-Objekte zu verarbeiten und Perl-Bibliotheken einzubinden. Diese Skripte bieten ähnliche Vorteile wie Python-basierte Skripte, mögen aber für Leute angenehmer sein, die mit Perl eher vertraut sind als mit Python oder die Perl-Bibliotheken verwenden wollen, für die es kein Python-Pendant gibt.

Sie können diese Skripte ihrer Zope-Anwendung genau wie jedes andere Objekt hinzufügen.

Skripte aufrufen

Zope-Skripte werden über das Web oder andere Skripte oder Objekte aufgerufen. Fast jede Art von Skript kann von jeder anderen Objekt-Art aufgerufen werden; Sie können ein Python-basiertes Skript von einem DTML-Skript oder ein eingebautes Skript von einem Perl-basierten Skript aufrufen lassen. Tatsächlich können Skripte wiederum Skripte aufrufen, die wieder andere Skripte aufrufen und so weiter. Wie Sie in Kapitel 4 (Dynamische Inhalte mit DTML) gesehen haben, können sie ohne weiteres ein Skript durch ein Skript ersetzen, das in einer anderen Programmiersprache geschrieben ist. Wenn Sie beispielsweise Perl benutzen, um eine Aufgabe auszuführen, sich aber später entschließen, daß dasselbe besser in Python getan werden sollte, können Sie normalerweise das Skript mit einem Python-basierten Skript mit derselben `id` ersetzen.

Wenn Sie ein Skript aufrufen, übergibt die Weise, auf die Sie es aufrufen, den Kontext, in dem es ausgeführt werden soll. Der Kontext eines Skriptes ist wichtig. Wenn Sie zum Beispiel ein Skript aufrufen, gibt es gewöhnlich ein bestimmtes Objekt, welches das Zentrum der Aufgabe des Skripts ist. Das bedeutet, Sie rufen das Skript im Kontext des Objekts auf, an dem Sie die Aufgabe ausführen wollen. Wir sagen vereinfacht, daß wir das Skript *auf* dem Objekt aufrufen.

Skripte aus dem Web aufrufen

Sie können ein Skript direkt aus dem Web aufrufen, indem Sie seinen URL aufrufen. Sie können ein einziges Skript über verschiedene Objekte aufrufen, indem Sie verschiedene URLs benutzen. Das funktioniert, weil Sie über die Benutzung verschiedener URLs dem Skript verschiedene Kontexte übergeben und Skripte kontextabhängig unterschiedlich operieren können. Das ist eine mächtige Möglichkeit, die ihnen Gelegenheit gibt, Logik auf Objekte wie Dokumente oder Ordner anzuwenden, ohne daß Sie den Code dazu im Objekt einbetten müssen.

Um ein Skript auf einem Objekt aus dem Web aufzurufen, besuchen Sei einfach den URL des Objekts, gefolgt vom Namen des Skripts. Das setzt das Skript in den Kontext des Objekts. Nehmen Sie beispielsweise an, Sie hätten eine Sammlung von Objekten und Skripten wie in Abbildung 8.1 dargestellt.

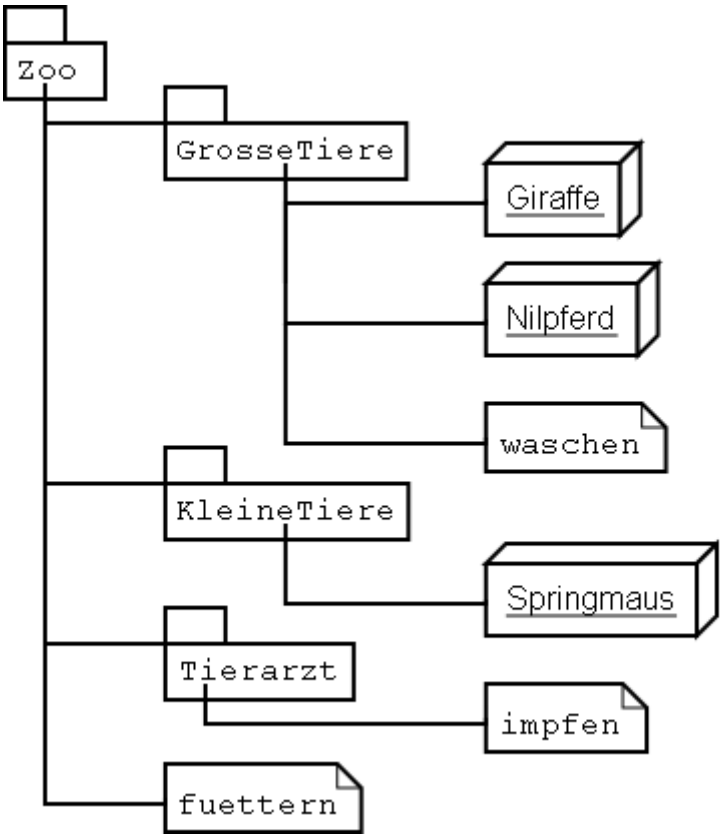


Abbildung 8.1 Eine Sammlung von Objekten und Skripten

Um das Skript *fuettern* auf dem Objekt *Nilpferd* aufzurufen, würden Sie den URL *Zoo/GrosseTiere/Nilpferd/fuettern* besuchen. Um das Skript *fuettern* auf dem Objekt *Springmaus* aufzurufen, können Sie den URL *Zoo/KleineTiere/Springmaus/fuettern* besuchen. Diese URLs bringen das Skript *fuettern* in den Kontext der Objekte *Nilpferd* oder *Springmaus*.

Zope benutzt den URL als eine Landkarte, um herauszufinden, welches Objekt und welches Skript Sie aufrufen wollen.

Zope zerlegt den URL und vergleicht ihn mit der Objekthierarchie, indem es sich rückwärts durcharbeitet, bis es eine Übereinstimmung für jeden Teil findet. Dieser Prozeß wird *URL-Traverse* (URL traversal) genannt. Wenn Sie Zope zum Beispiel den URL *Zoo/GrosseTiere/Nilpferd/fuettern* angeben, beginnt es beim Root-Ordner und schaut nach einem Objekt namens *Zoo*. Dann bewegt es sich zum Order *Zoo* und schaut dann nach einem Objekt namens *GrosseTiere*. Es bewegt sich zum Ordner *GrosseTiere* und sucht nach einem Objekt, das *Nilpferd* heißt. Es geht zum Objekt *Nilpferd* und sucht ein Objekt namens *fuettern*. Das Skript *fuettern* kann im Objekt *Nilpferd* nicht gefunden werden, aber Zope findet es aufgrund eines Vorgangs, der *Erwerbung* (acquisition) heißt, im Ordner *Zoo*.

"Erwerbung" tut zwei Dinge. Zuerst versucht es, das gesuchte Objekt in den Ordnern des aktuell aufgerufenen Objekts zu finden. Wenn das nicht klappt, zieht es sich entlang des URL-Pfades zurück und versucht es währenddessen weiter. In diesem Beispiel sucht Zope zunächst in *Nilpferd* nach dem Objekt *fuettern*, dann geht es zum ersten Ordner, *GrosseTiere*, und dann zum nächsten Ordner, *Zoo*, wo schließlich *fuettern* gefunden wird.

Nun hat Zope das Ende des URLs erreicht. Es ruft das letzte gefundene Objekt auf: *fuettern*. Das Skript *fuettern* handelt in seinem Kontext, der auf dem vorletzten gefundenen Objekt beruht, dem Objekt *Nilpferd*. So wird das Skript *fuettern* auf dem Objekt *Nilpferd* aufgerufen.

Genauso können Sie das Skript *waschen* mit dem URL *Zoo/GrosseTiere/Nilpferd/waschen* auf dem Objekt *Nilpferd* aufrufen. In diesem Fall erwirbt Zope das Skript *waschen* aus dem Ordner *GrosseTiere*.

Es sind auch komplexere Anordnungen möglich. Angenommen, Sie wollen das Skript *impfen* auf dem Objekt *Nilpferd* aufrufen. Welchen URL können Sie benutzen? Wenn Sie über den URL *Zoo/GrosseTiere/impfen* vorgehen, wird Zope das Skript *impfen* nicht finden können, weil es sich in keinem von *Nilpferd*s Behältern befindet.

Die Lösung besteht darin, den Pfad zum Skript als Teil des URL einzugeben. So wird Zope das richtige Skript finden, wenn es Erwerbung benutzt, um das Skript zu finden, während es den URL zurückverfolgt. Der URL zur Impfung des Nilpferds ist *Zoo/Tierarzt/GrosseTiere/Nilpferd/impfen*. Genauso sollten Sie den URL *Zoo/Tierarzt/KleineTiere/Springmaus/impfen* benutzen, wenn Sie das Skript *impfen* auf die *Springmaus* anwenden wollen.

Lassen Sie uns mitverfolgen, wie Zope den URL *Zoo/Tierarzt/GrosseTiere/Nilpferd/impfen* durchläuft. Zope beginnt im Root-Ordner und sucht nach einem Objekt namens *Zoo*. Es bewegt sich zum Ordner *Zoo* und schaut nach einem Objekt, das *Tierarzt* heißt. Es geht zum Ordner *Tierarzt* und sucht nach einem Objekt namens *GrosseTiere*. Der Ordner *Tierarzt* enthält kein Objekt mit diesem Namen, aber es kann den Ordner *GrosseTiere* über den Behälter *Zoo* ansprechen. Also bewegt es sich zum Ordner *GrosseTiere* und schaut nach einem Objekt *Nilpferd*. Dann geht es zum Objekt *Nilpferd* und sucht dort nach dem Objekt *impfen*. Weil weder das Objekt *Nilpferd* noch einer seiner Behälter ein Objekt namens *impfen* enthält, geht Zope den Pfad zurück und versucht dabei, das Objekt *impfen* zu finden. Zuerst zieht es sich zum Ordner *GrosseTiere* zurück, wo *impfen* immer noch nicht gefunden werden kann. Dann geht es zurück zum Ordner *Tierarzt*. Hier findet es das Skript *impfen*. Weil Zope nun zum Ende des aufgerufenen URL gekommen ist, wendet es das Skript *impfen* im Kontext des Objekts *Nilpferd* an.

Wenn Zope während einer URL-Traverse nach einem Sub-Objekt sucht, schaut es zuerst im aktuellen Objekt nach dem Sub-Objekt. Wenn es im aktuellen Objekt das Sub-Objekt nicht finden kann, schaut es in den Behältern des aktuellen Objekts. Wenn es das Sub-Objekt dann immer noch nicht finden kann, zieht es sich entlang des URL-Pfades zurück und sucht erneut. Es verfolgt diesen Prozeß, bis es entweder das Objekt findet oder einen Fehler meldet, wenn es nicht gefunden werden kann.

Das ist ein sehr nützlicher Mechanismus und er erlaubt Ihnen, recht ausdrucksvoll zu sein, wenn Sie URLs zusammenstellen. Der Pfad, den Sie Zope mitteilen, damit es seinen Weg zu einem Objekt findet, wird bestimmen, wie es Erwerbung benutzt, um die Skripte für das Objekt zu finden.

Skripte von anderen Objekten aufrufen

Sie können Skripte anderer Objekte aufrufen. Zum Beispiel ist es üblich, Skripte von DTML-Skripten aufzurufen.

Wie Sie in Kapitel 8 (Variablen und fortgeschrittenes DTML) gesehen haben, können Sie Zope-Skripte von DTML mit dem *call*-Tag aufrufen. Zum Beispiel:

```
<dtml-call aktualisiereInfo>
```

DTML wird das Skript *aktualisiereInfo* aufrufen. Sie brauchen nicht zu spezifizieren, ob das Skript in Perl, Python oder irgendeiner anderen Programmiersprache implementiert ist (Sie können genausogut andere DTML-Objekte oder SQL-Skripte auf diese Art aufrufen).

Wenn das Skript *aktualisiereInfo* Parameter braucht, müssen Sie entweder einen Namen für die DTML-Namensraum-Bindung (siehe unten: Bindende Variablen) auswählen, damit die Parameter im Namensraum nachgeschlagen werden können oder die Parameter in einem Ausdruck übergeben, wie hier:

```
<dtml-call expr="aktualisiereInfo(farbe='braun',
                                  muster='gefleckt')">
```

Skripte von Python oder von Perl aus aufzurufen, funktioniert auf dieselbe Art, nur daß Sie immer Skript-Parameter übergeben müssen, wenn Sie ein Python- oder Perl-Skript aufrufen. Hier ist ein Beispiel, wie Sie das Skript *aktualisiereInfo* aus Python aufrufen mögen:

```
context.aktualisiereInfo(farbe='braun',
                         muster='gefleckt')
```

Aus Perl könnten Sie dasselbe tun, indem Sie Standard-Perl-Semantik zum Aufrufen von Skripten benutzen:

```
$self->aktualisiereInfo(farbe => 'braun',
                        muster => 'gefleckt');
```

Jede Skript-Sprache hat ihre eigene Art, Skripte aufzurufen, es spielt aber keine Rolle, in welcher Sprache das aufgerufene Skript geschrieben ist, solange man die passenden Parameter übergibt.

Wenn ein Skript aufgerufen wird, wird es in wie beim Aufruf über das Web durch Erwerbung gesucht. Kehren wir zu unserem Nilpferd-fuettern-Beispiel aus dem letzten Absatz zurück und lassen Sie uns sehen, wie ein Nilpferd von Python oder Perl aus geimpft werden kann. Abbildung 8.2 zeigt eine leicht veränderte Objekthierarchie, die zwei Skripte enthält, *impfeNilpferd.py* und *impfeNilpferd.pl*.

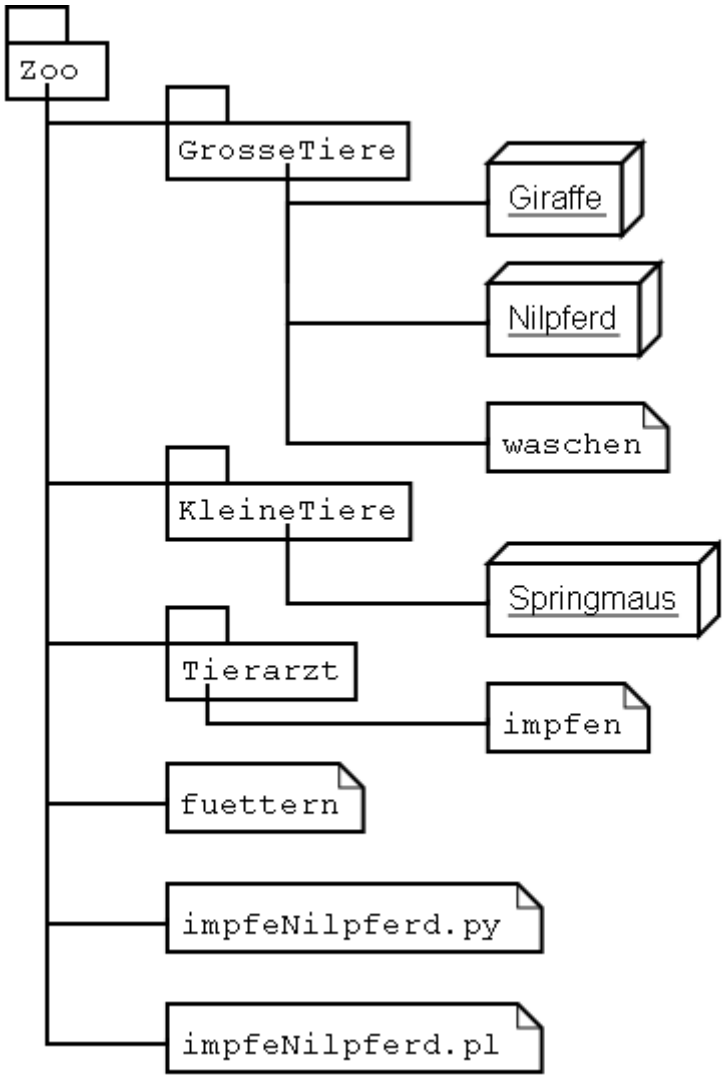


Figure 8-2 Eine Sammlung von Objekten und Skripten

Stellen sie sich vor, daß *impfeNilpferd.py* ein Python-Skript ist. Hier sehen Sie, wie Sie das Skript *impfen* auf dem Objekt *Nilpferd* von Python aus aufrufen:

```
context.Tierarzt.GrosseTiere.Nilpferd.impfen()
```

In anderen Worten: Sie greifen auf das Objekt zu, indem Sie denselben Erwerbungs-Pfad nutzen, den Sie gebrauchen würden, wenn Sie ihn vom Web aus aufrufen würden. Genauso könnten Sie in Perl sagen:

```
$self->Tierarzt->GrosseTiere->Nilpferd->impfen();
```

Skripte von anderen Skripten aus zu benutzen, ist dem Aufrufen von Skripten aus dem Web sehr ähnlich. Die Semantik ist etwas anders, aber die Erwerbungs-Regeln bleiben die

gleichen. Später in diesem Kapitel werden Sie mehr Beispiele sehen, wie Skripte in Perl oder Python funktionieren.

Parameter an Skripte übergeben

Alle Skripte können Parameter übergeben bekommen. Ein Parameter gibt einem Skript mehr Informationen darüber, was es tun soll. Wenn Sie ein Skript vom Web aus aufrufen, wird Zope versuchen, die Parameter für das Skript in der Web-Anfrage zu finden und sie an ihr Skript zu übergeben. Wenn Sie zum Beispiel ein Skript mit den Parametern *Delphin* und *REQUEST* haben, wird Zope in der Web-Anfrage nach *Delphin* suchen, und die Anfrage selbst als *REQUEST*-Parameter übergeben. Damit wird Formularverarbeitung in Skripten vereinfacht. Schauen Sie sich das folgende Formular an:

```
<form action="aktionsSkript">
  Name <input type="text" name="name"><br>
  Alter <input type="text" name="alter:int"><br>
  <input type="submit">
</form>
```

Ohne weitere Arbeit können Sie dieses Formular mit einem Skript namens *aktionsSkript* verarbeiten, das in seiner Parameter-Liste *name* and *alter* enthält:

```
## Script (Python) "aktionsSkript"
##parameters=name, alter
##
"Verarbeite Formular"
context.verarbeiteName(name)
context.verarbeiteAlter(alter)
return context.rueckmeldeNachricht()
```

Das erspart Ihnen, Werte explizit aus dem Formular zu nehmen. Im Beispiel werden die Formularelemente übrigens als Strings übergeben. Falls das Formular Check-Boxen enthält, oder Multiple-Selection Eingaben erlaubt, werden String-Listen erzeugt.

Zusätzlich zu den Formular-Variablen können Sie jede Anfrage-Varaible als Skript-Parameter festlegen. Um zum Beispiel auf die Anfrage- und Antwort-Objekte zuzugreifen, schließen Sie einfach `REQUEST` und `RESPONSE` in ihre Parameter-Liste ein. Anfrage-Variablen sind genauer beschrieben in Anhang B.

Eine bemerkenswerte Sache ist, daß die Variable *context* sich auf das Objekt bezieht, auf dem das Skript aufgerufen wurde. Das funktioniert in Perl-basierten Skripten ähnlich, zum Beispiel:

```
my $self = shift;
$self->verarbeiteName($name);
$self->verarbeiteAlter($alter);
return $context->rueckmeldeNachricht();
```

In der Python-Version des Beispiels gibt es ein subtiles Problem. Sie erwarten vielleicht eher einen Integer-Wert als einen String für das Alter. Sie könnten mit hilfe der in Python eingebauten *int* Methode den String manuell in ein Integer konvertieren:

```
alter=int(alter) # konvertiere einen String zu einem Integer
```

Aber diese manuelle Konvertierung könnte unbequem sein. Zope bietet Ihnen einen Weg, Formular-Eingabetypen im Formular selbst festzulegen, anstatt im verarbeitenden Skript. Statt die Variable *alter* im verarbeitenden Skript zu einem Integer zu konvertieren, können Sie im Formular bestimmen, daß es ein Integer ist:

```
Age <input type="text" name="alter:int">
```

Der Anhang `:int` am Namen des Formular-Inputs trägt Zope auf, den Formular-Input automatisch in einen Integer zu konvertieren. Wenn der Benutzer ihres Formulars etwas eingibt, das nicht in einen Integer konvertiert werden kann (etwas wie "22, werde bald 23"), wird Zope - wie in Abbildung 8.3 - einen Fehler melden.

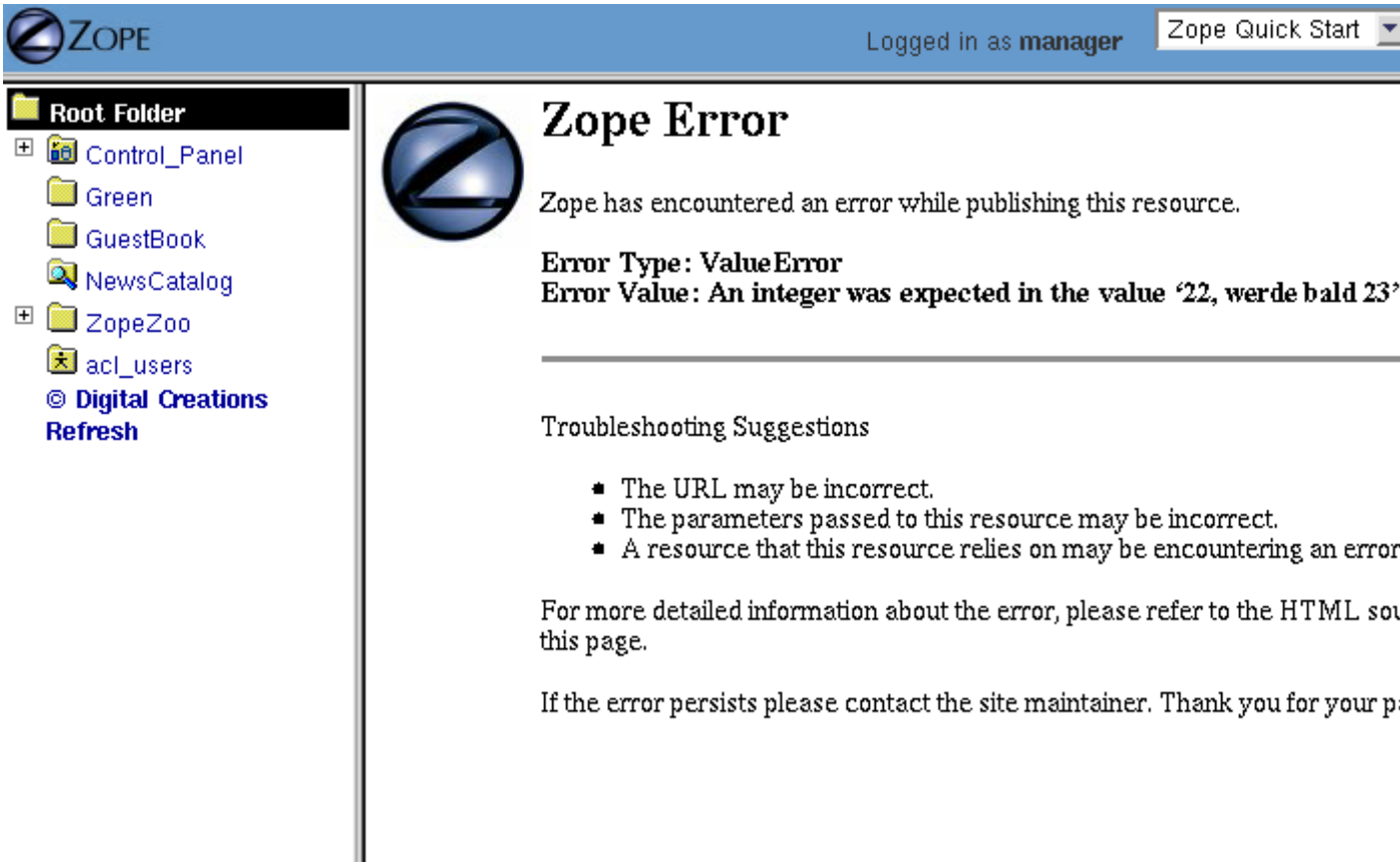


Figure 8.3 Parameter-Konvertierungsfehler

Es ist praktisch, daß Zope Konvertierungsfehler abfängt, aber vielleicht mögen Sie Zopes Fehlermeldungen nicht. Sie sollten die Benutzung von Zopes Konverter vermeiden, wenn Sie ihre eigenen Fehlermeldungen bereitstellen wollen.

Zope kann viele Parameter-Konvertierungen durchführen. Hier ist eine Liste von Zopes grundlegenden Parameter-Konvertern:

boolean
: Konvertiert eine Variablen nach wahr oder unwahr. Variablen, die 0, None, ein leerer String oder eine leere Sequenz sind, gelten als unwahr, alle anderen gelten als wahr.

int
: Konvertiert eine Variable zu einem Integer.

long
: Konvertiert eine Variable zu einem Langen Integer.

float
: Konvertiert eine Variable zu einer Fließkommazahl.

string
: Konvertiert eine Variable zu einem String. Die meisten Variablen sind schon Strings, daher wird dieser Konverter selten benutzt.

text
: Konvertiert eine Variable zu einem String mit normalisierten Zeilenumbrüchen. Verschiedene Browser auf verschiedenen Plattformen kodieren Zeilenenden auf verschiedene Weise, daher stellt dieses Skript sicher, daß die Zeilenenden konsistent sind, unabhängig davon, wie sie im Browser kodiert werden.

list
: Konvertiert eine Variable zu einer Python-Liste.

tuple
: Konvertiert eine Variable zu einem Python-Tupel. Ein Tupel ist wie eine Liste, kann aber nicht verändert werden.

tokens
: Konvertiert einen String zu einer Liste, indem die Leerzeichen als Trennzeichen angesehen werden

lines
: Konvertiert einen String zu einer Liste, indem die Zeilenumbrüche als Trennzeichen angesehen werden.

date
: Konvertiert einen String zu einem *DateTime*-Objekt. Die akzeptierten Formate sind ziemlich flexibel , zum Beispiel `10/16/2000`, `12:01:13 pm`.

required
: Gibt einen Fehler aus, wenn die Variable nicht gefüllt ist.

ignore_empty
: Schließt die Variable aus der Anfrage aus, wenn die Variable ein leerer String ist.

Diese Konverter arbeiten alle mehr oder weniger auf dieselbe Art, indem sie eine Variable in Gestalt eines Strings zu einem spezifischen Typ verarbeiten. Sie mögen diese Konverter aus dem Kapitel 3 (Grundlegende Zope-Objekte benutzen) wiedererkennen, wo wir Eigenschaften diskutiert haben. Diese Konverter werden von Zopes Eigenschafts-Anlagen benutzt, um Eigenschaften zum richtigen Typ zu konvertieren.

Die Konverter *list* und *tuple* können in Kombination mit anderen Konvertern benutzt werden. Das erlaubt Ihnen, jedem Element der Liste oder des Tupels zusätzliche Konverter zuzuweisen. Sehen Sie sich dieses Formular an:

```
      <form action="verarbeiteZeiten">
        <p>
        Ich würde es vorziehen, zu den folgenden Zeiten nicht gestört zu
werden:
        </p>
        <input type="checkbox" name="disturb_times:list:date" value="12:00
AM"> Mitternachts<br>
        <input type="checkbox" name="disturb_times:list:date" value="01:00
AM"> 1:00 morgens<br>
        <input type="checkbox" name="disturb_times:list:date" value="02:00
AM"> 2:00 morgens<br>
        <input type="checkbox" name="disturb_times:list:date" value="03:00
AM"> 3:00 morgens<br>
        <input type="checkbox" name="disturb_times:list:date" value="04:00
AM"> 4:00 morgens<br>
        <input type="submit" value="Abschicken">
      </form>
```

Wenn man die Konverter *list* und *date* gemeinsam benutzt, wird Zope jede ausgewählte Zeit in ein Datum konvertieren und dann alle ausgewählten Daten in einer Liste namens *disturb_times* kombinieren.

Eine komplexere Art der Konvertierung von Formularen ist, eine Serie von Eingaben zu Berichten (records) zu konvertieren. Berichte sind Strukturen, die Attribute haben. Bei der Benutzung von Berichten können Sie eine Anzahl von Formulareingaben in einer Variablen mit Attributen zusammenfassen. Die verfügbaren Berichts-Konverter sind:

record
: Konvertiert eine Variable zu einem Berichtsattribut.

records
: Konvertiert eine Variable zu einem Berichtsattribut in einer Liste von Berichten.

default
: Setzt einen voreingestellten Wert als Berichtsattribut ein, falls die Variable leer ist.

ignore_empty
: Ignoriert ein Berichtsattribut, wenn die Variable leer ist.

Hier sind einige Beispiele für die Anwendung dieser Konverter:

```
      <form action="verarbeitePerson">
        Vorname <input type="text" name="person.vname:record"><br>
        Nachname <input type="text" name="person.nname:record"><br>
        Alter <input type="text" name="person.alter:record:int"><br>
        <input type="submit" value="Abschicken">
      </form>
```

Dieses Formular ruft das Skript *verarbeitePerson* mit einem Parameter *person* auf. Die Variable *person* wird die Attribute *vname*, *nname* and *alter* haben. Hier ist ein Beispiel dafür, wie Sie die Variable *person* in ihrem Skript *verarbeitePerson* verwenden können:

```
      ## Skript (Python) "verarbeitePerson"
      ##parameters=person
      ##
      " einen Personen-Bericht verarbeiten "
      voller_name="%s %s" % (person.vname, person.nname)
      if person.alter < 21:
          return "Schade, %s. Sie sind nicht alt genug, um ein Erdferkel zu
adoptieren.
          " % full_name
      return "Danke, %s. Ihr Erdferkel ist auf dem Weg." % voller_name
```

Der Konverter *records* arbeitet wie der Konverter *record*, nur produziert er eine Liste von Berichten, nicht nur einen. Hier ist ein Beispielformular:

```
      <form action="verarbeiteMenschen">
        <p>
        Bitte geben Sie Informationen über einen oder mehrere ihrer
nächsten Verwandten ein.
        </p>
        <p>
        Vorname <input type="text" name="leute.vname:records">
        Nachname <input type="text" name="leute.nname:records">
        </p>
        <p>
        Vorame <input type="text" name="leute.vname:records">
        Nachname <input type="text" name="leute.nname:records">
        </p>
        <p>
        Vorame <input type="text" name="leute.vname:records">
        Nachname <input type="text" name="leute.nname:records">
        </p>
        <input type="submit" value="Abschicken">
      </form>
```

Dieses Formular wird das Skript *verarbeiteMenschen* mit einer Variablen namens *people* aufrufen, die eine Liste von Berichten ist. Jeder Bericht wird die Attribute *vname* und *nname* haben.

Eine andere nützliche Parameter-Konvertierung benutzt Formularvariablen, um die Aktion des Formulars neu zu schreiben. Das erlaubt Ihnen, ein Formular - abhängig von den Eingaben - an verschiedene Skripte zu übertragen. Das ist besonders nützlich, falls ein Formular mehrere Übertragungsbuttons enthält. Zopes Aktions-Konverter sind:

action
: Ändert die Aktion des Formulars. Das ist besonders nützlich, falls Sie mehrere Übertragungsbuttons in einem Formular haben. Jeder Button kann einem Skript zugeordnet werden, das aufgerufen wird, wenn der Button angeklickt wird, um das Formular zu übertragen.

default_action

Ändert das Aktions-Skript des Formulars, wenn kein anderer *method*-Konverter gefunden wird.

Hier ist ein Beispiel-Formular, das Aktions-Konverter benutzt:

```
      <form action="">
        <p>
        Wählen Sie einen oder mehrere Angestellte aus:
        </p>
        <input type="checkbox" name="angestellte:list" value="Larry">
Larry<br>
        <input type="checkbox" name="angestellte:list" value="Simon">
Simon<br>
        <input type="checkbox" name="angestellte:list" value="René">
Rene<br>
        <input type="submit" name="feuereAngestellte:action"
value="Feuern!"><br>
        <input type="submit" name="befoerdereAngestellte:action"
value="Befördern!">
      </form>
```

Dieses Formular wird entweder das Skript *feuereAngestellte* oder *befoerdereAngestellte* aufrufen, abhängig davon, welcher der beiden Übertragungsbuttons benutzt wird. Beachten Sie auch, wie es mit dem Konverter *list* eine Liste der Angestellten aufbaut. Formular-Konverter können beim Design von Zope-Anwendungen sehr nützlich sein.

Skript-Sicherheit

Alle Skripte, die durch das Web bearbeitet werden können, sind Gegenstand von Zopes Standard-Sicherheitsregeln. Die einzigen Skripte, die nicht Gegenstand dieser Sicherheitsregeln sind, sind diejenigen, die durch das Dateisystem bearbeitet werden müssen. Diese uneingeschränkten Skripte schließen die *Externen Skripte* von Python und Perl ein.

Kapitel 7 (Benutzer und Sicherheit) deckt Sicherheitsaspekte detaillierter ab. Sie sollten für mehr Informationen darüber, wie Skripte von Zopes Sicherheitszwängen eingeschränkt werden, in den Abschnitten *Rollen ausführbarer Objekte* und *Proxy-Rollen* nachschlagen.

Das Zope-API

Der bequeme Zugang zum Zope-API (Application Programmer Interface; Anwendungsprogrammierer-Schnittstelle)ist einer der Hauptgründe, mit Skripten an Zope zu arbeiten. Das Zope-API beschreibt eingebaute Aktionen, die auf Zope-Objekten aufgerufen werden können. Sie können das Zope-API im Hilfe-System untersuchen, wie in Abbildung 8.4 gezeigt.

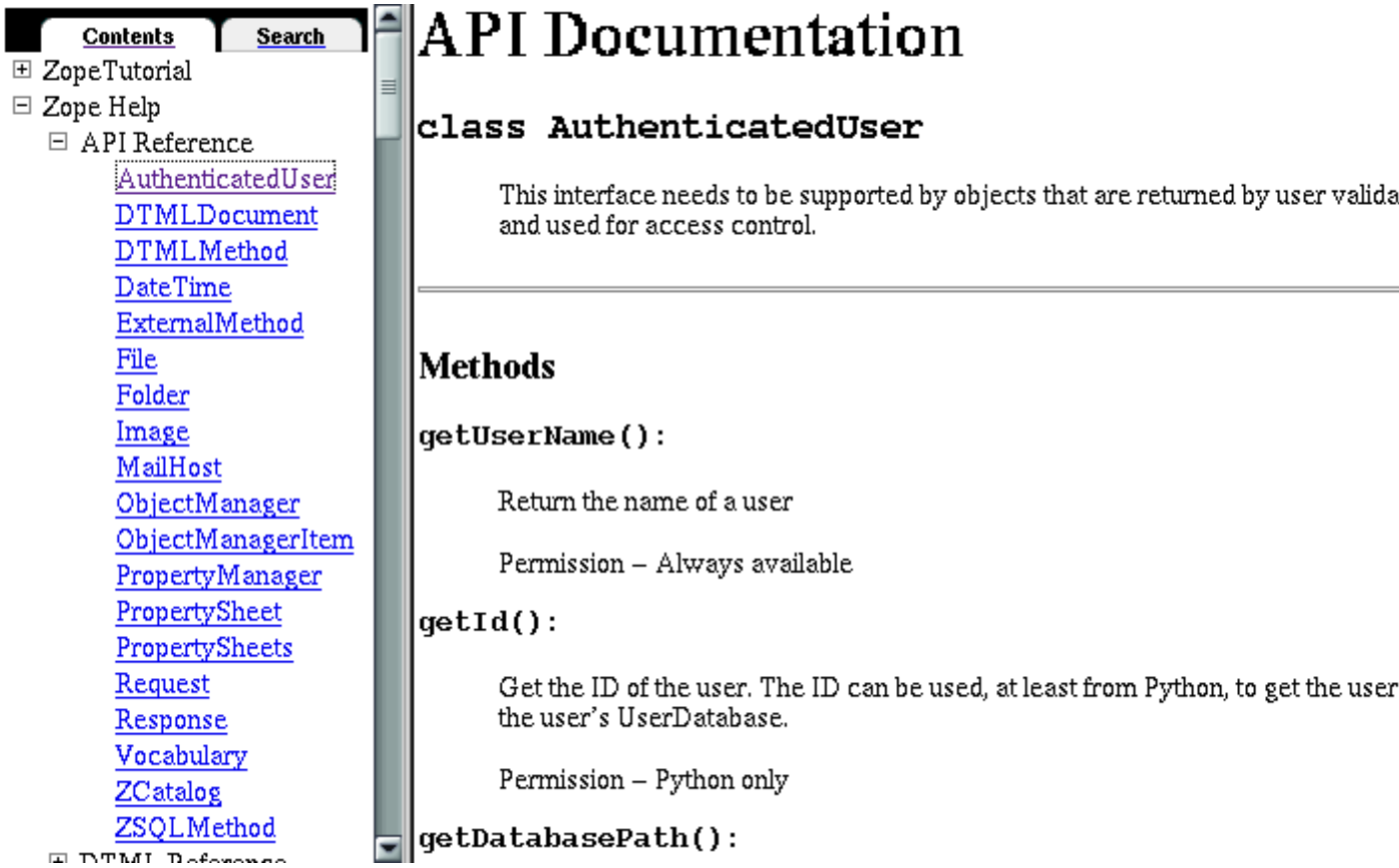


Abbildung 8.4 Dokumentation des Zope-APIs

Angenommen, Sie möchten ein Skript haben, das aus einer Datei, die Sie über ein Formular hochgeladen haben, ein Zope-Objekt in einem Ordner macht. Um das zu tun, müssen Sie eine Anzahl von Zope-API-Aktionen kennen. Es ist leicht genug, Dateien in Python oder Perl zu lesen, aber wenn Sie erst die Datei haben, müssen sie wissen, welche Aktionen aufgerufen werden müssen, um ein neues Datei-Objekt in einem Ordner anzulegen.

Es gibt viele andere Dinge, die Sie gern über ein Skript im Zope-API durchführen mögen. Jede Verwaltungsaufgabe, die über das Web ausgeführt werden kann, kann auch über ein Skript im Zope-API getan werden. Das schließt das Anlegen, Verändern und Löschen von Zope-Objekten ein. Sie können sogar Wartungsaufgaben wie einen Neustart von Zope oder das Packen der Zope-Datenbank durchführen.

Das Zope-API ist sowohl in Anhang B (API-Referenz) dokumentiert als auch in der online-Hilfe von Zope. Die API-Dokumentation zeigt Ihnen, welche Klassen von welchen anderen Klassen erworben werden. Zum Beispiel ererbt ein *Ordner* (Folder) vom *ObjectManager*. Das bedeutet, daß Ordner-Objekten alle Aktionen zur Verfügung stehen, die im Abschnitt *ObjectManager* der API-Referenz aufgelistet sind.

Python-basierte Skripte benutzen

Oben in diesem Kapitel haben Sie einige Beispiele für Skripte gesehen. Lassen Sie uns nun einen genaueren Blick auf Skripte werfen.

Die Programiersprache Python

Python ist eine hochentwickelte, objektorientierte Programmiersprache. Das meiste von Zope ist in Python geschrieben. Viele Leute mögen Python wegen seiner Klarheit, Einfachheit und seiner Fähigkeit, große Projekte zu strukturieren.

Es stehen viele Ressourcen zur Verfügung, um Python zu erlernen. Die Website der python.org bietet eine Menge Dokumentationen einschließlich eines Tutorials von Pythons "Erfinder", Guido van Rossum.

Python verfügt über eine reiche Auswahl an Modulen und Zusätzen. Sie können mehr erfahren über die Python standard library (englisch) auf der Website der python.org.

Eine andere sehr respektierte Quelle für Referenzmaterial ist die *Python Essential Reference* (englisch) von David Beazley, veröffentlich von New Riders.

Python-basierte Skripte anlegen

Um ein Python-basiertes Skript anzulegen, wählen Sie in der "Product add list" den Punkt *Script (Python)* aus. Nennen Sie das Skript *hallo* und klicken Sie auf den *Add and Edit*-Button. Sie sollten jetzt die Ansicht *Edit*ihres Skriptes sehen wie in Abbildung 8.5.

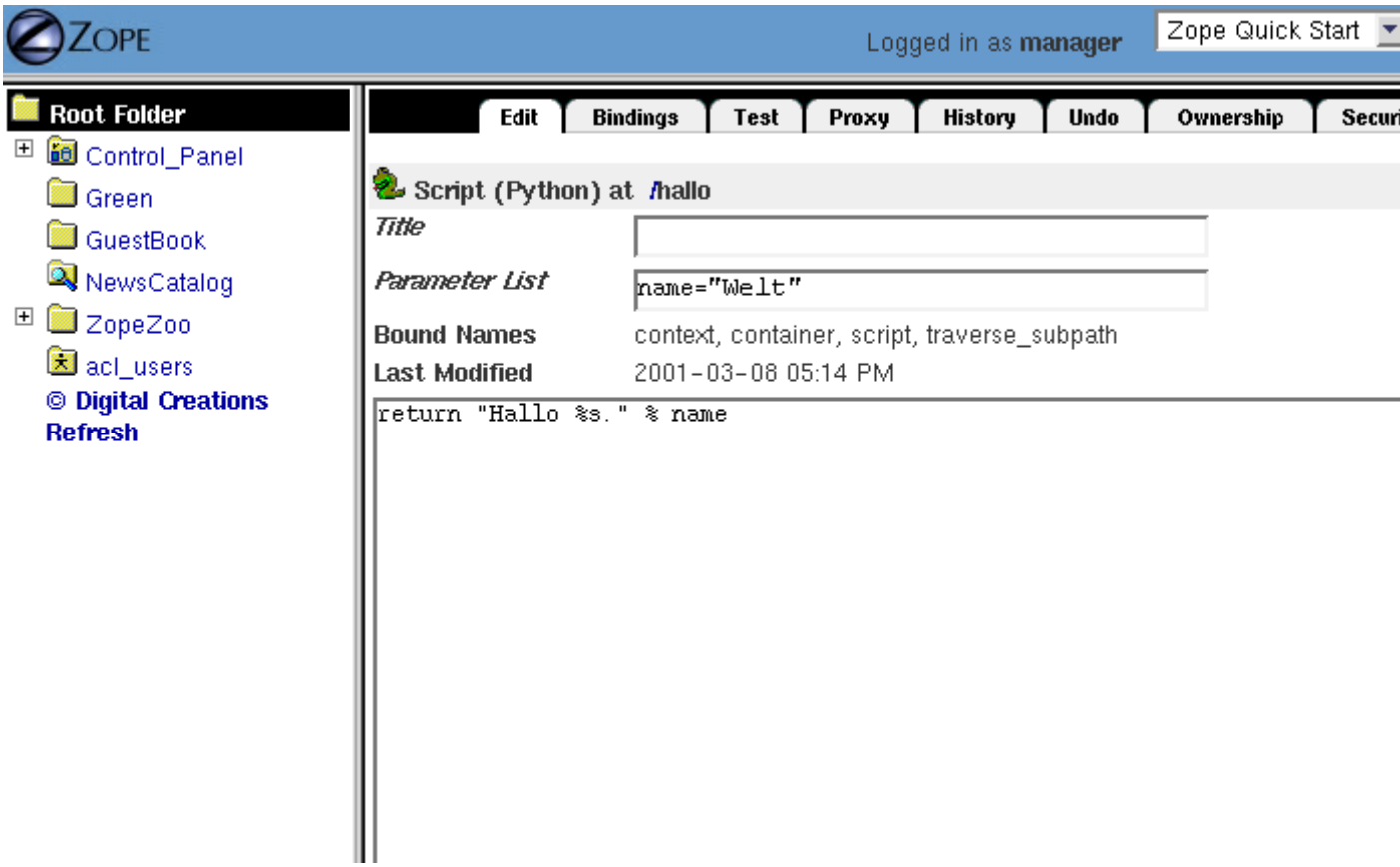


Abbildung 8.5 Skriptbearbeitungs-Ansicht

Auf diesem Bildschirm können Sie die Parameter und Inhalte ihres Skriptes kontrollieren. Sie können Parameter für ihr Skript im Feld *parameter list* eingeben. Tippen Sie den Inhalt ihres Skriptes in das Textfeld am Fuß des Bildschirms.

Geben Sie *name="Welt"* im Feld *parameter list* ein und schreiben Sie:´

```
return "Hallo %s." % name
```

in den Inhalt des Skripts. Das entspricht dem Folgenden in der Python-Syntax:

```
def hallo(name="Welt"):
    return "Hallo %s." % name
```

Sie können dieses Skript nun prüfen, indem Sie die Registerkarte *Test* aufrufen wie in Abbildung 8.6 gezeigt.

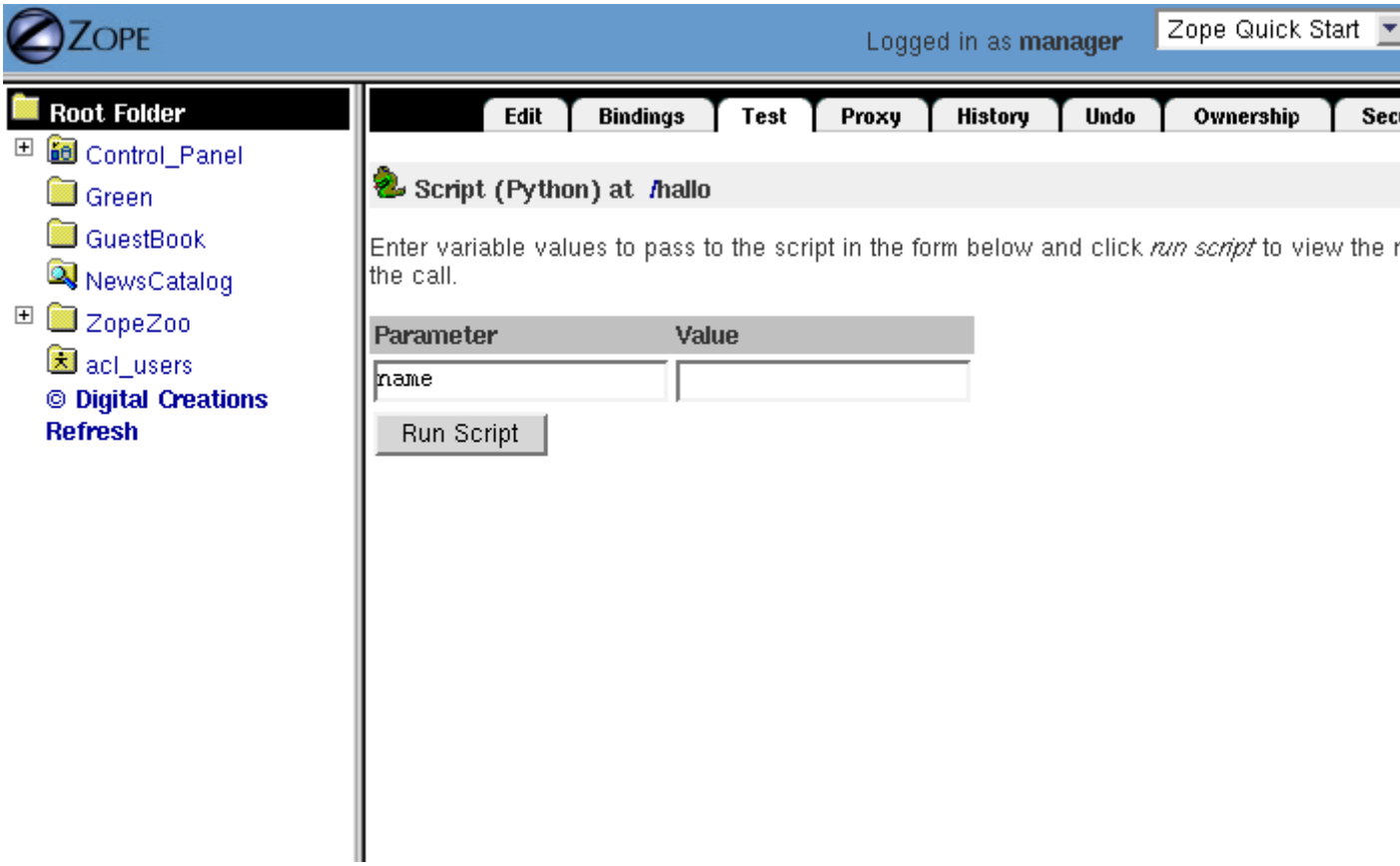


Abbildung 8.6 Ein Skript testen

Lassenn sie das Feld *name* leer und klicken Sie auf den Button *Run Script*. Zope sollte "Hallo Welt" zurückgeben. Nun gehen Sie zurück und versuchen, ihren Namen in das Feld *Value* einzugeben und dann den Button *Run Script* zu klicken. Zope sollte nun "Hallo" zu Ihnen sagen.

Wenn unter Zope Skripte aufgerufen werden, bekommen Sie über die Variable *context* Zugang zu Zope-Objekten. Zum Beispiel gibt dieses Skript die Zahl der Objekte zurück, die in einem bestimmten Zope-Objekt enthalten sind:

```
## Script (Python) "numberOfObjects
##
return len(context.objectIds())
```

Das Skript ruft `context.objectIds()` auf, um die Zahl der enthaltenen Objekte herauszufinden. Wenn Sie dieses Skript auf einem bestimmten Zope-Objekt aufrufen, wird die Kontext-Variable an den Objekt-Kontext gebunden. Wenn Sie also dieses Skript beim Besuch des URL *OrdnerA/OrdnerB/numberOfObjects* dieses Skript aufrufen, würden die Kontext-Parameter auf das Objekt *OrdnerB* verweisen.

Wenn Sie Logik in Python schreiben, werden Sie typischerweise Zope-Objekte suchzen wollen, andere Skripte aufrufen und Berichte zurückgeben. Nehemen Sie zum Beispiel an,. Sie wollen ein Arbeitsprozeß-System einrichten, in dem verschiedene Zope-Objekte mit Eigenschaften belegt sind, die ihren Status anzeigen. Sie mögen Berichte erstellen wollen, die zusammenfassen, welche Objekte sich in welchem Status befinden. Sie können Python benutzen, um Objekte zu suchen und ihre Eigenschaften abzufragen. Hier istr zumn Beispiel ein Skript, das *objectsForStatus* heißt und einen Parameter namens *status* hat:

```
## Skript (Python) "objectsForStatus"
##parameters=status
##
"""
Gibt alle Objekte zurück, die eine bestimmte Status-Eigenschaft
haben.
"""
results=[]
for object in context.objectValues():
    if object.getProperty('status') == status:
        results.append(object)
return results
```

Dieses Skript kreist durch die Sub-Objekte eines Objekts und gibt alle Sub-Objekte zurück, die die Eigenschaft *status* mit einem bestimmten Wert haben.

Sie können dieses Skript benutzen, um über DTML eMails zu senden. Zum Beispiel:

```
<dtml-sendmail>

To: <dtml-var VerantwortlichePerson>
Subject: Objekte in Warteschleife

Diese Objekte befinden sich in der Warteschleife und benötigen ihre
Aufmerksamkeit:
<dtml-in expr="objectsForStatus('Pending')">
  <dtml-var title_or_id> (<dtml-var absolute_url>)
</dtml-in>

</dtml-sendmail>
```

Dieses Beispiel zeigt, wie Sie DTML zur Präsentation oder Formatierung von Berichten verwenden können, während Python die Logik durchführt. Das ist ein sehr wichtiges Muster, das Sie in Zope immer wiederfinden werden.

Verarbeitung von Strings

Eine gebräuchliche Anwendung für Skripte ist die Verarbeitung von Strings. Python hat eine Anzahl von Standard-Modulen für die String-Verarbeitung. Sie können keine gewöhnliche Ausdrucksverarbeitung mit Python-basierten Skripten machen, aber Sie haben Zugang zum Modul *string*. Sie haben auch von DTML aus Zugang zum Modul *string*, aber es ist so viel einfacher von Python aus zu benutzen. Angenommen, Sie wollen alle Erscheinungen eines bestimmten Wortes in einem DTML-Dokument ändern. Hier ist ein Skript, *ersetzeWort*, das zwei Argumente akzeptiert, nämlich *Wort* und *Ersatz*. Es wird alle Erscheinungen eines bestimmten Wortes in einem DTML-Dokument ändern:

```
      ## Script (Python) "ersetzeWort"
      ##parameters=wort, ersatz
      ##
      """
      Ersetzt alle Erscheinungen eines Wortes im Quelltext eines DTML-
Dokuments mit einem
      Ersatz-Wort. Rufen Sie dieses Skript auf einem DTML-Dokument auf, um
es zu benutzen.
      Beachten Sie: Sie brauchen die Berechtigung "edit", um dieses Skript
wirksam auf einem
      Dokument auszuführen.
      """
      import string
      text=context.document_src()
      text=string.replace(text, wort, ersatz)
      context.manage_edit(text, context.title)
```

Sie können dieses Skript aus dem Web auf einem DTML-Dokument aufrufen, um den Quelltext des Dokuments zu bearbeiten. Der URL *Sumpf/ersetzeWort?wort=Alligator&ersatz=Krokodil* würde das Skript *ersetzeWort* auf einem Dokument namens *Sumpf* aufrufen und alle Erscheinungen des Wortes *Alligator* mit *Krokodil* ersetzen.

Das Modul *string*, auf das Sie über Skripte zugreifen können, hat nicht alle Möglichkeiten, die im Standard-String-Modul vopn Python verfügbar sind. Diese Begrenzungen wurden aus Sicherheitsgründen eingeführt. Siehe mehr Informationen zum Modul *string* im Anhang A.

Eine Sache, die Sie mit Skripten zu tun versucht sein könnten, ist, Python nach Objekten suchen zu lassen, die im Text oder den Eigenschaften ein bestimmtes Wort enthalten. Das können Sie machen, aber Zope hat ein viel besseres Instrument für diese Arbeit, den *Catalog*. Siehe Kapitel 11 (Inhalte durchsuchen und kategorisieren) für mehr Informationen über das Suchen mit Katalogen.

Mathe machen

Eine andere verbreuitete Anwendung von Skripten ist die Durchführung mathematischer Berechnungen, die für DTML unmöglich sind. Die Module *math* und *random* geben Ihnen über Python Zugang zu vielen mathematischen Funktionen. Diese Module sind Standard-Python-Dienste, wie auf der Website der Python.org beschrieben.

math
: Mathematischre Funktionen wie *sin* und *cos*.

random
: Funktionen zur Pseudo-Zufallszahlen-Generation.

Eine interessante Funktion des Moduls *random* ist die Funktion *choice*, die eine Zufallsauswahl einer Folge von Objekten zurückgibt. Hier ist ein Beispiel dafür, wie diese Funktion in einem Skript namens *randomImage* benutzt wird:

```
## Script (Python) "randomImage"
##
"""
Sobald dieses Skript auf einem Ordner aufgerufen wird, der Bilder
enthält,
gibt es ein Zufallsbild zurück.
"""
import random
return random.choice(context.objectValues('Image'))
```

Nehmen Sie an, Sie hätten einen Ordner *Bilder*, der eine Anzahl von Bildern enthielte. Sie könnten ein Zufallsbild aus dem Ordner auf folgende Weise in DTML anzeigen lassen:

```
<dtml-with Bilder>
  <dtml-var randomImage>
</dtml-with>
```

Dieses DTML ruft das Skript *randomImage* auf dem Ordner *Bilder* auf. Das Ergebnis ist ein HTML-*IMG*-Tag, daß auf ein Zufallsbild im Ordner *Bilder* verweis.

Bindungsvariablen

Ein Satz spezieller Variablen wird immer dann angelegt, wenn ein Python-basiertes Skript aufgreufen wird. Diese Variablen, definiert in der Ansicht *Bindings*, werden von ihrem Skript benutzt, um auf andere Zope-Objekte und -Skripte zuzugreifen.

Als Voreinstellung sind die Namen jener Bindungsvariablen auf zuverlässige Werte gesetzt und Sie sollten sie nicht zu ändern brauchen. Sie sind hier erläutert, damit Sie wissen, wie jede spezielle Variable funktioniert und wie Sie diese Variablen in ihremn Skripten benutzen können.

Kontext (context)
: Die *Kontext*-Bindung hat den Grundnamen *context*. Diese Variable bezieht sich auf das Objekt, auf dem das Skript aufgerufen wird.

Behälter (container)

Die *Behälter*-Bindung hat den Grundnamen *container*. Diese Variable bezieht sich auf den Ordner, in dem das Skript definiert ist.

Skript (script)
: Die *Skript*-Bindung hat den Grundnamen *script*. Diese Variable bezieht sich auf das Skript selbst.

Namensraum
: Die *Namensraum*-Bindung ist als Voreinstellung leer. Dies ist einje fortgeschrittene Variable, die Sie für keins der Beispiele aus diesem Buch brauchen werden. Wenn iohr Skript von einem DTML-Skript aufgerufen wird und Sie einen Namen für diese Bindung festgelegt haben, dann enthält die benannte Variable den DTML-Namensraum wie in Kapitel 8 (Variablen und fortgeschrittenes DTML) beschrieben. Wenn diese Bindung gesetzt ist, wird das Skript auch nach den Parametern im DTML-Namensraum suchen, wenn es von DTML aufgerufen wird, ohne explizit irgendwelche Argumente zu übergeben.

Sub-Pfad
: Die *Sub-Pfad*-Bindung hat den Grundnamen *traverse_subpath*. Dies ist eine fortgeschrittene Variable´, die Sie nicht für Beispiele in diesem Buch brauchen werden. Wenn ihr Skript traversiert wird - was bedeutet, daß andere Pfad-Elemente ihm in einem URL folgen - dann werden diese Pfad-Elemente in einer Liste von rechts nach links in dieser Variable angeordnet.

Wenn sie ihre Skripte per FTP bearbeiten, werden Sie feststellen, daß diese Bindungen in Kommentaren am oberen Ende ihrer Skript-Dateien aufgelistet sind. Zum Beispiel:

```
## Script (Python) "beispiel"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind subpath=traverse_subpath
##parameters=name, alter
##title=
##
return "Hallo %s, Sie sind %d Jahre alt." % (name, alter)
```

Sie können die Bindungen ihres Skripts ändern, indem Sie diese Kommentare ändern und dann ihr Skript hochladen.

Print-Statement-Unterstützung

Python-basierte Skripte haben einen besonderen Dienst, um Ihnen beim Drucken von Informationen zu helfen. Normalerweise werden Druck-Daten an die Standardausgabe gesendet und auf der Konsole dargestellt. Das ist für eine Server-Anwendung wie Zope unpraktisch, weil Sie die meiste Zeit hindurch keinen Zugang zur Server-Konsole haben. Skripte erlauben Ihnen sowieso das Drucken und auch, mit der speziellen Variable *printed* zurückzubekommen, was Sie gedruckt haben. Zum Beispiel:

```
## Script (Python) "printExample"
##
for word in ('Zope', 'on', 'a', 'rope'):
```

```
    print word
return printed
```

Diese Skript wird zurückgeben:

```
Zope
on
a
rope
```

Der Grund für den Umbruch zwischen den Worten ist, daß Python jeden gedruckten String in einer neuen0 Zeile einfügt.

Sie mögen das Print-Statement benutzen wollen, um einfaches Debugging in ihren Skripts durchzuführen. Für komplexere Kontrolle der Ausgabe sollten sie die Dinge wahrscheinlich eher selbst verwalten, indem Sie Daten sammeln, modifizieren und sie manuell zurückgeben, anstatt sich auf das Print-.Statement zu verlassen.

Sicherheitseinschränkungen

Skripte sind in ihren Rechten eingeschränkt, um ihre Fähigkeit zum Anrichten von Schaden zu begrenzen. Was könnte schädlich sein? Generell werden Sie durch Skripte davon abgehalten, auf private Zope-Objekte zuzugreifen, schädliche Änderungen an Zope-Objekten vorzunehmen, den Zope-Prozeß selbst zu stören und auf den Server zuzugreifen, auf dem Zope läuft. Diese Einschränkungen sind durch eine Ansammlung von Grenzen dessen implementiert, was ihre Skripte tun können.

Loop-Grenzen
: Skripte können endlose Loops anlegen. Wenn ihr Skript eine sehr große Menge an Loops durchführt, wird Zope einen Fehler melden. Diese Einschränkung deckt alle Arten von Loops ab, eingeschlossen *for*- und *while*-Loops. Der Grund für diese Einschränkung ist die Begrenzung ihrer Möglichkeiten, Zope durch endlose Loops zu blockieren.

Import-Grenzen
: Skripte können keine willkürlichen Pakete oder Module importieren. Daher sind Sie darauf beschränkt, das Einheitsmodul *Products.PythonScripts.standard* das Modul *AccessControl*, die per DTML verfügbaren Module (*string*, *random*, *math*, *sequence*) und Module zu importieren, die von den Produkt-Autoren spezifisch für Skripte verfügbar gemacht worden sind. Siehe Anhang B (API-Referenz) für mehr Informationen über diese Module. Wenn Sie jedes Python-Modul importieren können wollen, benutzen Sie ein Externes Skript (external method), wie unten im Kapitel beschrieben.

Zugriffsgrenzen
: Sie werden beim Zugriff auf Objekte durch die Standard-Sicherheitsregeln von Zope beschränkt. In anderen Worten: Der Benutzer, der Skripte ausführt, wird auf Authorisierung überprüft, sobald er auf Objekte zugreift. Wie bei allen ausführbaren Objekten können Sie die wirksamen Rollen verändern, die ein Benutzer hat, wenn er ein Skript aufruft, indem sie die *Proxy-Rollen* (Proxy Roles) benutzen - siehe Kapitel 7 (Benutzer und Sicherheit) für mehr Informationen. Außerdem können Sie nicht auf

Objekte zugreifen, deren Name mit einem Unterstrich beginnt, weil Zope diese Objekte als privat betrachtet.

Schreib-Grenzen
: Generell können Sie die Attribute von Zope-Objekten mit Skripten ändern. Sie sollten eher Skripte auf Zope-Objekten aufrufen, um sie zu ändern, anstatt direkt die Instanz-Attribute zu ändern.

Trotz dieser Begrenzungen könnte ein bestimmter Benutzer große Mengen an Prozessorzeit und Speicher mit der Benutzung von Python-basierten Skripten belegen. Solche bösartigen Skripte könnten durch die Benutzung großer Ressourcenmengen eine Art "denial of service" (DOS)-Angriff darstellen. Das sind schwer zu lösende Probleme und DTML leidet unter demselben Mißbrauchspotential. Was DTML angeht, sollten Sie vielleicht nur vertrauenwürdigen Leuten Zugang zu Skripten gewähren.

Eingebaute Funktionen

Python-basierte Skripte geben Ihnen eine leicht unterschiedliche Zusammenstellung von eingebauten Funktionen als Sie sie normalerweise in Python finden.Die meisten Änderungen wurden angelegt, um Sie von der Durchführung unsicherer Aktionen abzuhalten. Zum Beispiel ist die Funktion *open* nicht verfügbar, was Sie vor Zugang zum Dateisystem bewahrt. Um es teilweise für einige fehlende Einbauten zu öffnen, sind ein paar Extra-Funktionen verfügbar.

Diese eingeschränkten Einbauten arebiten genauso wie Python-Standard-Einbauten: *None*, *abs*, *apply*, *callable*, *chr*, *cmp*, *complex*, *delattr*, *divmod*, *filter*, *float*, *getattr*, *hash*, *hex*, *int*, *isinstance*, *issubclass*, *list*, *len*, *long*, *map*, *max*, *min*, *oct*, *ord*, *repr*, *round*, *setattr*, *str*, *tuple*. Für mehr Informationen darüber, was diese Einbauten tun, siehe die online-Python Documentation.

Die Funktionen *range* und *pow* sind verfügbar und arbeiten auf dieselbe Weise, wie sie es in Standard-Python tun; dennoch sind sie beschränkt, um zu verhindern, daß sehr große Zahlen und Sequenzen generiert werden. Diese Beschränkung hilft dabei, sich gegen DOS-Angriffe zu schützen, wie oben beschrieben.

Zusätzlich sind diese Funktionen verfügbar: *DateTime* und *test*. Siehe Anhang A (DTML-Referenz) für mehr Informationen über diese Funktionen.

Schließlich gibt es die Funktion *same_type* - die den Typ von zwei oder mehr Objekten vergleicht und "wahr" zurückgibt, wenn sie vom selben Typ sind - um das Fehlen der Funktion *type* auszugleichen. Also sagen Sie nicht:

```
if type(foo) == type([]):
      return "foo ist eine Liste"
```

um zu prüfen, ob `foo` eine Liste ist, sondern stattdessen benutzen Sie die Funktion *same_type*, um es zu prüfen:

```
if same_type(foo, []):
```

```
        return "foo ist eine Liste"
```

Lassen Sie uns nun einen Blick auf *Externe Skripte* (External Methods) werfen, die Macht und weniger Einschränkungen bieten als Python-basierte Skripte.

Externe Skripte benutzen

Manchmal geraten Ihnen die durch Skripte auferlegten Sicherheitseinschränkungen in den Weg. Sie mögen zum Beispiel Dateien von der Festplatte lesen oder auf das Netzwerk zugreifen oder einige fortgeschritene Bibliotheken für Sachen wie Regel-Ausdrücke oder Grafikbearbeitung nutzen. In diesen Fällen werden Sie *Externe Skripte* (External Methods) verwenden wollen.

Um Externe Skripte anzulegen, brauchen Sie Zugang zum Dateisystem. Das macht die Bearbeitung dieser Skripte etwas beschwerlicher, da Sie sie nicht direkt in ihrem Browser bearbeiten können. Dennoch ist das Verlangen von Zugriff auf das Dateisystem des Servers eine wichtige Sicherheitskontrolle. Wenn ein Benutzer Zugriff auf das Dateisystem des Servers hat, gibt ihm das auch schon die Fähigkeit, Zope zu beschädigen. Also stellt Zope sicher, daß nur bereits als vertrauenswürdig bekannte Leute Zugang haben, wenn verlangt wird, daß uneingeschränkte Skripte im Dateisystem bearbeitet werden.

Uneingeschränkte Skripte werden als Dateien auf dem Zope-Server im Verzeichnis *Extensions* angelegt und bearbeitet. Alternativ können Sie uneingeschränkte Skripte auch in einem Verzeichnis *Extensions* innerhalb eines installierten Zope-Product-Verzeichnisses anlegen und bearbeiten.

Legen sie eine Datei namens *Beispiel.py* im Zope-Verzeichnis *Extensions* auf ihrem Server an. In der Datei *Beispiel.py* geben Sie den folgenden Code ein:

```
def hallo(name="Welt"):
    return "Hallo %s." % name
```

Sie haben eine Python-Funktion in einem Python-Modul angelegt. Nun lassen Sie uns diese Funktion in einem Externen Skript benutzen.

Sie verwalten Externe Skripte auf dieselbe Weise wie Sie eingeschränkte Skripte verwalten, mit der Ausnahme daß Sie das Skript nicht selbst über das Web bearbeiten können. Statt Code zu bearbeiten, müssen Sie Zope sagen, wo es ihren Code im Dateisystem findet. Das müssen Sie tun, indem Sie den Namen ihrer Python-Datei und den Namen der Funktion innerhalb des Moduls spezifizieren.

Um ein Externes Skript anzulegen, wählen Sie *External Method* aus der Liste "Add" im ZMI. Sie werden zu einem Formular geführt, wo Sie eine `id` angeben müssen. Geben Sie "hallo" in das Feld *Id* und in das Feld *Function name* ein und "Beispiel" in das Feld *Module name*, und klicken Sie den Button *Add*. Sie sollten nun ein Externes Skript-Objekt in ihrem Ordner sehen. Klicken Sie darauf. Sie sollten zur Ansicht *Eigenschaften* (Properties) ihres neuen Externen Skripts geführt werden wie in Abbildung 8.7 gezeigt.

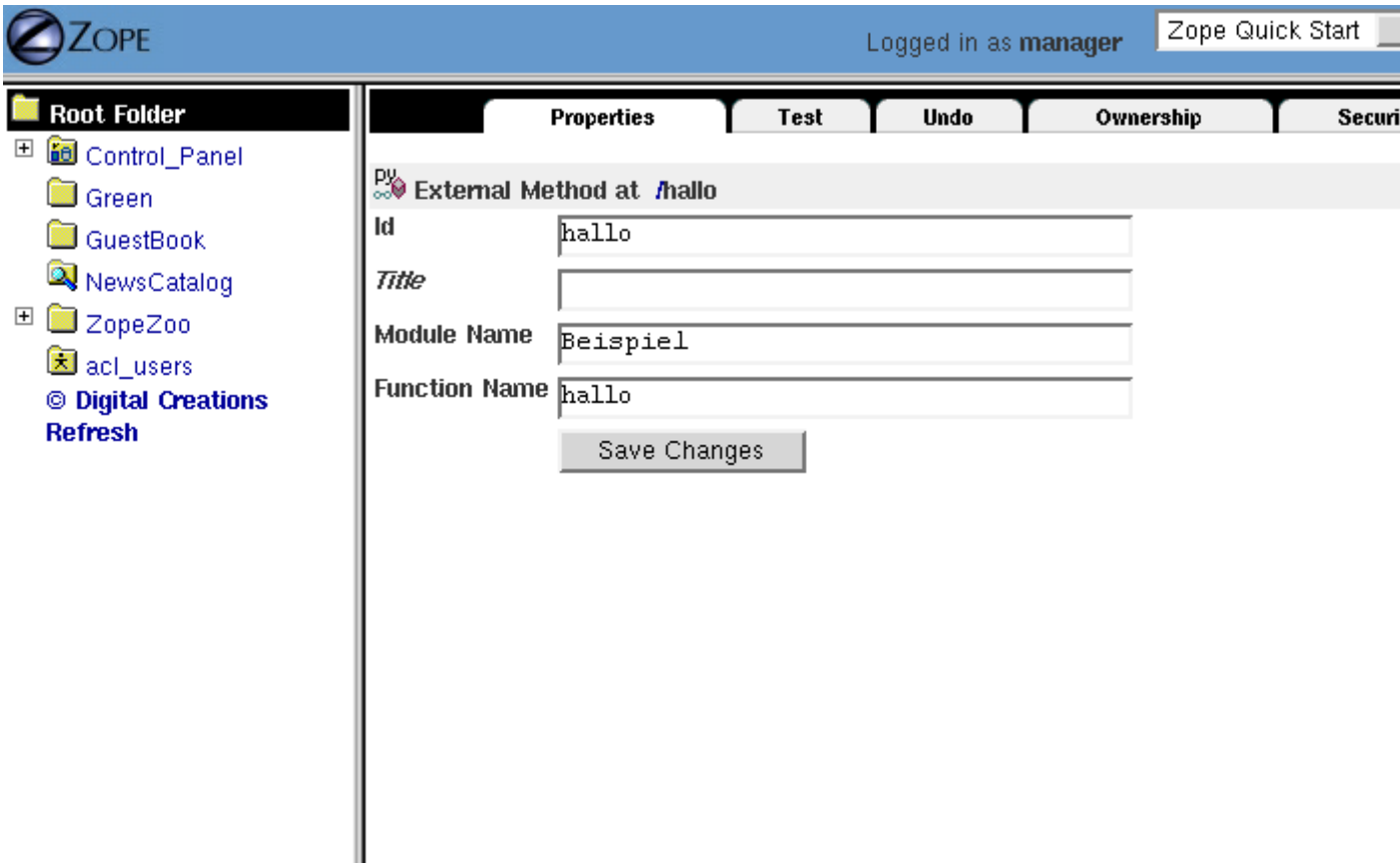


Figure 8.7 Eigenschaften-Ansicht Externer Skripte

Testen Sie nun ihr neues Skript, indem Sie die Ansicht *Test* aufrufen. Sie sollten eine Begrüßung sehen. Sie können dem Skript verschiedene Namen übergeben, indem Sie sie im URL spezifizieren. Zum Beispiel *hallo?name=Spanische+Inquisition*.

Dieses Beispiel ist genau dasselbe wie das Hallo-Welt-Beispiel, das Sie für den Gebrauch von Skripten gesehen haben. Tatsächlich bieten eingeschränkte Skripte eine bessere Lösung für einfache String-Verarbeitungsaufgaben wie diese, weil es einfacher ist, mit ihnen zu arbeiten.

Die Hauptgründe für die Benutzung uneingeschränkter Skripte liegen darin, auf das Dateisystem oder Netzwerk zuzugreifen oder Python-Pakete zu benutzen, die für eingeschränkte Skripte nicht verfügbar sind.

Hier ist ein Beispiel für ein Externes Skript, das die Python Imaging Library (PIL) benutzt, um eine Thumbnail-Version von einem exitierenden Grafik-Objekt in einem Ordner anzulegen. Geben Sie den folgenden Code in eine Datei namens *Thumbnail.py* im Ordner *Extensions* ein:

```
      def makeThumbnail(self, original_id, size=200):
          """
          Stellt über die Id einer vorhandenen Grafik eine Thumbnail-Grafik
her, wenn es auf einem
          Zope-Ordner aufgerufen wird.
```

```
          Der Thumbnail ist ein Zope-Image-Objekt, das eine verkleinert
JPG-Anzeige der Original-
          Grafik ist. Er hat eine Eigenschaft 'original_id', die auf die Id
der Grafik in
          Originalgröße gesetzt ist.
          """

          from PIL import Image
          from StringIO import StringIO
          import os.path

          # lege eine Thumbnail-Datei an
          original_image=getattr(self, original_id)
          original_file=StringIO(str(original_image.data))
          image=Image.open(original_file)
          image=image.convert('RGB')
          image.thumbnail((size,size))
          thumbnail_file=StringIO()
          image.save(thumbnail_file, "JPEG")
          thumbnail_file.seek(0)

          # lege eine Id für den Thumbnail an
          path, ext=os.path.splitext(original_id)
          thumbnail_id=path + '.thumb.jpg'

          # wenn es einen alten Thumbnail gibt, lösche ihn
          if thumbnail_id in self.objectIds():
              self.manage_delObjects([thumbnail_id])

          # lege das Zope-Image-Objekt an
          self.manage_addProduct['OFSP'].manage_addImage(thumbnail_id,
                                                         thumbnail_file,
                                                         'thumbnail image')
          thumbnail_image=getattr(self, thumbnail_id)

          # setze die Eigenschaft 'originial_id'
          thumbnail_image.manage_addProperty('original_id', original_id,
'string')
```

Sie müssen die PIL installiert haben, damit das Beispiel funktioniert. Siehe die PythonWorks-Website für mehr Informationen über die PIL. Um diesen Code zu benutzen, legen Sie ein Externes Skript mit dem Namen *makeThumbnail* an, das die Funktion *makeThumbnail* im Modul *Thumbnail* benutzt.

Nun haben sie ein Skript, das eine Thumbnail-Grafik anlegen wird. Sie können es auf einem Ordner mit einem URL wie *ImageFolder/makeThumbnail?original_id=Pferd.gif* aufrufen. Das würde eine Thumbnail-Grafik anlegen, die `Pferd.thumb.jpg` heißt.

Sie können ein Skript benutzen, um über alle Grafiken in einem Ordner zu loopen und Thumbnail-Grafiken von ihnen anzulegen. Legen Sie ein Skript namens *makeThumbnails* an:

```
## Skript (Python) "makeThumbnails"
##
for image_id in context.objectIds('Image'):
    context.makeThumbnail(image_id)
```

Es wird über alle Grafiken in einem Ordner loopen und für jede einen Thumbnail anlegen.

Nun rufen sie dieses Skript auf einem Ordner mit Grafiken darin auf. Es wird eine Thumbnail-Grafik für jede enthaltene Grafik anlegen. Versuchen Sie, das Skript *makeThumbnails* noch einmal auf diesem Ordner aufzurufen und Sie werden feststellen, daß es Thumbnails von ihren Thumbnails hergetsellt hat. Das ist nicht gut. Sie müssen das Skript *makeThumbnails* ändern, damit es vorhandene Thumbnail-Grafiken erkennt und keine Thumbnails von ihnen mehr anlegt. Weil alle Thumbnails die Eigenschaft *original_id* haben, können Sie als eine Unterscheidungsmethode zwischen Thumbnails und normalen Grafiken gegen diese Eigenschaft prüfen:

```
## Skript (Python) "makeThumbnails"
##
for image in context.objectValues('Image'):
    if not image.hasProperty('original_id'):
        context.makeThumbnail(image.getId())
```

Löschen Sie alle Thumbnail-Grafiken in ihrem Ordner und versuchen sie, ihr nachbearbeitetes Skript *makeThumbnails* auf dem Ordner aufzurufen. Jetzt scheint es richtig zu funktionieren.

Nun können Sie ihr Skript und das Externe Skript mir ein wenig DTML zusammenkleben. Legen Sie ein DTML-Skript an, das *displayThumbnails* heißt:

```
<dtml-var standard_html_header>

<dtml-if updateThumbnails>
  <dtml-call makeThumbnails>
</dtml-if>

<h2>Thumbnails</h2>

<table>
  <tr valign="top">

    <dtml-in expr="objectValues('Image')">
      <dtml-if original_id>
        <td><a href="&dtml-original_id;"><dtml-var sequence-
item></a><br>
        <dtml-var original_id></td>
      </dtml-if>
    </dtml-in>

  </tr>
</table>

<form>
  <input type="submit" name="updateThumbnails" value="Thumbnails
aktualisieren">
</form>

<dtml-var standard_html_footer>
```

Wenn Sie dieses DTML-Skript auf einem Ordner aufrufen, wird es durch alle Grafiken in dem Ordner loopen, alle Thumbnail-Grafiken anzeigen und Sie mit dem Orginal verlinken wie in Abbildung 8.8 gezeigt.

Abbildung 8.8 Thumbnail-Grafiken anzeigen

Dieses DTML-Skript schließt auch ein Formular ein, das Ihnen erlaubt, die Thumbnail-Grafiken zu aktualisieren. Wenn sie in ihrem Ordner Grafiken hinzufügen, löschen oder ändern, können Sie dieses Formular nutzen, um ihre Thumbnails zu aktualisieren.

Dieses Beispiel zeigt, wie Skripte, Externe und DTML-Skripte, gemeinsam benutzt werden. Python sorgt für die Logik, während DTML die Darstellung übernimmt. Ihre Externen Skripte gehen mit externen Paketen um, während ihre Skripte die einfache Verarbeitung von Zope-Objekten durchführen.

XML-Verarbeitung mit Externen Skripten

Sie können Externe Skripte benutzen, um damit dammich fast alles zu tun. Eine interessante Sache, die Sie machen können, ist die Kommunikation über XML. Sie können mit Externen Skripten XML generieren und verarbeiten.

Zope versteht schon ein paar XML-Nachrichten wie etwa XML-RPC und WebDAV. Während Sie Web-Anwendungen schreiben, die mit anderen Systemen kommunizieren, mögen sie die Fähigkeit haben wollen, XML-Nachrichten zu erhalten. Sie können XML auf eine Anzahl von Wegen erhalten: Sie können XML-Dateien aus dem Dateisystem oder über

das Netzwerk lesen, oder Sie können Skripte definieren, die XML-Argumente übernehmen, die von entfernten Systemen aufrufbar sind.

Wenn Sie eine XML-Nachricht erhalten haben, müssen sie das XML verarbeiten, um herauszufinden, was es bedeutet und wie darauf zu reagieren ist. Lassen Sie uns einen schnellen Blick darauf werfen, wie Sie XML mit Python manuell parsen können. Angenommen, Sie wollen ihre Web-Anwendung an einen Jabber-Chatserver anbinden. Sie mögen Benutzern erlauben wollen, Sie zu benachrichtigen und dynamische Antworten zu erhalten, die auf dem Status ihrer Web-Anwendung gründen. Nehmen Sie zum Beispiel an, Sie wollen Benutzern erlauben, mit Instant Messaging den Status von Tieren zu überprüfen. Ihre Anwendung sollte etwa so auf XML-Instant Messaging antworten:

```
<message to="kaefig_monitor@zopezoo.org" from="user@host.com">
  <body>
  Fütterungsstatus Affen
  </body>
</message>
```

Sie könnten den Inhalt der Nachricht auf Kommandos scannen, ein Skript aufrufen und Antworten wie diese zurückgeben:

```
<message to="user@host.com" from="cage_monitor@zopezoo.org">
  <body>
  Die Affen wurden zuletzt um 3.15 Uhr gefüttert.
  </body>
</message>
```

Hier ist eine Skizze davon, wie Sie diese XML-Messaging-Einheit mit einem Externen Skript in ihre Web-Anwendung implementieren könnten:

```
# Benutzt Python 2.x Standard-XML-Verarbeitungspakete. Siehe
# http://www.python.org/doc/current/lib/module-xml.sax.html fuer
# Informationen ueber Pythons SAX(Simple API for XML)-Unterstuetzung.
# Wenn Sie Python 1.5.2 benutzen, koennen sie das PyXML-Paket
erhalten.
# Siehe http://pyxml.sourceforge.net für mehr Informationen über
PyXML.

from xml.sax import parseString
from xml.sax.handler import ContentHandler

class MessageHandler(ContentHandler):
    """
    SAX message handler class

    Loest eine Nachricht nach oder von ihrem Nachrichtenkoerper auf
    """

    inbody=0
    body=""

    def startElement(self, name, attrs):
        if name=="message":
```

```
                self.recipient=attrs['to']
                self.sender=attrs['from']
            elif name=="body":
                self.inbody=1

        def endElement(self, name):
            if name=="body":
                self.inbody=0

        def characters(self, content):
            if self.inbody:
                self.body=self.body + content

    def receiveMessage(self, message):
        """
        Aufgerufen von einem Jabber-Server
        """
        handler=MessageHandler()
        parseString(message, handler)

        # rufe ein Skript auf, das einen Antwort-String auf einen
        # bestimmten Nachrichtenkoerper-String zurückgibt
        response_body=self.getResponse(handler.body)

        # lege eine XML-Antwortnachricht an
        response_message="""
          <message to="%s" from="%s">
            <body>
            %s
            </body>
          </message>""" % (handler.sender, handler.recipient,
response_body)

        # gib sie zurueck an den Server
        return response_message
```

Das Externe Skript *receiveMessage* benutzt Pythons SAX(Simple API for XML)-Paket, um die XML-Nachricht zu parsen. Die Klasse *MessageHandler* erhält Rückrufe, während Python die Nachricht parst. Der Handler speichert die Informationen, die ihn interessieren. Das Externe Skript benutzt die Handler-Klasse, indem es eine Instanz von ihr anlegt und übergibt sie an die Funktion *parseString*. Dann findet es eine Antwort-Nachricht heraus, indem es mit dem Nachrichtenkörper *getResponse* aufruft. Das Skript *getResponse* (hier nicht gezeigt) scannt den Nachrichtenkörper wahrscheinlich auf Kommandos, durchsucht den Status der Web-Anwendung und gibt eine Antwort zurück. Das Skript *receiveMessage* legt dann eine XML-Nachricht an, indem es die Antwort und die Absenderinformationen benutzt und gibt sie zurück.

Der entfernte Server würde dieses Externe Skript benutzen, indem das Skript *receiveMessage* mit dem Standard-HTTP POST-Kommando aufgerufen wird. Voîla, Sie haben einen brauchbaren XML-Chatserver aufgesetzt, der über HTTP läuft.

Fallstricke Externer Skripte

Während Sie grundsätzlich in dem, was sie in einem Externen Skript tun können, unbeschränkt sind, gibt es immer noch einige Dinge, die schwer zu tun sind.

Während ihr Python-Code tun kann, was er möchte, müssen sie bei der Arbeit mit im Rahmen von Zope dessen Regeln respektieren. Die Programmierung des Zope-Frameworks ist ein zu anspruchsvoller Punkt , um ihn hier zu diskutieren, und es gibt ein paar Dinge, denen sie sich bewußt sein sollten.

Probleme können auftreten, wenn Sie Instanzen ihrer eigenen Klassen an Zope übergeben und erwarten, daß sie funktionieren wie Zope-Objekte. Sie können beispielsweise in einem Externen Skript keine Klasse definieren und sie dann als Attribut einem Zope-Objekt zuweisen. Das macht Probleme mit Zopes Dauermaschinerie. Sie können auch nicht einfach Instanzen ihrer eigenen Klassen an DTML oder Skripte übergeben. Der Knackpunkt ist hier, daß ihre Instanzen keine Zope-Sicherheitsinformationen haben. Sie können nach Herzenslust ihre eigenen Klassen und Instanzen definieren und benutzen, aber erwarten Sie nicht, daß Zope sie direkt benutzt. Beschränken sie sich darauf, einfache Python-Strukturen wie Strings, Wörterbücher und Listen oder Zope-Objekte zurückzugeben.

Perl-basierte Skripte verwenden

Perl-basierte Skripte erlauben ihnen, Zope in Perl zu skripten. Wenn Sie Perl mögen, und Python nicht erlernen wollen, um Zope zu benutzen, sind diese Skripte für Sie. Um Perl-basierte Skripte zu verwenden, können Sie all ihre Lieblings-Perl-Module benutzen und Zope behandeln wie eine Ansammlung von Perl-Objekten.

Die Programmiersprache Perl

Perl ist eine hochentwickelte Skriptsprache wie Python. Grob betrachtet sind Perl und Python sehr ähnliche Sprachen, sie haben ähnlich primitive Datenkonstrukte und verwenden ähnliche Programmierkonstrukte.

Perl ist eine populärew Sprache für Internet-Skripting. In den frühen Tagen des CGI-Skriptings waren Perl und Python praktisch Synonyme. Perl ist immer noch die herrschende Internet-Skripting-Sprache.

Perl hat eine sehr reiche Sammlung von Modulen, um fast jede Computeraufgabe zu erledigen. CPAN (Comprehensive Perl Archive Network) ist der beste Führer zu Perl-Ressourcen.

Perl-basierte Zope-Skripte sind zum Herunterladen bei ActiveState verfügbar. Perl-basierte Skripte verlangen, daß Sie Perl und ein paar andere Pakete installiert haben, und wie diese Dinge installiert werden, ist jenseits des Horizonts dieses Buches. Schlagen Sie in der Dokumentation nach, die Sie bei den Perl-basierten Skripten vom obigen URL finden. Es gibt auch mehr Informationen, die Andy McKay auf der Zope.org bereitgestellt hat.

Perl-basierte Skripte anlegen

Perl-basierte Skripte sind Python-basierten Skripten ziemlich ähnlich. Beide haben Zugriff auf Zope-Objekte und werden auf ähnliche Weise aufgerufen. Hier ist das Hallo-Welt-Programm in Perl:

```
my $name=shift;
  return "Hallo $name.";
```

Lassen Sie uns einen Blick auf ein komplexeres Beispiels-Skript von Monty Tailor werfen. Es benutzt das `LWP::UserAgent`-Paket, um den URL des täglich erscheinenden Dilbert-Comic aus dem Netz zu erhalten. Legen sie ein Perl-basiertes Skript namens *get_dilbert_url* mit diesem Code an:

```
use LWP::UserAgent;

my $ua = LWP::UserAgent->new;

# hole die Dilbert-Seite
my $request = HTTP::Request->new('GET','http://www.dilbert.com');
my $response = $ua->request($request);

# sieh nach dem Grafik-URL im HTML
my $content = $response->content;
$content =~ m,(/comics/dilbert/archive/images/[^"]*),s;

# gib den URL zurück
return $content
```

Sie können sich den täglichen Dilbert-Comic anzeigen lassen, wenn Sie dieses Skript über DTML aufrufen, wo das Skript innerhalb eines HTML-*IMG*Tags aufgerufen wird:

```
<img src="&dtml-get_dilbert_url;">
```

Dennoch gibt es ein Problem mit diesem Code. Jedes Mal, wenn Sie den Cartoon anzeigen lassen, muß Zope eine Netzverbindung aufbauen. Das ist ineffizient und verschwenderisch. Es wäre viel besser, den aktuellen URL des Dilbert-Comics nur einmal täglich herauszufinden.

Hier ist ein Skript namens *cached_dilbert_url*, das die Situation verbessert, indem mit Hilfe einer Eigenschaft *dilbert_url_date* beobachtet wird, wann zum letzten Mal der Dilbert-URL geholt worden ist:

```
my $context=shift;
my $date=$context->getProperty('dilbert_url_date');

if ($date==null or $now-$date > 1){
    my $url=$context->get_dilbert_url();
    $context->manage_changeProperties(
      dilbert_url => $url
      dilbert_url_time => $now
    );
}
return $context->getProperty('dilbert_url');
```

Dieses Skript benutzt zwei Eigenschaften: *dilbert_url* und *dilbert_url_date*. Wenn der URL zu alt wird, wird ein neuer geholt. Sie können dieses Skript von DTML aus genauso benutzen wie das Original-Skript:

```
<img src="&dtml-cached_dilbert_url;">
```

Sie können Perl und DTML zusammen benutzen, um ihre Logik und ihre Darstellung zu kontrollieren.

Perl-basierte Skript-Sicherheit

Wie DTML und Python-basierte Skripte verhindern auch Perl-basierte Skripte im Zope-Sicherheitssystem, daß sie irgendetwas tun, was Sie nicht dürfen. Skript-Sicherheit ist in beiden Programmiersprachen ähnlich, aber es gibt ein paar Perl-spezifische Zwänge.

Erstens erlaubt Ihnen das Sicherheitssystem nicht, einen Ausdruck in Perl mit dem Befehl *eval* zu behandeln. Beachten sie zum Beispiel dieses Skript:

```
my $context = shift;
my $input = shift;

eval $input
```

Dieser Code nimmt ein Argument und evaluiert es in Perl. Das bedeutet, Sie könnten dieses Skript von - sagen wir - einem HTML-Formular aufrufen und die Inhalte eines der Formularelemente evaluieren. Das ist nicht erlaubt, weil das Formularelement bösartigen Code enthalten könnte.

Perl-basierte Skripte können auch keine neuen Variablen an irgendein Objekt zuweisen, außer lokale Variablen, die Sie mit *my* erklären.

DTML gegen Python gegen Perl

Zope gibt Ihnen viele Weg, um Skripte anzulegen. Für kleine Skript-Aufgaben macht die Wahl zwischen Python, Perl oder DTML wahrscheinlich keinen großen Unterschied. Für größere, Logik-orientierte Aufgaben sollten sie Python oder Perl benutzen. Sie sollten die Sprache wählen, die Ihnen am bequemsten ist. Natürlich könnte ihr Chef in dieser Angelegenheit mitreden wollen.

Nur zum Vergleich ist hier ein einfaches Skript, vorgeschlagen von Gisle Aas, dem Autor Perl-basierter Skripte, in drei verschiedenen Programmiersprachen.

In DTML:

```
<dtml-in objectValues>
  <dtml-var getId>: <dtml-var sequence-item>
</dtml-in>
done
```

In Python:

```
for item in context.objectValues():
    print "%s: %s" % (item.getId(), item)
print "done"
return printed
```

In Perl:

```
my $context = shift;
my @res;

for ($context->objectValues()) {
    push(@res, join(": ", $_->getId(), $_));
}
join("\n", @res, "done");
```

Trotz der Tatsache, daß Zope in Python implementiert ist, folgt es der Perl-Philosophie, daß es mehr als einen Weg gibt, es zu tun.

Fern-Skripting und Netzwerkdienste

Webserver werden benutzt, um Inhalte an Software-Clients auszuliefern; gewöhnlich Leute, die Webbrowser-Software benutzen. Der Software-Client kann also ein anderer Computer sein der ihren Webserver benutzt, um auf irgendeine Art von Dienst zuzugreifen.

Weil Zope Objekte und Skripte im Web darstellt, kann es benutzt werden, um ein mächtiges, gut organisiertes, sicheres Web-API für andere entfernte Netzwerk-Anwendungsclients zur Verfügung zu stellen.

Es gibt zwei übliche Wege, Zope aus der Entfernung zu skripten. Der erste Weg ist, ein simples Fern-Prozeßaufruf-Protokoll namens *XML-RPC* zu benutzen. XML-RPC wird benutzt,um einenen Prozeß auf einer entfernten Maschine auszuführen und ein Ergebnis auf der lokalen Maschine zu erhalten. XML-RPC wurde aufgebaut, um sprachneutral zu sein und in diesem Kapitel werden sie Beispiele in Python, Perl und Java sehen.

Der zweite übliche Weg, um Zope aus der Entfernung zu skripten, läuft über irgendeinen HTTP-Client, der mit einem Skript automatisiert werden kann. Viele Sprachbibliotheken enthalten simple, skriptbare HTTP-Clients und es gibt viele Programme, die Sie HTTP von der Kommandozeile aus skripten lassen.

XML-RPC benutzen

XML-RPC ist ein einfacher Fern-Prozeßaufruf-Mechanismus, der über HTTP arbeitet und XML benutzt, um Informtionen zu verschlüsseln. XML-RPC-Clients sind für viele Programmiersprachen implementiert worden, eingeschlossen Python, Perl, Java, JavaScript und TCL.

Tiefere Informationen zu XML-RPC sind auf der XML-RPC-Website zu finden.

Alle Zope-Skripte, die von URLs aufgerufen werden können, können über XML-RPC aufgerufen werden. Grundsätzlich bietet XML-RPC ein System, um Argumente zu Skripten zu ordnen, die vom Web aus aufgerufen werden können. Wie Sie oben gesehen haben, bietet Zope seine eigenen Ordnungskontrollen, die Sie über HTTP nutzen können. XML-RPC und Zopes eigenes Ordnen erfüllen zum großen Teil dasselbe. Der Vorteil von XML-RPC-Ordnen liegt darin, daß es ein breit unterstützter Standard ist, der auch das Ordnen von Rückgabe-Werten wie von Argument-Werten unterstützt.

Hier ist ein versponnenes Beispiel, das zeigt, wie durch entfernes Skripten über XML-RPC eine Massenentlassung von Pförtnern ausgelöst werden kann.

Hier ist der Code in Python:

```
import xmlrpclib

  server = xmlrpclib.Server('http://www.zopezoo.org/')
  for employeeID in server.JanitorialDepartment.personnel():
      server.fireEmployee(employee)
```

In Perl:

```
use Frontier::Client;

  $server = Frontier::Client->new(url => "http://www.zopezoo.org/");

  $employees = $server->call("JanitorialDepartment.personnel");
  foreach $employee ( @$employees ) {

    $server->call("fireEmployee",$server->string($employee));

  }
```

In Java:

```
try {
      XmlRpcClient server = new
XmlRpcClient("http://www.zopezoo.org/");
      Vector employees = (Vector)
server.execute("JanitorialDepartment.personnel");

      int num = employees.size();
      for (int i = 0; i < num; i++) {
          Vector args = new Vector(employees.subList(i, i+1));
          server.execute("fireEmployee", args);
      }

  } catch (XmlRpcException ex) {
      ex.printStackTrace();
  } catch (IOException ioex) {
      ex.printStackTrace();
```

```
}
```

In Wirklichkeit wird das obige Beispiel vielleicht nicht korrekt laufen, weil sie höchstwahrscheinlich das Skript *fireEmployee* schützen wollen. Das bringt das Thema Sicherheit mit XML-RPC auf. XML-RPC hat keinerlei eigene Sicherheitsversorgung; dennoch kann es - weil es unter HTTP läuft - existierende HTTP-Sicherheitskontrollen übernehmen. Tatsächlich behandelt Zope eine XML-RPC-Anfrage genau wie eine normale HTTP-Anfrage, mit Rücksicht auf die Sicherheitskontrollen. Das bedeutet, daß Sie in ihrer XML-RPC-Anfrage eine Authentifizierung bieten müssen, damit Zope Ihnen Zugriff auf geschützte Skripte gewährt. Der Python-Client unterstützt zum Zeitpunkt der Niederschrift dieses Textes keine Kontrolle von HTTP-Authentifizierungs-Headern. Dabei ist es ein ziemlich trivialer Zusatz. Zum Beispiel schließt ein Artikel auf XML.com, Internet Scripting: Zope and XML-RPC, ein Patch für Pythons XML-RPC-Unterstützung ein, wo gezeigt wird, wie ihrem XML-RPC-Client HTTP-Authentifizierungs-Header hinzugefügt werden können.

Fern-Skripten mit HTTP

Jeder HTTP-Client kann zum Fern-Skriten von Zope verwendet werden.

Auf UNIX-Systemen haben sie eine Anzahl von Werkzeugen zur Verfügung, um Zope fernzuskripten. Ein einfaches Beispiel ist, *wget* zu benutzen, um die URLs von Zope-Skripten aufzurufen und *cron*, um die Skript-Aufrufe zu planen. Stellen Sie sich beispielsweise vor, Sie hätten ein Zope-Skript, das die Löwen füttert und Sie würden es gern jeden Morgen aufrufen. Sie können *wget* benutzen, um das Skript etwa so aufzurufen:

```
$ wget --spider http://www.zopezope.org/Loewen/fuettern
```

Die Option *spider* sagt *wget*, daß es die Antwort nicht als Datei speichern soll. Nehmen Sie an, ihr Skript ist geschützt und verlangt Authentifizierung. Sie können ihren Benutzernamen und das Paßwort mit *wget* übergeben, um auf geschützte Skripte zuzugreifen:

```
$ wget --spider --http_user=ZooKeeper --http_pass=SecretPhrase
http://www.zopezope.org/Loewen/fuettern
```

Lassen Sie uns nun *cron* benutzen, um dieses Kommando jeden Morgen um 8.00 Uhr aufzurufen. Bearbeiten sie ihre crontab-Datei mit dem Kommando *crontab*:

```
$ crontab -e
```

Dann fügen sie eine Zeile hinzu, um wget jeden Tag um 8.00 Uhr aufzurufen:

```
0 8 * * * wget -v --spider --http_user=ZooWaerter --
http_pass=SecretPhrase
http://www.zopezoo.org/Loewen/fuettern
```

Der einzige Unterschied zwischen der Benutzung von *cron* und dem manuellen Aufruf von *wget* ist, daß Sie den Schalter *v* benutzen sollten, wenn Sie *cron* benutzen, weil Sie die Ausgabe des Kommandos *wget* nicht interssiert.

Lassen sie uns für unser Schlußbeispiel richtig pervers werden. Weil Netzwerkfähigkeit in so viele verschiedene Systeme eingebaut ist, ist es einfach, einen Kandidaten zu finden, für den das Skripten von Zope eher unwahrscheinlich ist. Wenn Sie einen internetfähigen Toaster hätten, könnten Sie auch damit wahrscheinlich Zope skripten. Lassen sie uns Microsoft Word als unseren Beispiels-Zope-Client nehmen. Alles Notwendige besteht darin, sich mit Word darauf zu einigen, daß es einen URL kitzelt.

Der einfachste Weg, Zope mit Word zu skripten ist es, Word zu sagen, es soll ein Dokument öffnen und dann - wie in Abbildung 8.9 gezeigt - einen Zope-Skript-URL als Dateinamen einzugeben.

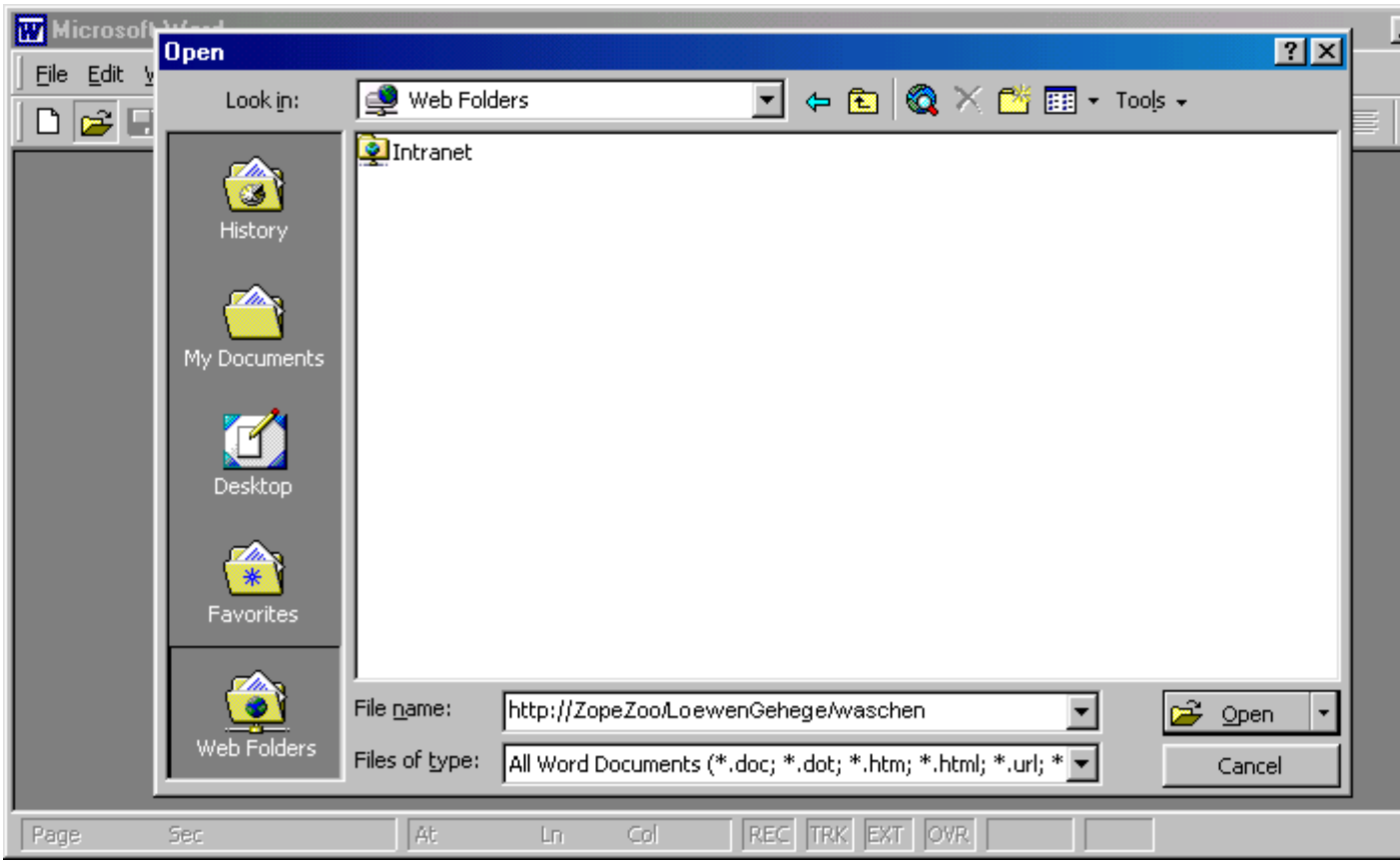


Abbildung 8.9 Einen URL mit Microsoft Word aufrufen

Word wird dann den URL laden und die Ergebnisse des Zope-Skript-Aufrufs zurückgeben. Trotz der Tatsache, daß Word Sie auf diese Weise keine Argumente POSTen läßt, können sie GET-Argumente übergeben, indem Sie sie als Teil des URLs eingeben.

Sie können dieses Verhalten sogar kontrollieren, indem sie Words eingebautes Visual Basic-Scripting benutzen. Hier ist zum Beispiel ein Visual Basic-Fragment, das Word dazu bringt, ein neues Dokument zu öffnen, indem es einen Zope-Skript-URL benutzt:

```
      Documents.Open
FileName:="http://www.zopezoo.org/LoewenGehege/waschen?seife_verwenden=1&wa
sser_temp=heiss"
```

Sie könnten Visual Basic nutzen, um Zope-Skript-URLs auf viele verschiedene Weise aufzurufen.

Zopes URL zur Übersetzung per Skript-Aufruf ist der Schlüssel zum Fern-skripten. Weil Sie Zope so einfach mit simplen URLs kontrollieren können, können Sie Zope auch einfach mit fast jedem netzwerkfähigen System skripten.

Schluß

Zope bietet Skripten mit Python und Perl. Mit Skripten können Sie Zope-Objekte kontrollieren und ihre Anwendungslogik, Daten und Darstellung verbinden. Sie können auch ernsthafte Programmierungsaufgaben wie Grafik-Verarbeitung und XML-Parsing durchführen.

Im nächsten Kapitel werden sie etwas über den ZCatalog lernen, Zopes eingebaute Suchmaschine.

Zopebuch: Inhaltsverzeichnis

Dieses Dokument ist momentan noch nicht vollständig übersetzt.

Kapitel 11: Inhalt durchsuchen und kategorisieren

Mit Zope wird eine eingebaute Suchmaschine namens ZCatalog geliefert, mit der man viele Zope-Objekte in Kategorien einteilen und nach ihnen suchen kann. Die Suchmaschine kann auch für externe Daten z. B. aus relationalen Datenbanken, Dateien und anderen Webseiten verwendet werden. Neben der Suchfunktion dient der Katalog auch zum Verwalten von Objektgruppen.

Der Katalog enthält ein umfangreiches Suchformular, das eine Volltextsuche ermöglicht und gleichzeitig in mehreren Suchindizes agieren kann. Ausserdem behält ZCatalog Veränderungen der Metadaten von indizierten Objekten im Auge. ZCatalog wird in der Regel für folgende Vorgänge verwendet:

Massenkatalogisierung
: Katalogisierung vieler Objekte auf einmal.

Automatische Katalogisierung
: Katalogisierung von Objekten bei ihrer Erstellung und Veränderung.

Einführung in die Massenkatalogisierung

Sehen wir uns an, wie man mit ZCatalog nach Dokumenten suchen kann. Wenn man eine grosse Zahl von Objekte auf einmal katalogisiert, nennt man dies *Massenkatalogisierung*. Die Massenkatalogisierung erfordert drei Arbeitsschritte:

- Erstellen eines ZCatalogs
- Zu katalogisierende Objekte bestimmen und indizieren
- Erstellen einer web-basierten Schnittstelle für die Suche im Katalog

Erstellen Sie mit Hilfe der Produktauswahlliste ein *ZCatalog* Objekt. Sie gelangen zu dem in Abbildung 9-1 gezeigten Erstellungsformular.

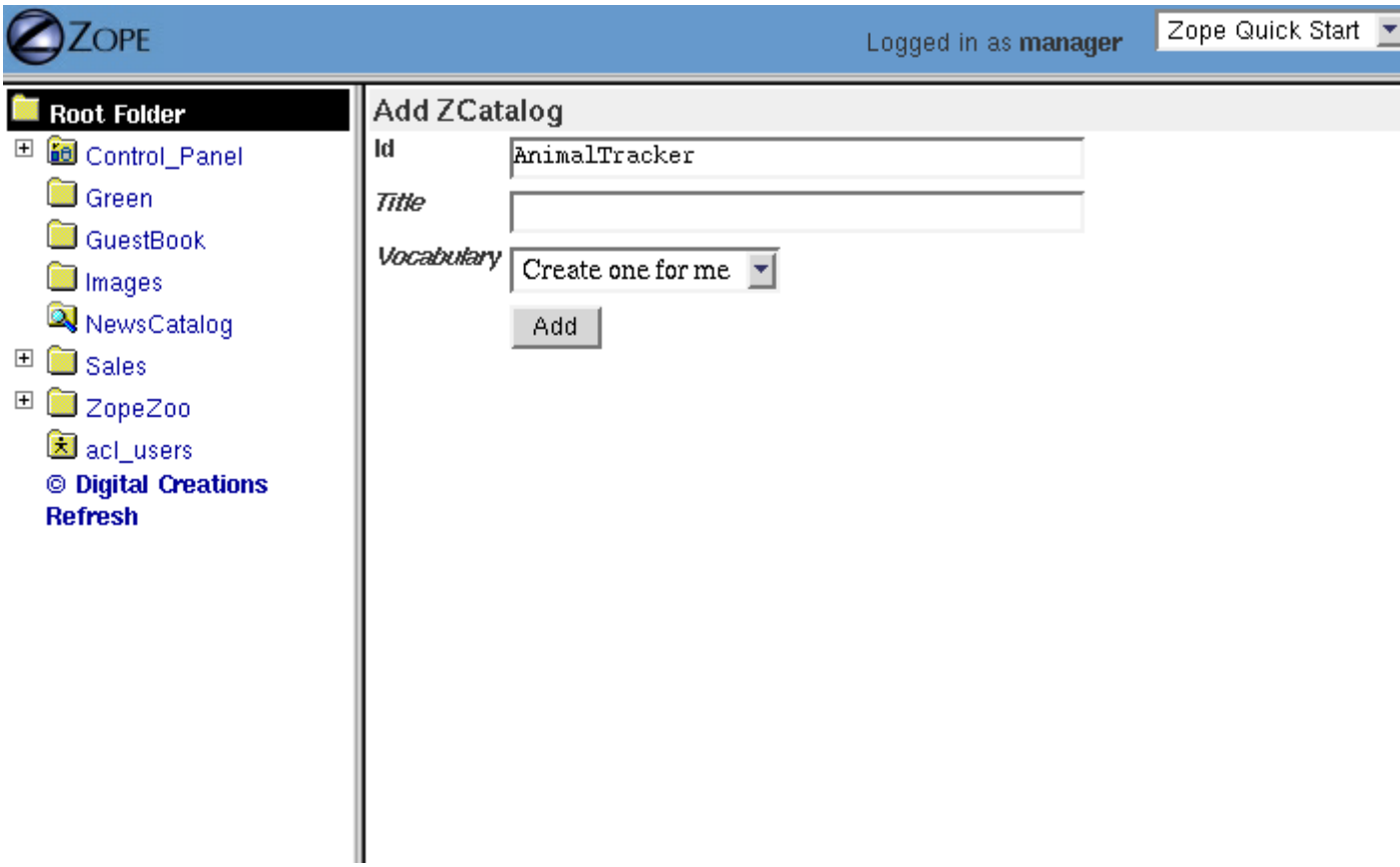


Abbildung 9-1 ZCatalog-Erstellungsformular

Das Erstellungsformular frägt nach einer *Id* und einem *Title*. Das dritte Formular-Feld ist die *Vocabulary* Auswahl-Box. Lassen Sie zunächst die Option "Create one for me" ausgewählt. Geben Sie als Id "AnimalTracker" ein und klicken Sie auf *Add* um das neue Katalog-Objekt zu erstellen. Das Katalog-Symbol ähnelt dem eines Ordners mit einer kleinen Lupe darin. Wählen Sie das Symbol *AnimalTracker* um die Ansicht *Contents* auf dem Bildschirm anzuzeigen.

Ein ZCatalog sieht ähnlich aus wie ein Ordner, enthält aber zusätzliche Reiter. Sechs Reiter entsprechen denen eines Ordners. ZCatalog-Objekte haben folgende Ansichten: *Contents*, *Catalog*, *Properties*, *Indexes*, *MetaData*, *Find Objects*, *Advanced*, *Undo*, *Security* und

Ownership. Wenn man ein ZCatalog-Objekt anklickt, wird zuerst die *Contents*-Ansicht angezeigt. Man kann dort neue Objekte hinzufügen, die der ZCatalog wie ein normaler Ordner beherbergt. Objekte, die sich im Katalog befinden, sind aber nicht zwangsläufig über den Katalog suchbar.

Den ZCatalog haben wir angelegt, bestimmen wir nun die zu katalogisierenden Objekte. Gehen wir von einer Webseite über einen Zoo mit Informationen über Tiere aus. Erstellen Sie zwei DTML-Dokumente, die Informationen über Reptilien und Amphibien beinhalten:

Title: Chilenischer Vier-Augen-Frosch

> Der Chilenische Vier-Augen-Frosch hat zwei helle Punkte auf dem Rumpf, die wie grosse Augen aussehen. Wenn der Frosch sitzt verbergen die Schenkel diese Punkte. Wenn sich ein Feind näher, senkt der Frosch seinen Kopf und hebt seinen Rumpf, so dass er wie ein viel grösserer Kopf aussieht. Frösche sind Amphibien.

Title: Teppich-Python

> Die *Morelia spilotes variegata* hat eine durchschnittliche Länge von 2,4 Meter. Sie ist eine mittelgrosse Python mit schwarzgrauen Flecken-, Kreuzband- und Streifenmustern oder eine Kombination derselben auf einem hellen gelbschwarzen braunen Hintergrund. Schlangen sind Reptilien.

Besucher Ihres Zoos möchten nun nach Informationen über die Tiere suchen können. Eifrige Herpetologen möchten gerne wissen, ob im Zoo ihre Lieblings-Schlange lebt, deshalb müssen Sie eine Stichwortsuche bereitstellen, die alle Dokumente anzeigt, die die Stichworte enthalten. Die Suche nach Informationen ist eine der hauptsächlichen Web-Anwendungen.

Der vorhin erstellte *AnimalTracker*-ZCatalog kann alle Dokumente des Zope-Webangebots katalogisieren und ermöglicht dem Benutzer eine Suche nach Stichwörtern. Wählen Sie den *AnimalTracker*-ZCatalog aus und klicken sie auf den Reiter *Find Objects*.

In dieser Ansicht teilen Sie dem ZCatalog mit, welche Art von Objekten katalogisiert werden sollen. Wir wollen nun alle DTML-Dokumente katalogisieren, wählen Sie dazu *DTML Document* aus der Mehrfachauswahl *Find objects of type* und klicken Sie auf *Find and Catalog*.

Der ZCatalog sucht nun alle DTML-Dokumente beginnend in dem Verzeichnis, in dem er selbst plaziert ist. Er durchsucht den Ordner, danach alle Unterverzeichnisse, deren Unterverzeichnisse, usw. Wenn diese viele Objekte enthalten, kann der Vorgang durchaus eine längere Zeit in Anspruch nehmen.

Nach einer Zeitspanne erscheint die *Catalog*-Ansicht und teilt dem Anwender den Status des Katalogisierungsvorganges mit.

Darunter erscheint eine Liste über die katalogisierten Objekte. Es sind ausschliesslich DTML-Dokumente. Man kann die Objekte anklicken um sich sicher zu sein, dass es die gewünschten sind.

Nun haben wir den Arbeitsschritt zur Katalogisierung von Objekten beendet. Die Objekte sind in der ZCatalog-Datenbank abgelegt. Jetzt gehen wir zum dritten Arbeitsschritt über, in dem ein Suchformular und eine Ergebnisseite für den ZCatalog erstellt wird.

Such- und Ergebnis-Formulare

Gehen Sie in das *AnimalTracker*-Katalogobjekt und wählen Sie *Z Search Interface* aus der Produktauswahlliste. Geben Sie den *AnimalTracker*-ZCatalog als suchbares Objekt an, wie es in Abbildung 9-2 gezeigt ist.

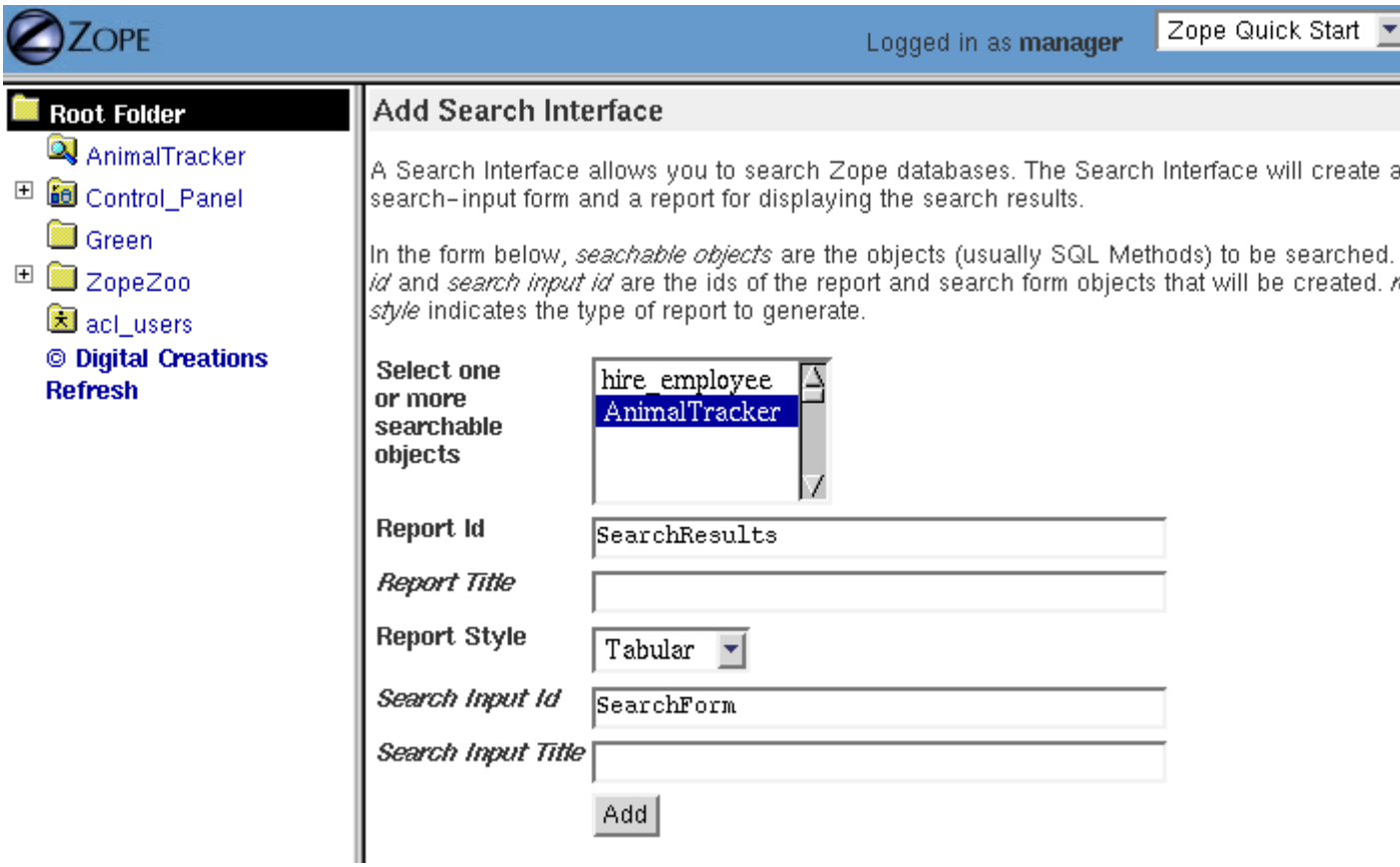


Abbildung 9-2 Erstellen eines Suchformulares für einen ZCatalog

Benennen Sie die *Report Id* "Suchergebnisse" und die *Search Input Id* "Suchformular" und klicken Sie dann auf *Add*. Dadurch werden zwei neue DTML-Methoden im ZCatalog namens *Suchformular* und *Suchergebnisse* erstellt.

Diese Objekte sind zwar im ZCatalog *enthalten*, werden aber nicht vom ZCatalog *katalogisiert*. Der *AnimalTracker* enthält nur katalogisierte DTML-Dokumente. Die Suchformular- und Suchergebnisse-Methoden sind lediglich eine Benutzerschnittstelle für die Suche innerhalb der Dokumente mit Informationen über Tiere im Katalog. Sie können das überpüfen, indem Sie sich den Reiter *Cataloged Objects* ansehen. Dort werden die beiden DTML-Methoden nicht aufgeführt.

Wählen Sie die Methode *Suchformular* und klicken Sie dort auf den Reiter *View*. Das Formular enthält einige Felder. Es gibt ein Suchfeld für jeden Index im ZCatalog. Indizes werden im nächsten Abschnitt beschrieben. Klicken Sie für unser Beispiel in das *PrincipiaSearchSource* Feld. Alle anderen Felder des Formulars können leer gelassen werden.

Sie können nach Dokumenten suchen, die im ZCatalog *AnimalTracker* erfasst sind, indem Sie Suchwörter in das *PrincipiaSearchSource* Feld eingeben. Geben Sie zum Beispiel das Wort "Reptilien" ein. Der ZCatalog *AnimalTracker* wird durchsucht und eine einfache Tabelle ausgegeben, die alle Dokumente anzeigt, die das Wort "Reptilien" enthalten. Die Teppich-Python sollte in der Tabelle angezeigt werden. Man kann auch mehrere Wörter angeben, die Suche geht dann über alle Dokumente, die eines der Wörter enthalten, z. B. "Reptilien Amphibien". Es sollten nun die beiden Dokumente über den Vier-Augen-Frosch und die Teppich-Python angezeigt werden. Herzlichen Glückwunsch, damit haben Sie erfolgreich einen Katalog angelegt, Dokumente darin indiziert und eine Suche auf die Dokumente ausgeführt.

Kataloge konfigurieren

Es gibt mit ZCatalog weitaus mächtigere und komplexere Suchmöglichkeiten als wir bisher ausgeführt haben. Sehen wir uns an, wie ein Katalog seine Information abspeichert. Mit diesem Wissen kann man sich den Katalog so zurechtschneidern, dass er die Art Suche zur Verfügung stellt, die man haben möchte.

Indizes definieren

In ZCatalog werden Informationen über Objekte und ihre Inhalte in schnellen Datenstrukturen, die *Indizes* genannt werden, gespeichert. Indizes können viele Informationen sehr schnell ablegen und wiederfinden. Man kann verschiedene Indizes definieren, die verschiedene Informationen über die Objekte enthalten können. Zum Beispiel kann ein Index den Inhalt von DTML-Dokumenten, ein anderer Objekte, die bestimmte Eigenschaften haben, einschliessen.

Bei einer Suche im ZCatalog werden nicht die Inhalte der einzelnen Objekte durchsucht, das wäre bei einer grossen Anzahl von Objekten viel zu aufwändig. Vor einer Suche untersucht der ZCatalog die Objekte und sammelt genau die Informationen für die er eingerichtet ist. Diesen Vorgang nennt man *indizieren*. Ab diesem Zeitpunkt kann man nach bestimmten Kriterien suchen, der ZCatalog liefert genau die Objekte zurück, auf die die Kriterien zutreffen.

Denken Sie an einen Index in einem Buch. Man kann in einem Buch zum Beispiel nach dem Wort *Python* suchen und erhält etwa folgende Angabe:

```
Python: 23, 67, 227
```

Das Wort *Python* kommt hier auf drei Seiten vor. Ein Index in Zope arbeitet ähnlich, ausser dass er das Suchmuster - im Beispiel das Wort *Python* - in einer Liste von Objekten sucht anstatt auf Buchseiten.

In Zope können Indizes von einem Katalog hinzugefügt und entfernt werden. Dazu kommt eine Index-Schnittstelle, die in Abbildung 9-3 gezeigt ist, zum Einsatz:

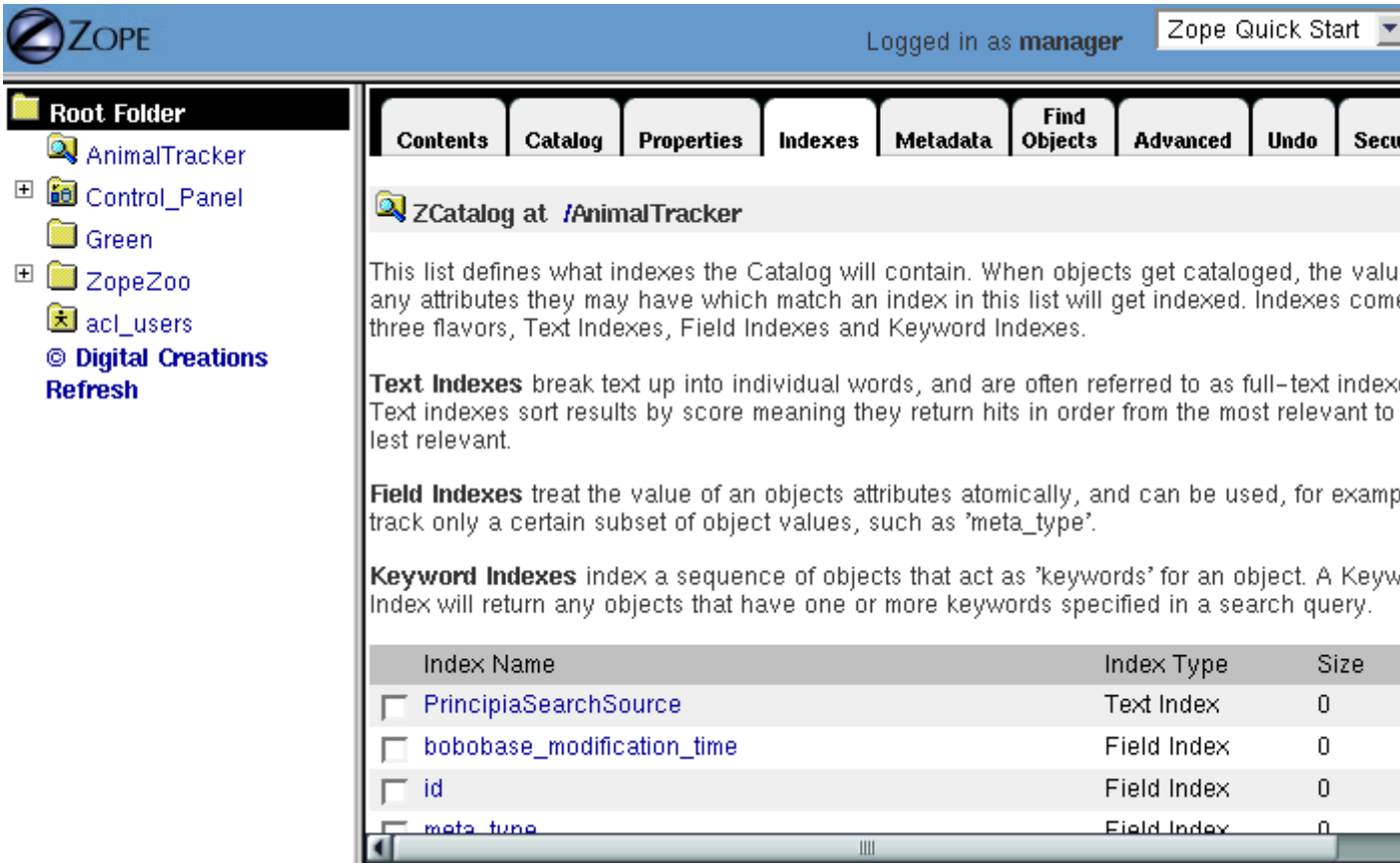


Abbildung 9-3 Indizes verwalten

ZCatalog bringt bereits einige vordefinierte Indizes mit. Jeder Index hat einen Namen (z. B. *PrincipiaSearchSource*) und einen Typ (z. B. *TextIndex*).

Beim Katalogisieren eines Objektes wird mit Hilfe jedes Index' das Objekt untersucht. Der Katalog sieht in Attributen und Methoden nach, um den Wert eines Objekts für jeden Index zu ermitteln. Bei DTML-Dokumenten wird zum Beispiel beim Katalogisieren mit einem *PrincipiaSearchSource*- Index die Methode *PrincipiaSearchSource* des Dokuments aufgerufen. Der Katalog speichert deren Ergebnisse in seinem *PrincipiaSearchSource*-Index ab. Wenn der Katalog in dem Objekt keine passende Methode finden kann, wird das Objekt ignoriert. Das Objekt wird also nicht im Index abgelegt. Es gibt vier Index-Arten:

Text-Index (TextIndex)
: Dieser Index durchsucht ein Objekt nach Texten; er wird für eine Volltextsuche verwendet.

Feld-Index (FieldIndex)
: Dieser Index durchsucht ein Objekt nach bestimmten Werten, er wird für Datums-Objekte, Zahlen oder bestimmte Zeichenketten verwendet.

Schlagwort-Index (KeywordIndex)
: Dieser Index sucht nach Gruppen bestimmter Werte. Er ähnelt dem Feld-Index, ermöglicht aber eine Suche nach Gruppen von Werten.

Pfad-Index (PathIndex)
: Dieser Index sucht nach allen Objekten, die bestimmte URL-Bestandteile enthalten. Man kann z. B. alle Objekte suchen, deren Pfad mit `/Tiere/Zoo` beginnt.

Auf die verschiedenen Indix-Typen wird weiter hinten in diesem Kapitel näher eingegangen. Auf der *Indexes*-Ansicht eines ZCatalogs können neue Indizes angelegt werden. Zunächst wählt man in dem Auswahlfeld einen Index-Typ aus und klickt auf *Add*, falls der Browser nicht bereits von selbst das Erstellungs-Formular anzeigt. Danach füllt man das Feld *Id* aus und klickt auf Add. Man erhält einen neuen leeren Index in dem ZCatalog. Um diesen mit Informationen muss man den Indizierungsvorgang erneut ausführen. Dies erreicht man indem man auf die Schaltfläche *Update Catalog* in der Ansicht *Advanced* des ZCatalogs klickt. Der Vorgang kann eine Weile dauern, wenn man viele Objekte hat.

Einen Index kann man aus dem Katalog auch wieder entfernen, indem man das Kontrollkästchen links neben dem Namen des Index aktiviert und auf *Remove Index* klickt. Es werden sowohl der Index als auch sein Inhalt entfernt. Der Vorgang kann mit Undo auch wieder zurückgenommen werden.

Metadaten definieren

Ein ZCatalog kann nicht nur Informationen über Objekte indizieren sondern auch in einer *tabellarischen Datenbank* genannt *Metadaten-Tabelle* abspeichern. Die *Metadaten-Tabelle* funktioniert ähnlich wie eine Tabelle in einer relationalen Datenbank. Sie besteht aus einer oder mehreren *Spalten*, die das *Schema* der Tabelle definieren. Die Tabelle ist mit *Zeilen* gefüllt, die Informationen über die katalogisierten Objekte enthalten können. Die Metadaten-Spalten müssen nicht zwingend Indizes im Katalog entsprechen. Mit Indizes kann man suchen, Metadaten ermöglichen einen Bericht über die Suchergebnisse.

Die Metadaten-Tabelle ist für die Generierung des Berichtes über die Suchergebnisse nützlich. Sie verfolgt die Informationen über Objekte, die in den Suchberichten erscheinen. Wenn man z. B. eine Metadaten-Tabellen-Spalte namens *absolute_url* erstellt, können die Berichte diese Informationen benutzen um einen Link zu den gesuchten Objekten zu erzeugen.

Man legt eine neue Metadaten-Tabellen-Spalte an, in dem man den Namen in der Ansicht *Metadata* bei *Add Metadata* einträgt und auf die Schaltfläche *Add* klickt. Um eine Spalte zu entfernen, wählt man das Kontrollkästchen beim Spaltennamen an und klickt dann auf die Schaltfläche *Delete*. Die Spalte und ihr Inhalt in jeder Zeile werden entfernt. Die Funktion kann mit der Undo-Funktion rückgängig gemacht werden. Sehen wir uns nun die Suche im Katalog etwas genauer an.

Suche im Katalog

Man kann einen Katalog durchsuchen, in dem man ihm einen Suchtext übergibt. Dieser Suchtext beschreibt, nach was man in einem oder mehreren Indizes suchen möchte. Der Katalog kann diese Information aus der Web-Anfrage entnehmen, es ist aber auch eine Übergabe aus einem DTML- oder Python-Script denkbar. Der Katalog antwortet auf die Anfrage mit einer Liste von Einträgen, die zu den katalogisierten Objekten gehören, auf die die Suchanfrage passt.

Suchen mit Formularen

In diesem Kapitel verwendeten wir das *Z Search Interface* um automatisch ein Such- und Antwort-Formular zu erstellen, mit dem man im Katalog suchen kann (die Zusammenhänge zwischen Formular und Action wurden bereits in Kapitel 4 "Dynamischer Inhalt mit DTML" behandelt). Das *Z Search Interface* generiert ein sehr einfaches Suchformular und einen sehr einfachen Bericht. Mit diesen beiden Methoden kann man die Funktionsweise des Katalogs und Anpassungsmöglichkeiten der Suchschnittstelle kennenlernen.

Im weiteren behandeln wir einen Katalog, der News-Einträge enthält. Jeder News-Eintrag hat einen Inhalt, einen Autor und ein Erstellungsdatum. Der Katalog verfügt über drei Indizes, jeweils einen für jedes dieser Attribute. Der Inhalts-Index ist ein Text-Index, für Autor und Datum wird ein Feld-Index verwendet. Mit folgendem Suchformular kann man Anfragen an den Katalog stellen:

```
<dtml-var standard_html_header>

<form action="Report" method="get">
<h2><dtml-var document_title></h2>
Abfrageparameter eingeben:<br><table>

<tr><th>Inhalt</th>
    <td><input name="Inhalt" width=30 value=""></td></tr>
<tr><th>Autor</th>
    <td><input name="Autor" width=30 value=""></td></tr>
<tr><th>Datum</th>
    <td><input name="Datum"  width=30 value=""></td></tr>

<tr><td colspan=2 align=center>
<input type="SUBMIT" value="Abfrage senden">
</td></tr>
</table>
</form>

<dtml-var standard_html_footer>
```

Dieses Formular besteht aus drei Eingabefeldern namens *inhalt*, *autor* und *datum*. Die Benennungen müssen zu den Namen der Indizes im Katalog passen. Folgender Bericht liefert das Ergebnis zu der Suche:

```
<dtml-var standard_html_header>

<table>
  <dtml-in NewsCatalog>
  <tr>
    <td><dtml-var Autor></td>
    <td><dtml-var Datum></td>
  </tr>
  </dtml-in>
</table>

<dtml-var standard_html_footer>
```

Einige Punkte müssen wir näher betrachten. Das Herz des Berichts ist das Tag *in*:

```
        <dtml-in NewsCatalog>
```

Dieses Tag ruft den Katalog *NewsCatalog* auf. Beachten Sie, dass nicht alle Parameter aus dem Suchformular (Inhalt, Autor, Datum) im Ergebnis-Bericht auftauchen. Zope stellt automatisch sicher, dass die Parameter aus dem Suchformular dem Katalog übergeben werden. Man muss lediglich selbst dafür sorgen, dass der Ergebnis-Bericht den Katalog aufruft. Zope entnimmt die Suchwörter der Web-Anfrage und reicht sie an den Katalog weiter.

Der Katalog liefert eine Folge von *Datensatz-Objekten* (ähnlich wie ZSQL-Methoden) zurück. Diese Datensatz-Objekte passen zu den Treffern der Suche, also den Objekten die zu den von Ihnen eingegebenen Kriterien passen. Damit ein Datensatz in einer Suche gefunden wird, muss er zu allen Kriterien jedes einzelnen Index passen. Wenn man also einen Autor und ein paar Suchworte für den Inhalt angibt, liefert der Katalog nur die Datensätze zurück, die sowohl zu dem Autor als auch zum Inhalt passen.

Datensatz-Objekte haben ein Attribut für jede Spalte in einer Datenbank-Tabelle. Katalog-Datensatz-Objekte aarbeiten ähnlich, außer, dass sie ein Attribut für jede Spalte der Metadaten-Tabelle enthalten. In Wirklichheit ist der Zweck der Metadatentabelle die Definition eines Schemas für die Datensatzobjekte, die Anfragen im Katalog zurückliefern.

Suchanfragen mit Python

Mit DTML lassen sich von Suchformularen leicht Anfragen an einen Katalog realisieren. DTML stellt sicher, dass die Suchparameter an den Katalog korrekt weitergeleitet werden.

Manchmal möchte man einen Katalog nicht von einem Web-Formular aus durchsuchen; stattdessen soll an einer Stelle ein anderes Programmmodul eine Anfrage an den Katalog stellen. Nehmen wir zum Beispiel an, dass im Zope Zoo ein Informationsbalken angebracht werden soll, der nur die News-Einträge anzeigt, die zu den gerade angezeigten Tieren gehören. Wie früher bereits beschrieben, baut der Zope Zoo auf Ordner auf, die passend zu den Tierarten aufgeteilt sind. Jede Ordner-ID bezeichnet die Tierart, die sich in dem Ordner befindet. Der In unserem Beispiel sollen alle News-Einträge angezeigt werden, die die ID des aktuellen Ordners enthalten. Das folgende Skript namens *relevanteSektionsNachrichten* stellt Suchanfragen an den NewsCatalog mit der aktuellen Ordner-Id:

```
        ## Script (Python) "relevanteSektionsNachrichten"
        ##
        """ Liefert Nachrichten, die für die aktuelle Ordner-Id relevant
sind. """
        id=context.getId()
        return context.NewsCatalog({'Inhalt' : id})
```

Das Skript stellt Anfragen an den NewsCatalog, indem es ihn wie eine Methode aufruft. Kataloge erwarten eine *Zuordnung* (mapping) als erstes Argument, wenn sie aufgerufen werden. Das Argument ordnet den Namen eines Index einem Such-Objekt zu. In diesem Fall wird im Index *Inhalt* nach allen Neuigkeiten-Einträgen gesucht, die den Namen es aktuellen Ordners enthalten. Um das Skript in dem Informationsbalken einzubinden, muss das standard_html_header-Objekt im Zope-Zoo wie folgt angepasst werden:

```
<html>
<body>
<dtml-var style_sheet>
<dtml-var navigation>
<ul>
<dtml-in relevanteSektionsNachrichten>
  <li><a href="&dtml-absolute_url;"><dtml-var title></a></li>
</dtml-in>
</ul>
```

Diese Methode geht davon aus, dass *absolute_url* und *title* als Metadaten-Spalten im NewsCatalog definiert sind. Der Informationsbalken an der Seite zeigt nun eine einfache Liste von Neuigkeiten-Einträgen anzeigen, die die Id des aktuell angezeiten Ordners enthalten.

Einzelheiten beim Suchen und Indexieren

Zuvor haben Sie gesehen, dass der Katalog drei Indexarten unterstützt: Text-Indizes, Feld-Indizes und Schlagwort-Indizes. Wir wollen diese Indizes näher untersuchen, um zu verstehen, wofür sie gut sind und wie sie abfragbar sind.

Text-Indizes durchsuchen

Ein Text-Index wird zum Indexieren von Texten genutzt. Nach dem Indexieren können Sie den Index nach Objekten durchsuchen, die bestimmte Wörter enthalten. Text-Indizes unterstützen eine reichhaltige Grammatik, mit der fortgeschritteneres Suchen als nur nach einem Wort möglich ist. Der Text-Index von ZCatalog kann:

- suchen mit Bool'schen Ausdrücken wie "Wort1 AND Wort2". Dieser Ausdruck sucht nach allen Objekten, die *sowohl* "Wort1" *als auch* "Wort2" enthalten. Gültige Bool'sche Operatoren sind AND, OR und AND NOT.
- durch Klammerung wie "(Wort1 AND Wort2) OR Wort3" die Reihenfolge beim Suchen steuern. Dieser Ausdruck liefert Objekte, die sowohl "Wort1" wie auch "Wort2" enthalten, *aber* auch diejenigen mit "Wort3".
- mit einfachen Jokern wie "Z" suchen, wenn Sie eine besondere Art *Vokabular* (wird etwas später erläutert) benutzen; dieses Suchen liefert alle Wörter, die mit "Z" beginnen.

Diese fortschrittlichen Leistungsmerkmale können gemischt werden. Zum Beispiel liefert "((Robert AND Onkel) AND NOT Zoo*)" alle Objekte, die die Wörter "Robert" und "Onkel", aber nicht Wörter enthalten, die mit "Zoo" beginnen wie "Zoologe", "Zoologie" und "Zoo" selbst.

Das Abfragen eines Text-Indexes mit diesen fortschrittlichen Leistungsmerkmalen geschieht genauso wie mit den originalen einfachen Möglichkeiten. In ein HTML- Suchformular für DTML-Dokumente geben Sie z. B. "Koala AND Löwe" ein, und Sie erhalten alle Dokumente, in denen die Wörter Koala und Löwe vorkommen. Abfragen auf einen *TextIndex* arbeiten von Python aus ähnlich. Angenommen, Ihr Skript *relevanteSektionsNachrichten* soll keine Nachrichten bereitstellen, die das Wort "katastrophal" enthalten:

```
## Script (Python) "relevanteSektionsNachrichten"
##
""" Liefert relevante, nicht-katastrophale Nachrichten """
id=context.getId()
return context.NewsCatalog(
         {'Inhalt' : id + ' AND NOT katastrophal'}
        )
```

Text-Indizes sind sehr mächtig. Wenn sie mit den später in diesem Kapitel beschriebenen Automatischen Katalogisierungsmustern kombiniert werden, erhalten sie die Möglichkeit einer automatischen Volltextsuche aller Ihrer Objekte, wenn Sie sie anlegen und editieren.

Vokabulare

Vokabulare werden von Text-Indizes benutzt. Ein Vokabular ist ein Objekt, das sprachspezifische Text-Indexierungsmöglichkeiten verwaltet. Damit der ZCatalog mit jeder Sprache arbeiten kann, muss er das Verhalten dieser Sprache kennen. Zum Beispiel gilt für alle Sprachen:

- Sie haben unterschiedliche Konzepte zum Begriff *Wort*. Im Englischen und vielen anderen Sprachen sind Wörter durch die Abgrenzung mit "weißen Leerstellen" festgelegt, in anderen Sprachen wie im Chinesischen und Japanischen werden sie durch ihren Gebrauch im Zusammenhang definiert.
- Sie haben verschiedene Konzepte für den Begriff *Sperrwort*. Ein Sperrwort (engl. stop word) ist ein gebräuchliches Wort, das bei der Indexierung übergangen werden soll. Das französische Wort *nous* ist außergewöhnlich gebräuchlich in französischen Texten und sollte daher wahrscheinlich nicht berücksichtigt werden, aber im Englischen mag seine Katalogisierung ganz sinnvoll sein, weil es sehr selten vorkommt.
- Sprachen haben unterschiedliche Synonyme. Das Synonym-Paar *automobile/car* würde keinen Sinn in irgendeiner anderen Sprache als Englisch machen.
- Sprachen haben unterschiedliche Konzepte bei den Wortstammbildung. Im Englischen ist es für Text-Indizes üblich, Endungen wie *ing* von Wörtern abzutrennen, so dass *bake* und *baking* als übereinstimmende Wörter aufgefasst werden. Dies wird Wortstamm-Bildung genannt. Das Abtrennen dieser Endungen würde nur für Englisch Sinn ergeben. Andere Sprachen müssen ihre eigene Wortstamm-Bildung berücksichtigen (oder sie weglassen).

Vorhandene Vokabulare

Für den ZCatalog sind zur Zeit verfügbar:

Flache Vokabulare
: Flache Vokabulare sind sehr einfach und erledigen minimale Aufgaben für die englische Sprache.

Geklumpte Vokabulare
: Geklumpte Vokabulare sind komplexere Vokabulare, mit denen Joker-Abfragen auf englische Texte möglich werden. Der Nachteil von ihnen ist ihr sehr viel größerer

Platzbedarf im Hauptspeicher und in der Datenbank gegenüber den flachen Vokabularen.

Hinter den Vokabularen steckt die Idee, die Indexierung von Texten auf jede Sprache anpassen zu können. Daher können in Zukunft möglicherweise andere Sprachen unterstützt werden, wenn jemand ein Vokabular für diese Sprache erstellt. Das Erstellen eines eigenen Vokabulars ist ein Fortgeschrittenen-Thema und geht über den Themenkreis dieses Buchs hinaus.

Vokabulare benutzen

Wenn Sie einen neuen ZCatalog erstellen, enthält das ZCatalog-Anlegeformular einen Auswahlkasten, mit dem Sie das Vokabular auswählen. Wenn Sie kein Vokabular auswählen, erzeugt ZCatalog automatisch ein flaches Vokabular und fügt es zum ZCatalog-Inhalt (dies können Sie in der Ansicht *Contents* des AnimalTracker sehen, der von Ihnen für die Beispiele in diesem Kapitel erzeugt wurde).

Um ein geklumptes Vokabular oder irgendein anderes Vokabular zu benutzen, muss es von Ihnen erzeugt werden, bevor der Katalog angelegt wird, der es benutzen soll. Ein ZCatalog kann jedes Vokabular benutzen, das zu seinem Inhalt gehört, oder jedes Vokabular, das er in der Zope-Ordner-Hierarchie *über* sich finden kann.

Suchen in Feld-Indizes

Feld-Indizes unterscheiden sich geringfügig von Text-Indizes. Ein Text-Index benutzt den Wert, den er in Ihrem Objekt findet, z. B. den Text einer Nachrichteneinheit. Das heißt, er teilt den Text in einzelne Wörter und indexiert alle einzelnen Wörter.

Ein Feld-Index teilt demgegenüber den gefundenen Wert nicht auf. Er indexiert den gesamten gefundenen Wert. Dies ist nützlich für Objekte, denen bestimmte Eigenschaften zugeordnet sind.

In dem Nachrichtenbeispiel erzeugten Sie zwei Feld-Indizes, *Datum* und *Autor*. Mit dem bestehenden Suchformular sind diese Felder nicht besonders nützlich. Um sie effektiver zu nutzen, müssen Sie das Suchformular etwas ändern. Bevor Sie dies tun, wollen wir einige Anwendungsfälle für diese Indizes betrachten.

Der *Datum*-Index ermöglicht die Suche nach Nachrichten anhand des Zeitpunkts, an dem sie angelegt wurden. Das vorhandene Suchformular ist nicht nützlich, da es als Eingabe die bis auf die Sekunde *genaue* Zeitangabe verlangt, um bei der Suche Ergebnisse zu erhalten. Das ist offensichtlich nicht sehr nützlich. Besser würde die Suche nach einem *Zeitraum* sein, wie z. B. nach Nachrichten, die in den letzten 24 Stunden oder im letzten Monat hinzugefügt wurden.

Der *Autor*-Index ermöglicht die Suche nach Nachrichten von bestimmten Autoren. Allerdings müssen Sie die genaue Schreibweise des Autorennamens kennen. Daher würde es besser sein, wenn Sie die Auswahl aus einer Liste aller *vertretenen* Autoren treffen können.

Feld-Indizes sind dafür bestimmt, das Suchen in beiden Fällen zu erleichtern: die Suche innerhalb eines Bereichs und die Suche aufgrund im Index gespeicherter Werte. Um hiervon

Gebrauch zu machen, müssen Sie das Suchformular nur ein wenig ändern. Versuchen wir als erstes Beispiel die Suche mit einem Zeitraum.

Ähnlich wie Text-Indizes können den Feld-Indizes spezielle Optionen mitgeteilt werden, mit denen dann diese Leistungsmerkmale bereitgestellt werden können. Sie sind als Formularteile aufzunehmen, damit sie zu Katalog-Abfragen umgesetzt werden. Hier nehmen wir das Suchformular, das im vorigen Abschnitt *Suchen mit Formularen* benutzt wurde, und ergänzen es mit einem Formularteil, mit dem die Suche nach Nachrichten seit "gestern". "letzter Woche", "letztem Monat", "letztem Jahr" oder "irgendwann" ermöglicht wird:

```
        <dtml-var standard_html_header>

        <form action="Report" method="get">
        <h2><dtml-var document_title></h2>
        Suche nach Nachrichten:<br><table>

        <tr><th>Inhalt</th>
            <td><input name="Inhalt" width=30 value=""></td></tr>
        <tr><th>Autor</th>
            <td><input name="Autor" width=30 value=""></td></tr>
        <tr>
          <td><p>Geändert seit:</p></td>
          <td>
            <input type="hidden" name="date_usage" value="range:min">
            <select name="Datum:date">
              <option value="<dtml-var expr="ZopeTime(0)"
>">irgendwann</option>
              <option value="<dtml-var expr="ZopeTime() - 1"
>">gestern</option>
              <option value="<dtml-var expr="ZopeTime() - 7" >">letzter
Woche</option>
              <option value="<dtml-var expr="ZopeTime() - 30" >">letztem
Monat</option>
              <option value="<dtml-var expr="ZopeTime() - 365" >">letztem
Jahr</option>
            </select>
          </td>
        </tr>

        <tr><td colspan="2" align="center">
        <input type="SUBMIT" value="Abfrage senden">
        </td></tr>
        </table>
        </form>
        <dtml-var standard_html_footer>
```

Dadurch sollte das Suchformular wie Abbildung 9-4 aussehen.

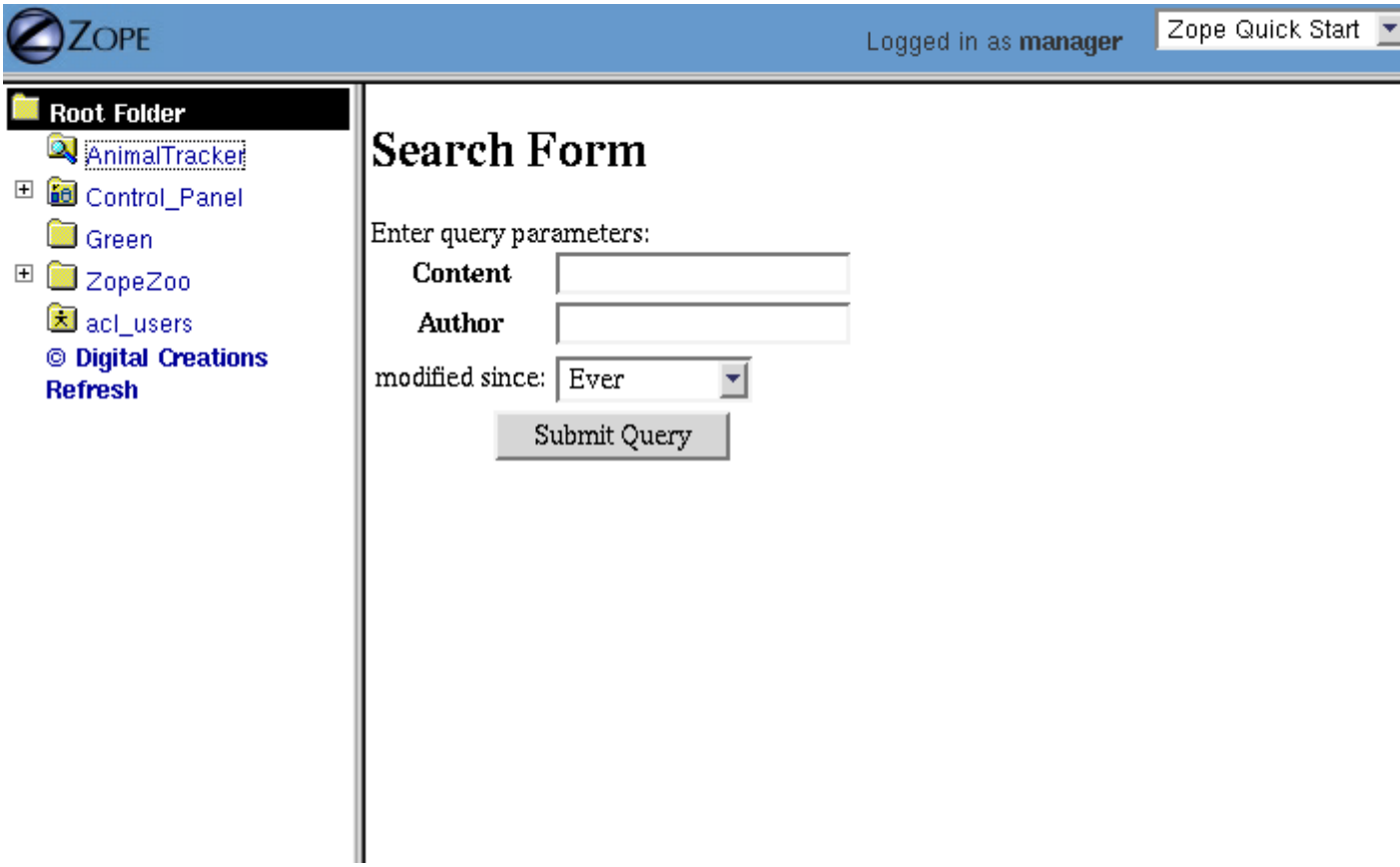


Abbildung 9-4 Suche nach einem Zeitraum

Dieses HTML-Formular ändert das *Datum*-Format aus dem alten Suchformular. Anstatt eines einfachen Textkastens bietet es einen Auswahlkasten, mit dem Sie das Datum auswählen können. Aber Achtung: es handelt sich um eine *Zeitraum*-Suche. Können Sie die Stelle finden, mit der dem *Datum*-Feld-Index gesagt wird, nach einen Bereich zu suchen? Hier ist sie:

```
<input type="hidden" name="Datum_usage" value="range:min">
```

Hierbei handelt es sich um ein spezielles HTML-Formularelement, das *verstecktes* Element genannt wird. Es erscheint nicht auf dem Formular, wird aber trotzdem an Zope gesandt, wenn Sie die das Formular abschicken. Dieses spezielle Element - hier *Datum_usage* genannt - informiert den Feld-Index, dass der Wert im *Datum*-Formularelement als *untere Bereichsgrenze* zu verwenden ist. D.h., der Feld-Index gibt nicht nur die Objekte zurück, die genau diese Zeitangabe haben, sondern auch diejenigen Objekte mit *jedem danach liegenden Datum*.

Für jede Feldindex-Art gibt es Begriffe für die Bereichsnutzung. Hierfür wird ein zusätzliches Suchargument benutzt, bei dem der Name des Indizes mit der Endung "_usage" ergänzt wird. Anstelle der Untergrenze können Sie auch eine *Obergrenze* bestimmen; das versteckte Formularelement ist dann wie folgt zu ändern:

```
<input type="hidden" name="Datum_usage" value="range:max">
```

Diese Anweisung veranlasst das Suchformular, alle Nachrichten *vor* einem bestimmten Zeitpunkt statt nach ihm bereitzustellen.

Die "_usage"-Syntax kann auch benutzt werden, wenn der Katalog von einem Skript wie hier von *relevanteKuerzlicheSektionsNachrichten* aufgerufen wird:

```
## Script (Python) "relevanteKuerzlicheSektionsNachrichten"
##
""" Liefert relevante, kürzlich hinzugefügt Nachrichten """
id=context.getId()
return context.NewsCatalog(
         {'Inhalt'    : id,
          'Datum'       : ZopeTime() - 7,
          'Datum_usage' : 'range:min',
         }
        )
```

Dieses Skript arbeitet wie das alte *relevanteSektionsNachrichten* Skript, allerdings zeigt es nur die in der letzten Woche hinzugefügten Nachrichten.

Sie können gleichzeitig die untere und obere Bereichsgrenze zur Suche verwenden. Hierbei ist aber etwas zu beachten. Normalerweise benutzt ZCatalog den übergebenen Wert als Suchbegriff, wenn überhaupt kein Bereich oder wenn nur ein Grenzwert angegeben ist. Wenn Sie *zwei* Bereichsgrenzen angeben, benötigt der ZCatalog auch zwei Suchwerte, nicht nur einen. Hier ist das *relevanteKuerzlicheSektionsNachrichten* Skript von oben mit einer geringfügigen Änderung, die eine *Liste* von Datumsobjekten statt eines einzelnen Datumswertes in das Skript einbaut:

```
## Script (Python) "relevanteKuerzlicheSektionsNachrichten"
##
"""
Liefert relevante Nachrichten, die im letzten Monat,
aber nicht in der letzten Woche bearbeitet worden sind.
"""
id=context.getId()
return context.NewsCatalog(
         {'Inhalt'    : id,
          'Datum'       : [ZopeTime() - 30, ZopeTime() - 7],
          'Datum_usage' : 'range:min:max',
         }
        )
```

Dieses Skript gibt alle relevanten Nachrichten zurück, die im letzten Monat, aber *nicht* in der letzten Woche, geändert worden sind. Wenn zwei Grenzwerte benutzt werden, ist es wichtig, darauf zu achten, dass die Reihenfolge der Werte mit der Reihenfolge der Bereichsbegriffe übereinstimmt. Wenn Sie zufällig die Begriffe "min" und "max" vertauschen, aber nicht gleichzeitig die beiden Zeitangaben, dann werden Sie *keine* Suchergebnisse erhalten, da Sie

eine Abfrage gestellt haben, die keinen Sinn macht (durch Lieferung eines Minimalwertes, der größer ist als der angegebene Maximalwert).

Der zweite Anwendungsfall, den wir oben beschrieben, betraf die Suche anhand einer Liste der im Katalog enthaltenen Autoren. Die hierfür in ZCatalog vorgesehene spezielle Methode heißt *uniqueValuesFor*. Die Methode uniqueValuesFor gibt eine Liste von Werten eines bestimmten Indizes zurück, wobei jeder Wert nur einmal in der Liste vorkommt, auch wenn hierzu mehrere Einträge im Katalog vorhanden sind. Lassen Sie uns das Suchformular nochmals ändern, und ersetzen wir den originalen Eingabekasten *Autor* mit etwas Nützlicherem:

```
        <dtml-var standard_html_header>

        <form action="Report" method="get">
        <h2><dtml-var document_title></h2>
        Suche nach Nachrichten:<br><table>

        <tr><th>Inhalt:</th>
            <td><input name="Inhalt" width=30 value=""></td></tr>
        <tr valign="top">
           <td><p>Autor:</p></td>

           <td>
             <select name="Autor:list" size=6 MULTIPLE>
             <dtml-in expr="AnimalTracker.uniqueValuesFor('Autor')">
               <option value="<dtml-var sequence-item>">
               <dtml-var sequence-item></option>
             </dtml-in>
             </select>
           </td>
         </tr>

        <tr>
          <td><p>Geändert seit:</p></td>
          <td>
            <input type="hidden" name="Datum_usage" value="range:min">
            <select name="Datum:date">
              <option value="<dtml-var "ZopeTime(0)" >">irgendwann</option>
              <option value="<dtml-var "ZopeTime() - 1" >">gestern</option>
              <option value="<dtml-var "ZopeTime() - 7" >">letzter
Woche</option>
              <option value="<dtml-var "ZopeTime() - 30" >">letztem
Monat</option>
              <option value="<dtml-var "ZopeTime() - 365" >">letztem
Jahr</option>
            </select>
          </td>
        </tr>

        <tr><td colspan="2" align="center">
        <input type="SUBMIT" name="SUBMIT" value="Abfrage senden">
        </td></tr>
        </table>
        </form>
        <dtml-var standard_html_footer>
```

Das neue wichtige Code-Stück, das dem Suchformular hinzugefügt worden ist, lautet:

```
<select name="Autor:list" size=6 MULTIPLE>
<dtml-in expr="AnimalTracker.uniqueValuesFor('Autor')">
  <option value="<dtml-var sequence-item>">
  <dtml-var sequence-item></option>
</dtml-in>
</select>
```

Weiter ist das HTML ein bisschen geändert worden, damit die Bildschirmanzeige Sinn macht.

In diesem Beispiel haben wir das Formularelement *Autor* von einem einfachen Textkasten in einen HTML-Mehrfach-Auswahl-Kasten geändert. Dieser Kasten enthält eine Liste mit den Autoren, die im *Autor*-Feldindex enthalten sind. Nun sieht das Suchformular wie in Abbildung 9-5 aus.

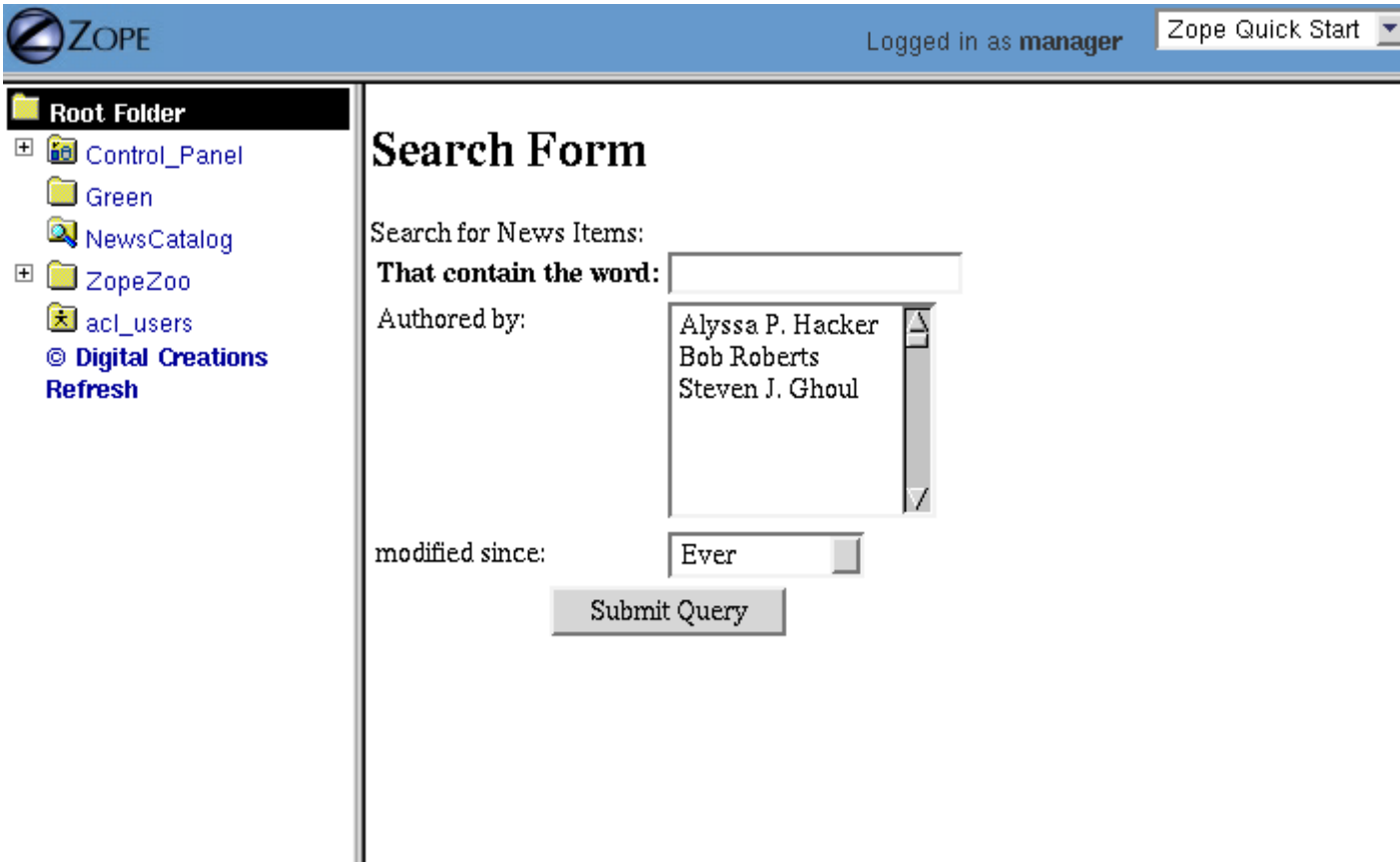


Abbildung 9-5 Bereichssuche und vorgegebene Autoren

Das war's. Sie können dieses Suchformular um weitere HTML-Formularelemente ergänzen und die Suche damit so komplex machen, wie Sie es möchten. Im nächsten Abschnitt werden wir zeigen, wie Sie die weitere Indexart, die Schlagwort-Indizes benutzen können.

Suchen mit Schlagwort-Indizes

Ein *Schlagwort-Index* nimmt eine Folge der Schlagwörter von Objekten auf und kann nach Objekten durchsucht werden, die ein Schlagwort oder mehrere Schlagwörter besitzen.

Nehmen wir an, Sie haben eine Reihe von Bild-(Image-)Objekten, denen eine Eigenschaft *Thema* zugeordnet ist. Die Eigenschaft *Themen* ist eine Zeileneigenschaft, die die relevanten Themen für ein bestimmtes Bild auflistet, z. B. für ein Bild der Königin Victoria "Portrait", "19. Jahrhundert" und "Frau".

Die Themen ermöglichen die Kategorisierung von Bildern. Jedes Bild gehört zu einer oder zu mehreren Kategorien, je nachdem, was in seiner Eigenschaft *Themen* eingetragen ist. So gehört das Bild der Königin Victoria in dem oben aufgeführten Beispiel zu drei Kategorien und kann daher gefunden werden, wenn nach einem dieser drei Begriffe gesucht wird.

Sie können einen *Schlagwort*-Index für die Suche nach der Eigenschaft *Themen* benutzen. Hierfür definieren Sie einen *Schlagwort*-Index mit dem Namen *Themen* in Ihrem ZCatalog. Dann katalogisieren Sie Ihre Bilder. Danach sollten Sie in der Lage sein, alle Portraits mit Hilfe eines Suchformulars und dem Eintrag "Portrait" im Feld *Themen* zu finden. Sie können auch alle Bilder im Zusammenhang mit dem 19. Jahrhundert finden, wenn Sie bei der Suche "19. Jahrhundert" eingeben.

Wichtig ist hierbei, dass dasselbe Bild mehr als einer Kategorie zugeordnet sein kann. Dies ermöglicht eine höhere Flexibilität für die Suche und die Kategorisierung Ihrer Objekte als beim Feld-Index. Ihr Portrait von Königin Victoria könnte in einem Feld-Index nur mit einer einzigen Art kategorisiert werden. Mit einem Schlagwort-Index kann es auf verschiedene Arten kategorisiert werden.

Häufig werden Sie nur eine kleine Liste mit Schlagwörtern für einen *Schlagwort*-Index haben. In diesem Fall werden Sie es vorziehen, mit der *uniqueValuesFor*-Methode ein angepasstes Suchformular zu verwenden. Zum Beispiel ist hier ein Ausschnitt mit DTML-Anweisungen, mit denen ein Mehrfach-Auswahl-Kasten für alle Werte aus dem *Thema*-Index erzeugt wird:

```
        <select name="Themen:list" multiple>
        <dtml-in expr="uniqueValuesFor('Themen')">
          <option value="&dtml-sequence-item;"><dtml-var sequence-
item></option>
        </dtml-in>
        </select>
```

Mit diesem Suchformular werden dem Benutzer eine Reihe gültiger Suchbegriffe vorgegeben. Sie können mehrere Themen auswählen, und Zope wird alle Bilder finden, die zu einem oder mehreren Themen gehören, die Sie ausgewählt haben. Nicht nur jedes Objekt kann mehreren verschiedenen indexierten Themen zugeordnet sein, sondern Sie können auch mehrere Themen als Suchbegriff angeben und damit alle Objekte finden, denen eine oder mehrere dieser Angaben zugeordnet sind.

Suchen in Pfad-Indizes

Pfad-Indizes ermöglichen die Suche nach Objekten aufgrund ihrer Anordnung in Zope. Angenommen, sie haben ein Objekt mit dem Pfad `/Zoo/Tiere/Afrika/Tiger.doc`. Dann können Sie dieses Objekt mit den Pfad-Abfragen `/Zoo` oder `/Zoo/Tiere`oder `/Zoo/Tiere/Afrika` finden. Mit anderen Worten: der Pfad-Index ermöglicht das Finden von Objekten in einem angegebenen Ordner (und darunter).

Wenn Sie zusammengehörende Objekte in demselben Ordner anordnen, können Sie die Pfad-Indizes benutzen, um diese Objekte schnell aufzusuchen. Zum Beispiel:

```
<h2>Eidechsen-Bilder</h2>

<p>
<dtml-in expr="Catalog(meta_type='Image',
                       path='/Zoo/Tiere/Eidechse')">
<a href="&dtml-absolute_url;"><dtml-var title></a>
</dtml-in>
</p>
```

Diese Abfrage sucht einen Katalog nach allen Bildern durch, die im `/Zoo/Tiere/Eidechse`-Ordner und darunter vorhanden sind. Sie erzeugt für jedes Bild eine Verbindung.

Abhängig davon, wie Sie Ihre Objekte in Ihrer Site anordnen, können die Pfad-Indizes mehr oder weniger nützlich sein. Wenn Sie die Objekte unabhängig von ihrem Thema plazieren (z. B., Sie legen sie in der Regel in persönlichen "home"-Ordnern an), sind Pfad-Indizes nur von begrenztem Wert. In diesem Fall sind Schlagwort- und Feld-Indizes nützlicher.

Advanced Searching with Records

A new feature in Zope 2.4 is the ability to query indexes more precisely using record objects. Record objects contain information about how to query an index. Records are Python objects with attributes, or mappings. Different indexes support different record attributes.

Keyword Index Record Attributes

`query`
: Either a sequence of words or a single word. (mandatory)

`operator`
: Specifies whether all keywords or only one need to match. Allowed values: `and`, `or`. (optional, default: 'or')

For example:

```
# big or shiny
results=Catalog(categories=['big, 'shiny'])

# big and shiny
results=Catalog(categories={'query':['big','shiny'],
                                     'operator':'and'})
```

The second query matches objects that have both the keywords "big" and "shiny". Without using the record syntax you can only match objects that are big or shiny.

Field Index Record Attributes

`query`

Either a sequence of objects or a single value to be passed as query to the index (mandatory)

`range`

Defines a range search on a Field Index (optional, default: not set).

Allowed values:

`min`
Searches for all objects with values larger than the minimum of the values passed in the `query` parameter.
`max`
Searches for all objects with values smaller than the maximum of the values passed in the `query` parameter.
`minmax`
Searches for all objects with values smaller than the maximum of the values passed in the `query` parameter and larger than the minimum of the values passwd in the `query` parameter.

For example:

```
# items modified in the last week
results=Catalog(bobobase_modification_time={
                  'query':DateTime() - 7,
                  'range': 'min'}
                )
```

This query matches objects with a `bobobase_modification_time` of less than `DateTime() -7`. Compare this query with one defined in `relevantRecentSectionNews` earlier in this chapter which uses `date_usage` to accomplish the same query.

Text Index Record Attributes

`query`

Either a sequence of words (seperated by white space) or a single word to be passed as query to the index. (mandatory)

`operator`

Specifies how to combine the search terms. (optional, default: 'or').

Allowed values:

`and`
All terms must be present.
`or`
At least one term must be present.
`andnot`
The first term must be present, but none of the rest of the terms.

There's not much reason to use record queries with text indexes since you can embed the operator information in the query string itself in a very flexible manner.

Path Index Record Attributes

`query`
: Path to search for either as a string (e.g. "/Zoo/Birds") or list (e.g. ["Zoo", "Birds"]). (mandatory)

`level`
: The path level to begin searching at. (optional, default: '0')

Suppose you have a collection of objects with these paths:

1. /aa/bb/aa
2. /aa/bb/bb
3. /aa/bb/cc
4. /bb/bb/aa
5. /bb/bb/bb
6. /bb/bb/cc
7. /cc/bb/aa
8. /cc/bb/bb
9. /cc/bb/cc

Here are some examples queries and their results to show how the `level` attribute works:

- query='/aa/bb', level=0 returns 1, 2, 3
- query='/bb/bb', level=0 returns 4, 5, 6
- query='/bb/bb', level=1 returns 2, 5, 8
- query='/bb/bb', level=-1 returns 2, 4, 5, 6, 8
- query='/xx', level=-1 returns none

You can use the level attribute to flexibly search different parts of the path.

As of Zope 2.4.1, you can also include level information in a search without using a record. Simply use a tuple containing the query and the level. Here's an example tuple: `("/aa/bb", 1)`.

Creating Records in HTML

You can also perform record queries using HTML forms. Here's an example showing how to create a search form using records:

```
        <form action="Report" method="get">
        <table>
        <tr><th>Search Terms (must match all terms)</th>
            <td><input name="content.query:record" width=30
value=""></td></tr>
            <input type="hidden" name="content.operator:record"
value="and">
        <tr><td colspan=2 align=center>
        <input type="SUBMIT" value="Submit Query">
        </td></tr>
        </table>
        </form>
```

For more information on creating records in HTML see the section "Passing Parameters to Scripts" in Chapter 10, Advanced Zope Scripting.

Stored Queries

While the main use of the Catalog is to provide interactive searching, you can also use *stored queries* to categorize and organize your site. For example, in the section on keyword indexes you saw how you can use the Catalog and properties to search for categories of Images such as portraits. In addition to providing interactive searching for categories of Images you can create web pages with canned queries. So for example, here's some DTML that you could use for a page that displays all your portraits:

```
<dtml-var standard_html_header>

<h1>Portraits</h1>

<dtml-in expr="ImageCatalog({'topics':'Portraits'})">
<p>
<dtml-var sequence-item>
<dtml-var title_or_id>
</p>
</dtml-in>

<dtml-var standard_html_footer>
```

The dynamic nature of this page is not visible to the viewer. However, just add another portrait, update the catalog and this page will automatically include the new Image.

This technique can be very powerful. Not only can you organize and display public resources, but you can easily institute workflow systems by tagging objects with properties to indicate their state and cataloging them. After that it's easy for you to create pages for different people that show which objects need their attention. This technique is even more powerful when using the *Automatic Cataloging* pattern.

Automatisches Katalogisieren

Automatisches Katalogisieren ist ein fortgeschrittenes Verwendungsmuster des ZCatalogs, der Objekte aktuell hält, falls sie geändert wurden. Wenn Objekte erstellt, geändert oder gelöscht werden, muss das automatisch von einem ZCatalog verfolgt werden. Dies setzt voraus, dass die Objekte den Katalog benachrichtigen, wenn sie erstellt, geändert oder gelöscht wurden.

Das hat eine Reihe von Vorteilen gegenüber der Massenkatalogisierung. Massenkatalogisierung ist einfach, hat aber Nachteile. Die Gesamtmenge von Inhalten, die Sie in einer Transaktion indexieren können, ist äquivalent zum Speicher, der für den Zope Prozess, plus dem temporärer Speicher zur Verfügung steht, den das System benötigt. Mit anderen Worten, je mehr sie in einem Zug indexieren wollen, desto leistungsfähiger muss Ihre Computerhardware sein. Massenkatalogisierung ist gut bis zu einigen tausend Objekten, aber darüber hinaus funktioniert die automatische Indexierung viel besser.

Ein anderer wichtiger Vorteil des automatischen Katalogisierens ist, dass es Objekte verwalten kann, die sich ändern. Wenn Objekte entstehen und sich ändern, ist die

Indexinformation immer aktuell, sogar für sich rasch ändernde Informationsquellen wie Nachrichtenbretter (message boards).

In diesem Abschnitt erläutern wir Ihnen ein Beispiel, in dem Nachrichten- (News) Elemente erstellt werden, die Autoren Ihrer Site hinzufügen können. Diese Elemente werden automatisch katalogisiert. Das Beispiel gliedert sich in zwei Teile:

- Erzeugung eines neuen Objekttyps für die Katalogisierung.
- Erzeugung eines Katalogs zur Katalogisierung der neu erstellten Objekte.

As of Zope 2.3, none of the "out-of-the-box" Zope objects support automatic cataloging. This is for backwards compatibility reasons. For now, you have to define your own kind of objects that can be cataloged automatically. One of the ways this can be done is by defining a *ZClass*.

A ZClass is a Zope object that defines new types of Zope objects. In a way, a ZClass is like a blueprint that describes how new Zope objects are built. Consider a news item as discussed in examples earlier in the chapter. News items not only have content, but they also have specific properties that make them news items. Often these Items come in collections that have their own properties. You want to build a News site that collects News Items, reviews them, and posts them online to a web site where readers can read them.

In this kind of system, you may want to create a new type of object called a *News Item*. This way, when you want to add a new news item to your site, you just select it from the product add list. If you design this object to be automatically cataloged, then you can search your news content very powerfully. In this example, you will just skim a little over ZClasses, which are described in much more detail in Chapter 14, "Extending Zope."

New types of objects are defined in the *Products* section of the Control Panel. This is reached by clicking on the Control Panel and then clicking on *Product Management*. Products contain new kinds of ZClasses. On this screen, click "Add" to add a New product. You will be taken to the Add form for new Products.

Name the new Product "News" and click "Generate". This will take you back to the Products Management view and you will see your new Product.

Select the *News* Product by clicking on it. This new Product looks a lot like a Folder. It contains one object called *Help* and has an Add menu, as well as the usual Folder "tabs" across the top. To add a new ZClass, pull down the Add menu and select *ZClass*. This will take you to the ZClass add form, as shown in Figure 9-6.

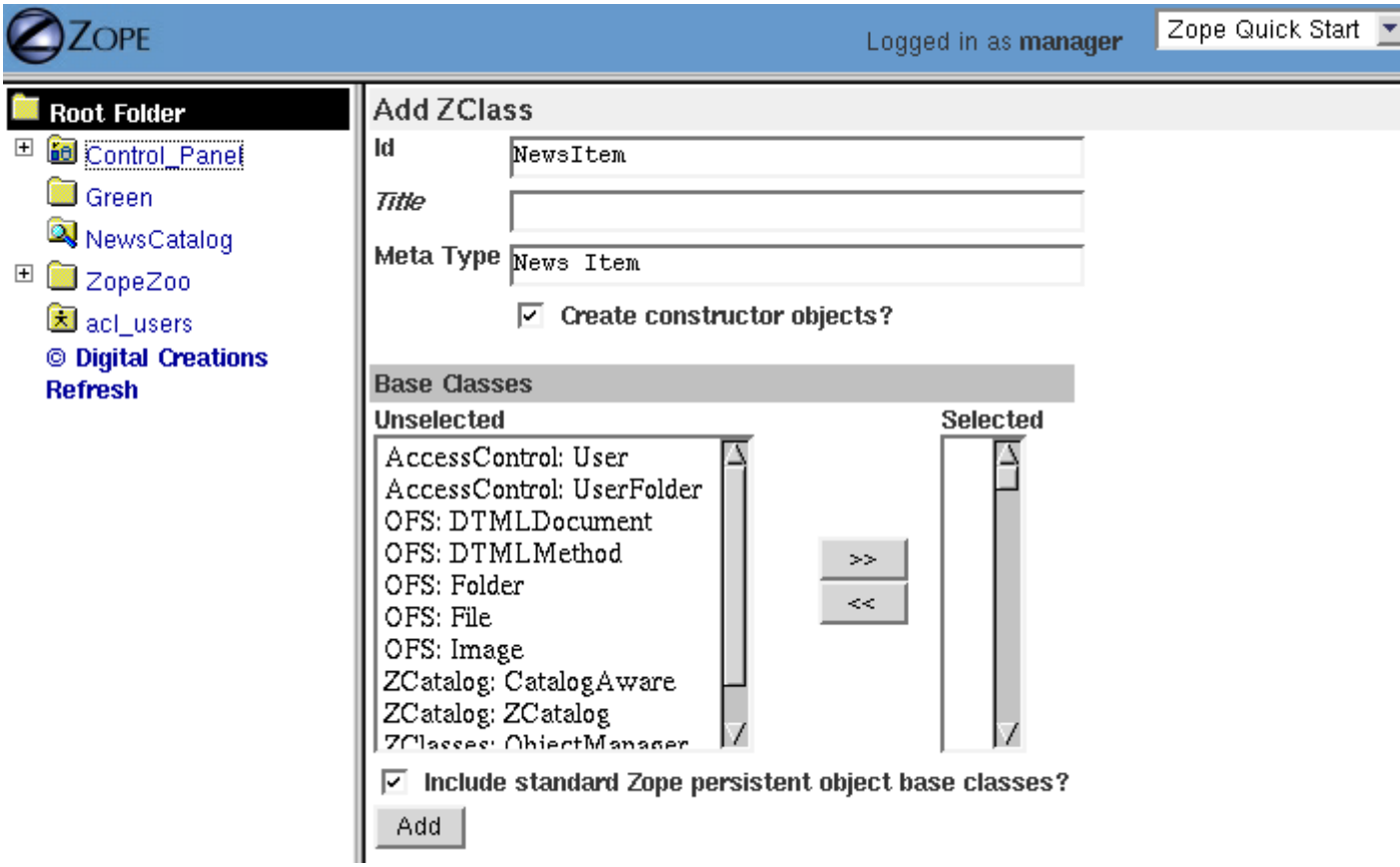


Figure 9-6 ZClass add form

This is a complicated form which will be explained in much more detail in Chapter 14, "Extending Zope". For now, you only need to do three things to create your ZClass:

- Specify the Id "NewsItem" This is the name of the new ZClass.
- Specify the meta_type "News Item". This will be used to create the Add menu entry for your new type of object.
- Select *ZCatalog:CatalogAware* from the left hand *Base Classes* box, and click the button with the arrow pointing to the right hand *Base Classes* box. This should cause *ZCatalog:CatalogAware* to show up in the right hand window.

When you're done, don't change any of the other settings in the Form. To create your new ZClass, click *Add*. This will take you back to your *News* Product. Notice that there is now a new object called *NewsItem* as well as several other objects. The *NewsItem* object is your new ZClass. The other objects are "helpers" that you will examine more in Chapter 14, "Extending Zope".

Select the *NewsItem* ZClass object. Your view should now look like Figure 9-7.

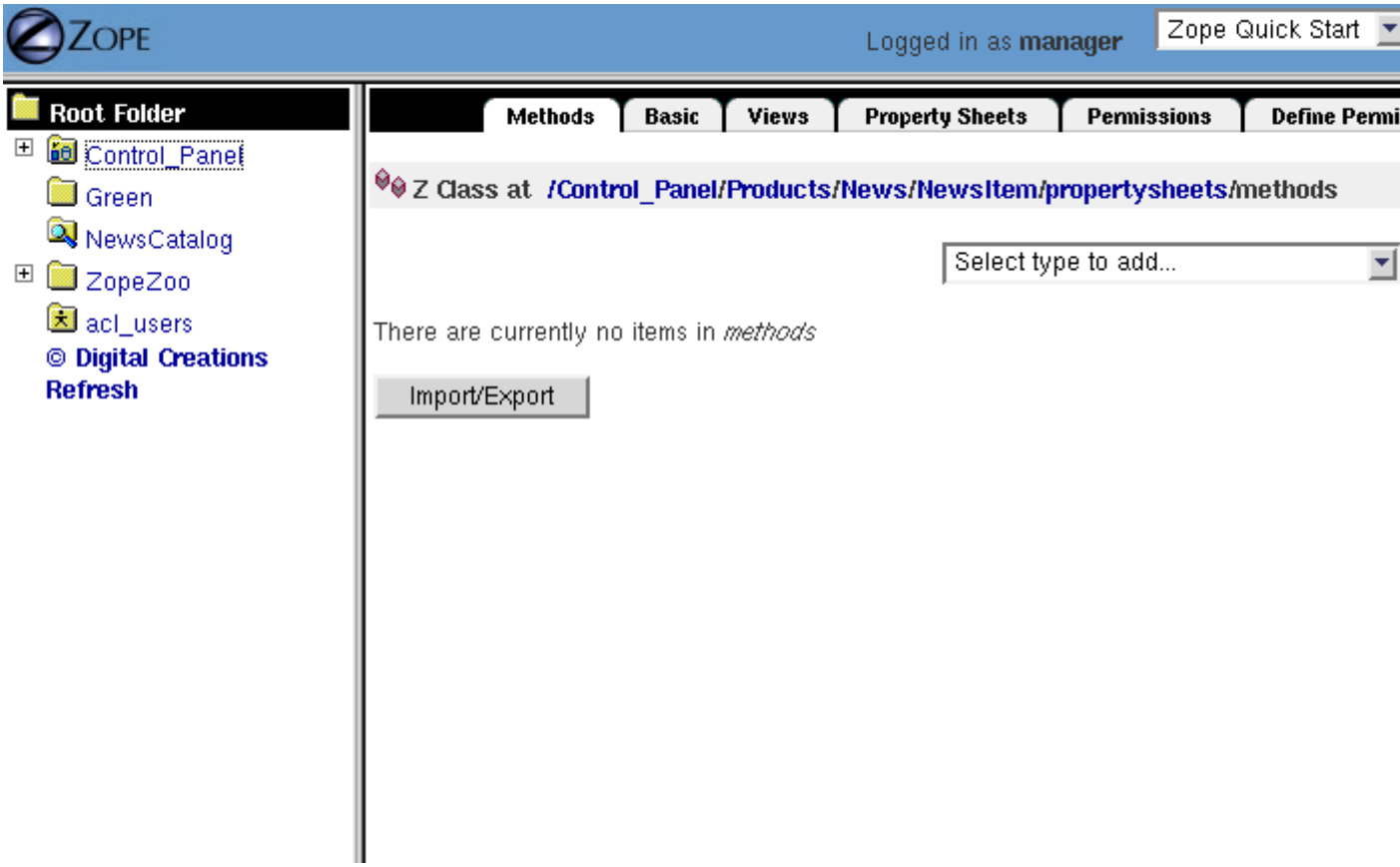


Figure 9-7 A ZClass Methods View

This is the *Methods* View of a ZClass. Here, you can add Zope objects that will act as *methods on your new type of object*. Here, for example, you can create DTML Methods or Scripts and these objects will become methods on any new *News Items* that are created. Before creating any methods however, let's review the needs of this new "News Item" object:

News Content
: The news Item contains news content, this is its primary purpose. This content should be any kind of plain text or marked up content like HTML or XML.

Author Credit
: The News Item should provide some kind of credit to the author or organization that created it.

Date
: News Items are timely, so the date that the item was created is important.

Keywords
: News Items fit into various lists of categories. By convention, these lists of categories are often called *keywords*.

You may want your new News Item object to have other properties, these are just suggestions. To add new properties to your News Item click on the *Property Sheets* tab. This takes you to the *Property Sheets* view.

Properties are added to new types of objects in groups called *Property Sheets*. Since your object has no property sheets defined, this view is empty. To add a New Property Sheet, click *Add Common Instance Property Sheet*, and give the sheet the name "News". Now click *Add*. This will add a new Property Sheet called *News* to your object. Clicking on the new Property Sheet will take you to the *Properties* view of the *News* Property Sheet, as shown in Figure 9-8.

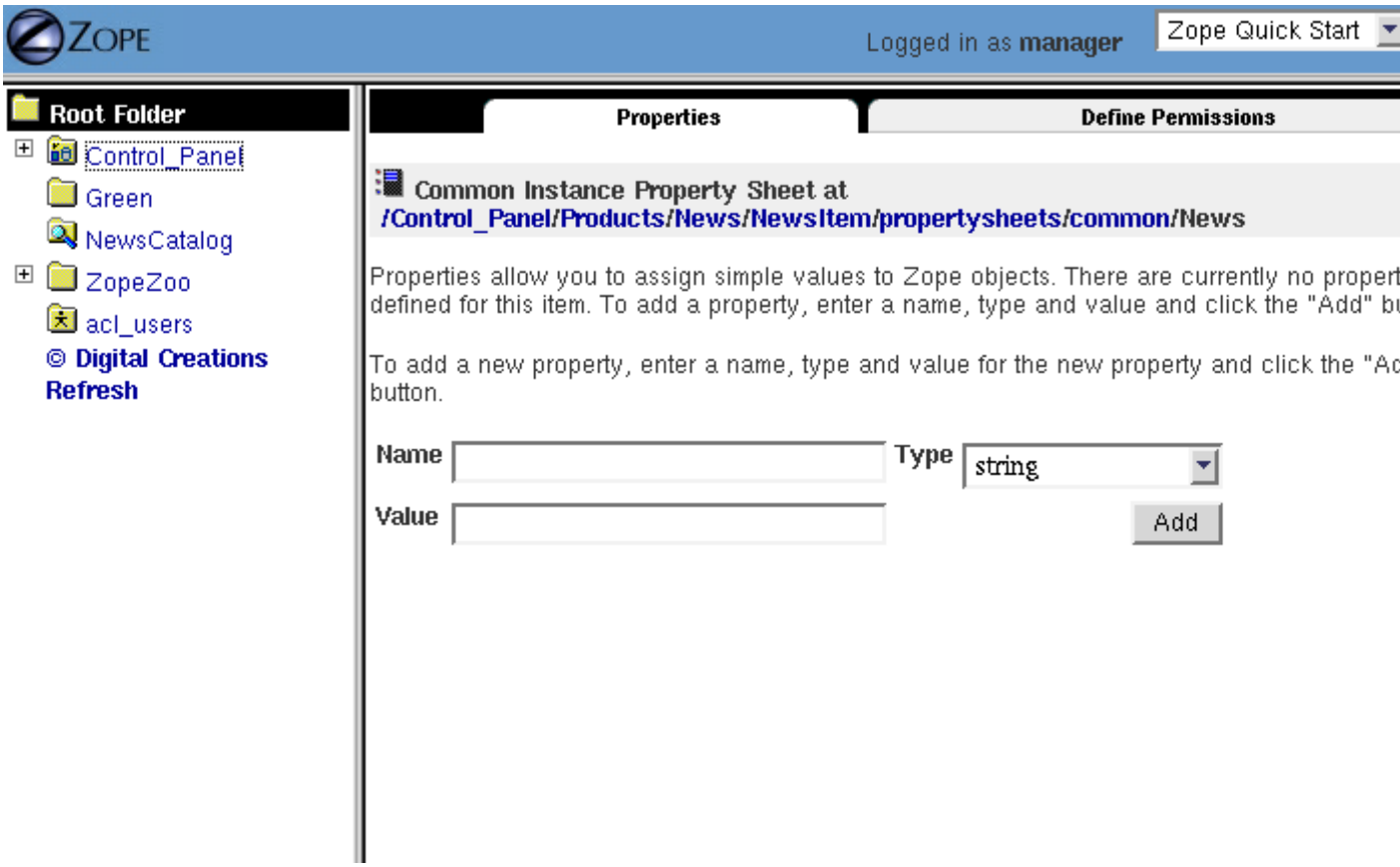


Figure 9-8 The properties screen for a Property Sheet

This view is almost identical to the *Properties* view found on Folders and other objects. Here, you can create the properties of your News Item object. Create three new properties in this form:

content
: This property's type should be *text*. Each newly created News Item will contain its own unique content property.

author
: This property's type should be *string*. This will contain the name of the news author.

date
: This property's type should be *date*. This will contain the time and date the news item was last updated. A *date* property requires a value, so for now you can enter the string "01/01/2000".

That's it! Now you have created a Property Sheet that describes your News Items and what kind of information they contain. Properties can be thought of as the *data* that an object contains. Now that we have the data all set, you need to create an *interface* to your new kind of objects. This is done by creating new *Views* for your object.

Click on the *Views* tab. This will take you to the *Views* view, as shown in Figure 9-9.

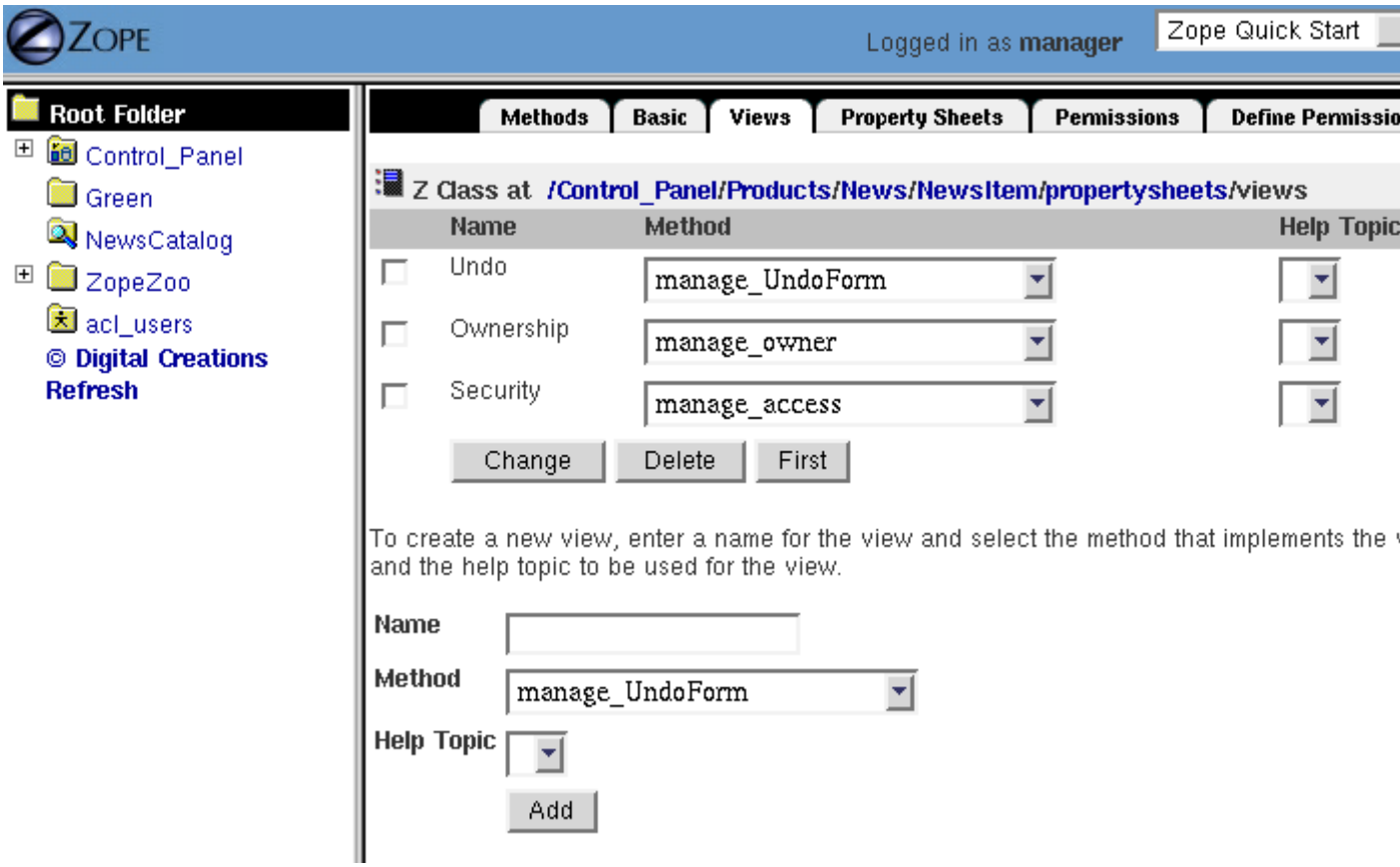


Figure 9-9 The Views view

Here, you can see that Zope has created three default Views for you. These views will be described in much more detail in Chapter 14, "Extending Zope", but for now, it suffices to say that these views define the tabs that your objects will eventually have.

To create a new view, use the form at the bottom of the Views view. Create a new View with the name "News" and select "propertysheets/News/manage" from the select box and click *Add*. This will create a new View on this screen under the original three Views, as shown in Figure 9-10.

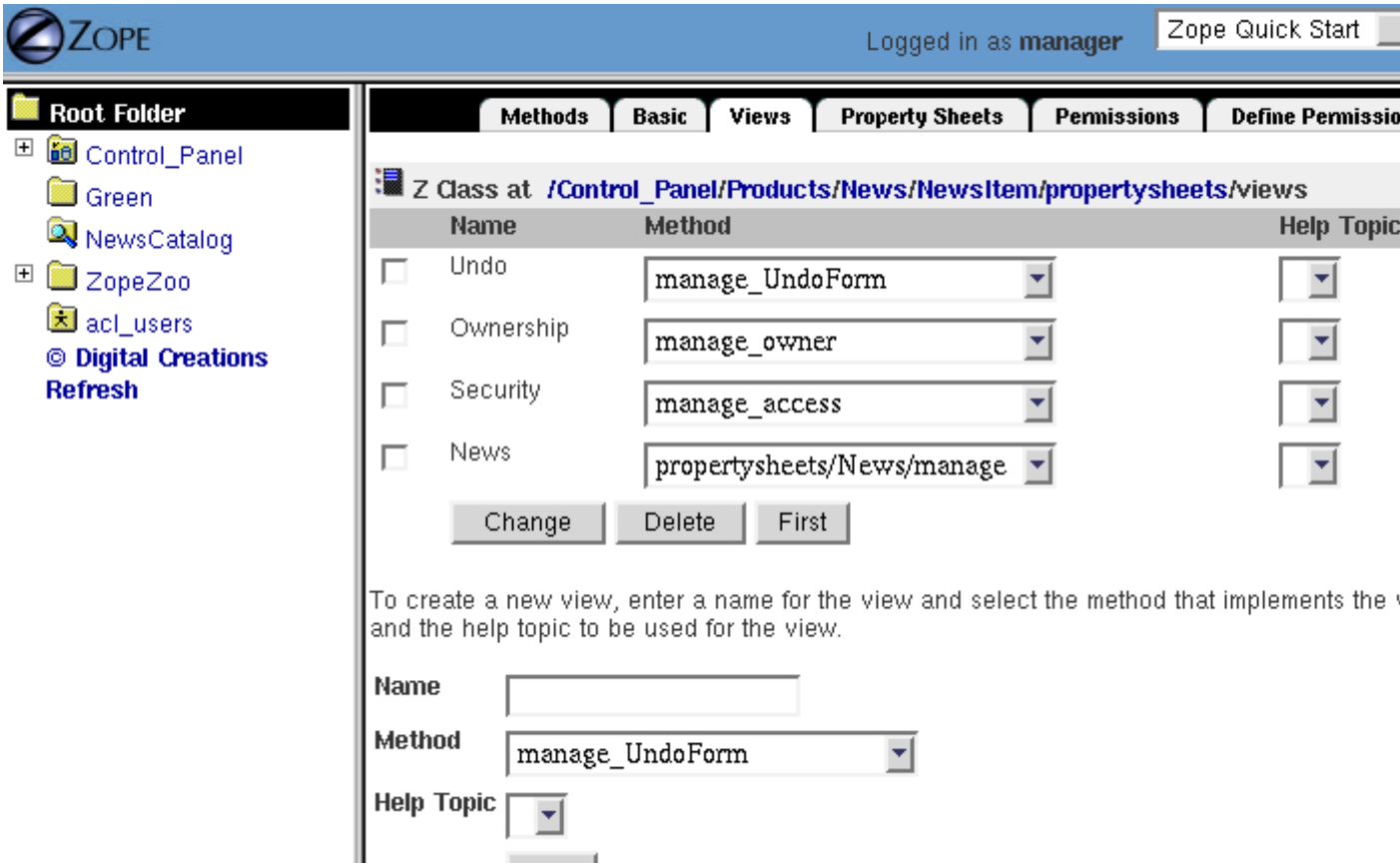


Figure 9-10 The new News View

Since this View is going to give us the ability to edit the News Item, we want to make it the first view that you see when you select a News Item object. To change the order of the views, select the newly created *News* view and click the *First* button. This should move the new view from the bottom to the top of the list.

The final step in creating a ZClass is defining the methods for the class. Methods are defined on the *Methods* View. Click on the *Methods* tab and you will be taken to the Methods view. Select 'DTML Method' from the add list and add a new DTML Method with the id "index_html". This will be the default view of your news item. Add the following DTML to the new method:

```
<dtml-var standard_html_header>

<h1>News Flash</h1>

<p><dtml-var date></p>

<p><dtml-var author></p>

<P><dtml-var content></p>

<dtml-var standard_html_footer>
```

That's it! You've created your own kind of object called a *News Item*. When you go to the root folder, you will now see a new entry in your add list.

But don't add any new News Items yet, because the second step in this exercise is to create a Catalog that will catalog your new News Items. Go to the root folder and create a new catalog with the id *Catalog*.

Like the previous two examples of using a ZCatalog, you need to create Indexes and a Meta-Data Table that make sense for your objects. First, delete the default indexes in the new ZCatalog and create the following indexes to replace them:

content
: This should be a TextIndex. This will index the content of your News Items.

title
: This should be a TextIndex. This will index the title of your News Items.

author
: This should be a FieldIndex. This will index the author of the News Item.

date
: This should be a FieldIndex. This will index the date of the News Item.

After creating these Indexes, delete the default Meta-Data columns and add these columns to replace them:

- author
- date
- title
- absolute_url

After creating the Indexes and Meta-Data Table columns, create a search interface for the Catalog using the Z Search Interface tool described previously in this chapter.

Now you are ready to go. Start by adding some new News Items to your Zope. Go anywhere in Zope and select *News Item* from the add list. This will take you to the add Form for News items.

Give your new News Item the id "KoalaGivesBirth" and click *Add*. This will create a new News Item. Select the new News Item.

Notice how it has four tabs that match the four Views that were in the ZClass. The first View is *News*, this view corresponds to the *News* Property Sheet you created in the News Item ZClass.

Enter your news in the *contents* box:

```
Today, Bob the Koala bear gave birth to little baby Jimbo.
```

Enter your name in the *Author* box, and today's date in the *Date* box.

Click *Change* and your News Item should now contain some news. Because the News Item object is *CatalogAware*, it is automatically cataloged when it is changed or added. Verify this by looking at the *Cataloged Objects* tab of the ZCatalog you created for this example.

The News Item you added is the only object that is cataloged. As you add more News Items to your site, they will automatically get cataloged here. Add a few more items, and then experiment with searching the ZCatalog. For example, if you search for "Koala" you should get back the *KoalaGivesBirth* News Item.

At this point you may want to use some of the more advanced search forms that you created earlier in the chapter. You can see for example that as you add new News Items with new authors, the authors select list on the search form changes to include the new information.

Conclusion

The cataloging features of ZCatalog allow you to search your objects for certain attributes very quickly. This can be very useful for sites with lots of content that many people need to be able to search in an efficient manner.

Searching the ZCatalog works a lot like searching a relational database, except that the searching is more object-oriented. Not all data models are object-oriented however, so in some cases you will want to use the ZCatalog, but in other cases you may want to use a relational database. The next chapter goes into more details about how Zope works with relational databases, and how you can use relational data as objects in Zope.

Zopebuch: Inhaltsverzeichnis

Dieses Dokument wird momentan übersetzt und ist zur Zeit leider nur in englisch verfügbar.

Chapter 12: Relational Database Connectivity

Zope uses an object database to store Zope objects. Relational databases such as Oracle, Sybase and PostgreSQL use a different store information in a different way. Relational databases store their information in tables as shown in Figure 10-1.

ROW	FIRST NAME	LAST NAME	AGE
#1	Bob	McBob	42
#2	John	Johnson	24
#3	Steve	Smith	38

Figure 10-1 Relational Database Table

Information in the table is stored in rows. The table's column layout is called the *schema*. A standard language, called the Structured Query Language (SQL) is used to query and change tables in relational databases.

Zope does not store its information this way. Zope's object database allows for many different types of objects that have many different types of relationships to each other. Relational data does not easily map onto objects since relational data assumes a much simpler table-oriented data model. Zope provides several mechanisms for taking relational data and using it in Zope's object-centric world including Database Adapters and SQL Methods which we will discuss in detail in this chapter.

The most common use for Zope's relational database support is to put existing relational databases on the web. For example, suppose your Human Resources Department has an employee database. Your database comes with tools to allow administrators run reports and change data. However, it is hard for employees to see their own records and perform simple maintenance such as updating their address when they move. By interfacing your relational

database with Zope, your employees can use any web browser to view and update their records from the office or at home.

By using your relational data with Zope you get all of Zope's benefits including security, dynamic presentation, networking, and more. You can use Zope to dynamically tailor your data access, data presentation, and data management.

To use a relational database in Zope you must create two different Zope objects, a Database Connection and a Z SQL Method. Database Connections tell Zope how to connect to a relational database. Z SQL Methods describe an action to take on a database. Z SQL Methods use Database Connections to connect to relational databases. We'll look more closely at these two types of objects in this chapter.

Using Database Connections

Database Connections are used to establish and manage connections to external relational databases. Database Connections must be established before database methods can be defined. Moreover, every Z SQL Method must be associated with a database connection. Database adapters (or DAs for short) are available for a number of databases:

Oracle
: Oracle is a powerful and popular commercial relational database. This DA is written and commercially supported by Zope Corporation. Oracle can be purchased or evaluated from the Oracle Website.

Sybase
: Sybase is another popular commercial relational database. The Sybase DA is written and commercially supported by Zope Corporation. Sybase can be purchased or evaluated from the Sybase Website.

ODBC
: ODBC is a cross-platform, industry standard database protocol supported by many commercial and open source databases. The ODBC DA is written and commercially supported by Zope Corporation.

PostgreSQL
: PostgreSQL is a leading open source relational database. There are several database adapters for PostgreSQL including ZPoPy which is maintained by Zope community member Thierry Michel. You can find more information about PostgreSQL at the PostgreSQL web site.

MySQL
: MySQL is a fast open source relational database. You can find more information about MySQL at the MySQL web site. The MySQL DA is maintained by Zope community member Monty Taylor.

Interbase
: Interbase is an open source relational database from Borland/Inprise. You can find more information about Interbase at the Borland web site. You may also be interested in FireBird which is a community maintained offshoot of Interbase. The Zope Interbase adapter is maintained by Zope community member Bob Tierney.

Gadfly
: Gadfly is a relational database written in Python by Aaron Waters. Gadfly is included with Zope for demonstration purposes and small data sets. Gadfly is fast, but is not intended for large amounts of information since it reads the entire database into memory. You can find out more about Gadfly at the Chordate website.

Other than Gadfly, all relational databases run as processes external to Zope. In fact, your relational database need not even run on the same machine as Zope, so long as Zope can connect to the machine that the database is running on. Installing and setting up relational databases is beyond the scope of this book. All of the relational databases mentioned have their own installation and configuration documentation that you should consult for specific details.

Because Gadfly runs inside Zope, you do not need to specify any connection information for Zope to find the database. Since all other kinds of databases run externally to Zope, they require you to specify how to connect to the database. This specification, called a *connection string*, is different for each kind of database. For example, Figure 10-2 shows the PostgreSQL database connection add form.

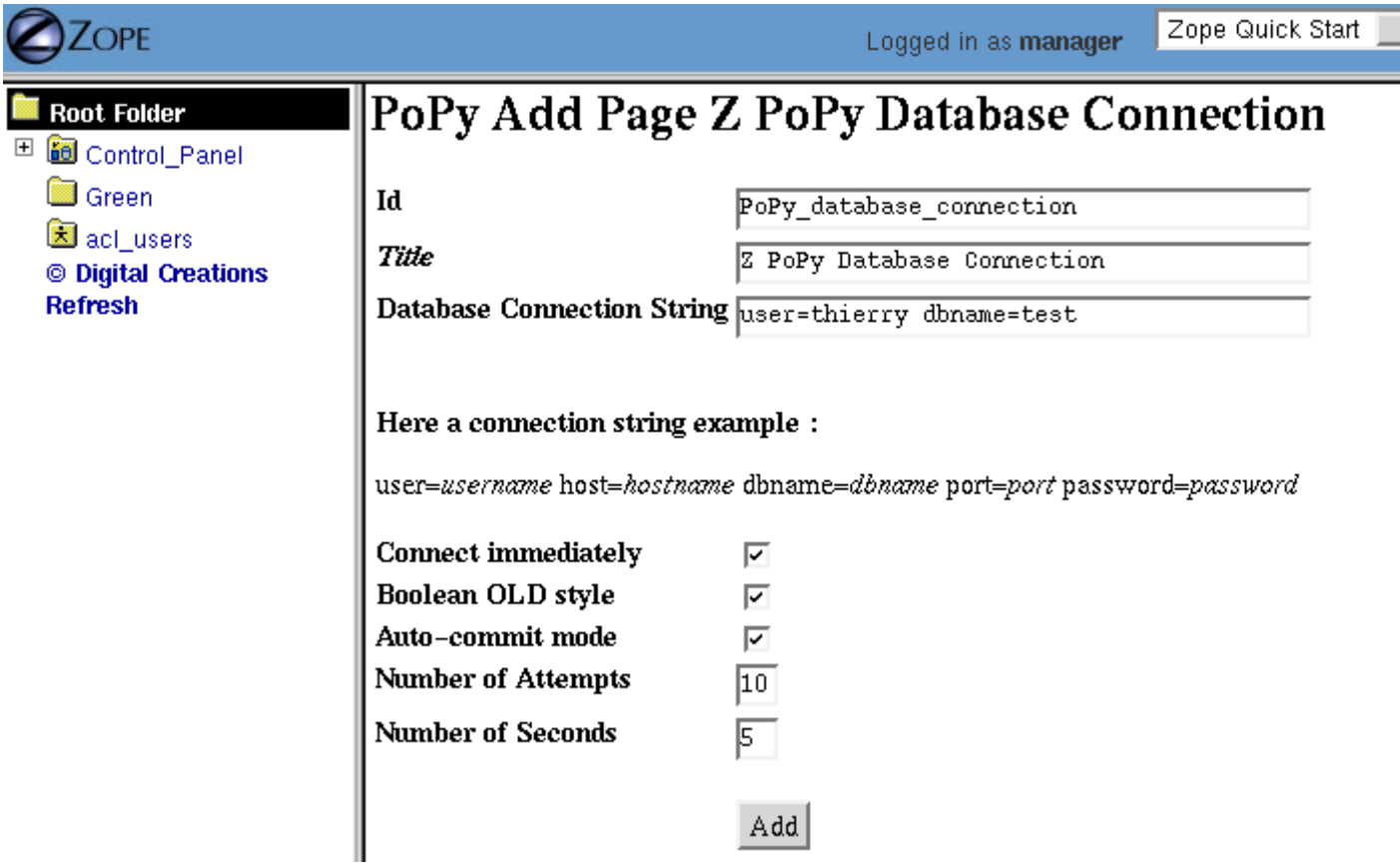


Figure 10-2 PostgreSQL Database Connection

For PostgreSQL, the connection string format is shown above in Figure 10-2.

In order to use your relational database of choice from Zope, you must download and install the database adapter for your specific relational database. Database adapters can be downloaded from the Products section of Zope.org The exception to this is Gadfly, which is included with Zope. All the examples in this chapter use Gadfly, but the procedures described apply to all databases.

After installing the database adapter product for your database, you can create a new database connection by selecting it from the *Add List*. All database connections are fairly similar. Select the *Z Gadfly Database Connection* from the add list. This will take you to the add form for a Gadfly database connection.

Select the *Demo* data source, specify *Gadfly_database_connection* for the id, and click the *Add* button. This will create a new Gadfly Database Connection. Select the new connection by clicking on it.

You are looking at the *Status* view of the Gadfly Database Connection. This view tells you if you are connected to the database, and there is a button to connect or disconnect. In general Zope will manage the connection to your database for you so there is little reason to manually control the connection. For Gadfly connecting and disconnecting are meaningless, but for external databases you may wish to connect or disconnect manually to do database maintenance.

The next view is the *Properties* view. This view shows you the data source and other properties of the Database Connection. This is useful if you want to move your Database Connection from one data source to another. Figure 10-3 shows the *Properties* view.

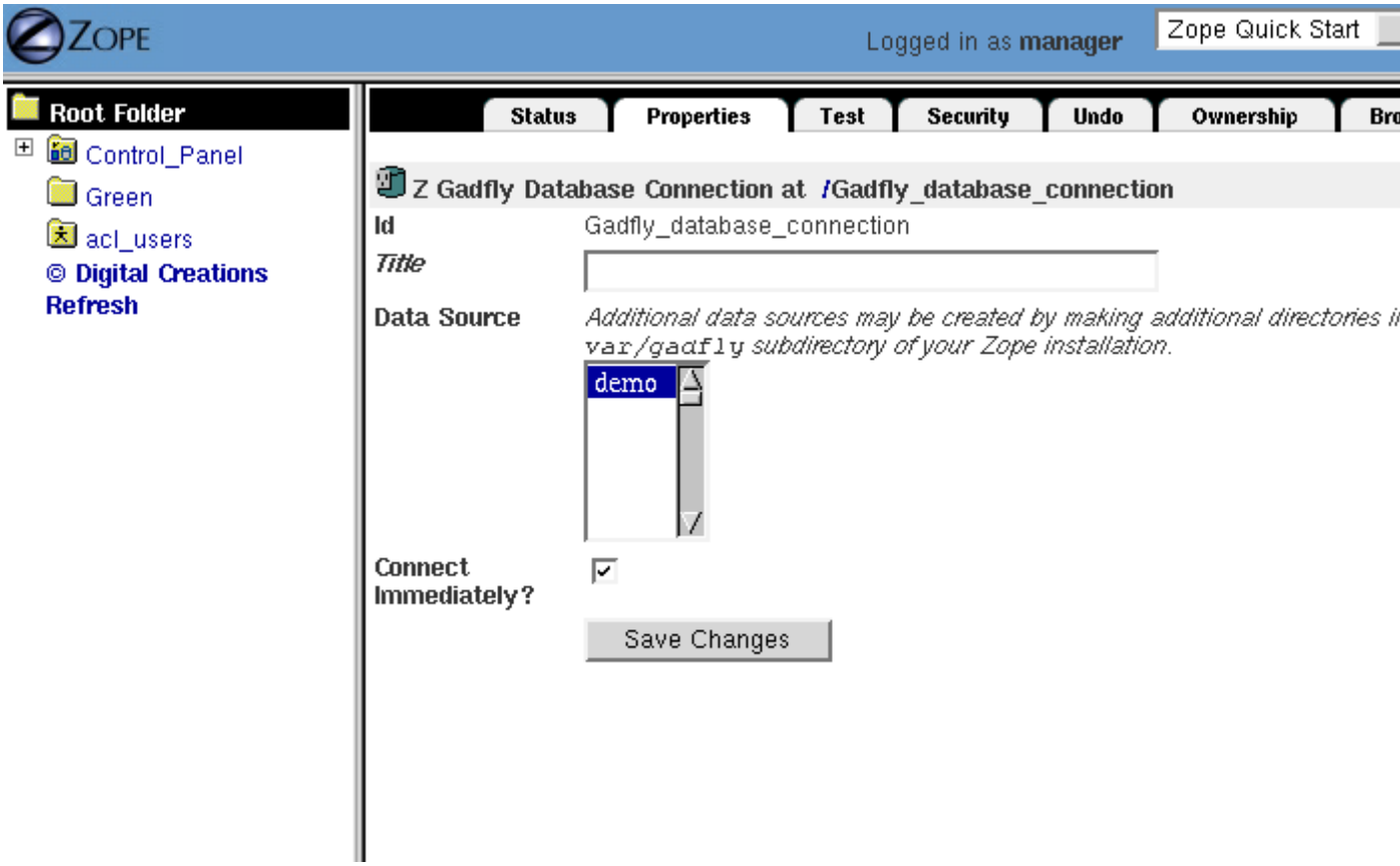


Figure 10-3 The Properties view

You can test your connection to a database by going to the *Test* view. This view lets you type SQL code directly and run it on your database. This view is just for testing your database and issuing one time SQL commands (like creating tables). This is *not* the place where you will

enter most of your SQL code. SQL commands reside in Z SQL Methods which are discussed later in this chapter.

Let's create a table in your database to use in this chapter's examples. The *Test* view of the Database Connection allows you to send SQL statements directly to your database. You can create tables by typing SQL code directly into the *Test* view; there is no need to use a SQL Method to create tables. Create a table called *employees* with the following SQL code:

```
CREATE TABLE employees
(
emp_id integer,
first varchar,
last varchar,
salary float
)
```

Click the *Submit Query* button to run the SQL command. Zope should return a confirmation screen that tells you what SQL code was run and the results if any.

The SQL used here may differ depending on your database. For the exact details of creating tables with your database, check the user documentation from your specific database vendor.

This SQL will create a new table in your Gadfly database called *employees*. This table will have four columns, *emp_id*, *first*, *last* and *salary*. The first column is the employee id, which is a unique number that identifies the employee. The next two columns have the type *varchar* which is similar to a string. The *salary* column has the type *float* which holds a floating point number. Every database supports different kinds of types, so consult your documentation to find out what kind of types your database supports.

To ensure that the employee id is a unique number you can create an index on your table. Type the following SQL code in the *Test* view:

```
CREATE UNIQUE INDEX emp_id ON employees
(
emp_id
)
```

Now you have a table and an index. To examine your table, go to the *Browse* view. This view lets you view your database's tables and their schemas. Here, you can see that there is an *employees* table, and if you click on the *plus symbol*, the table expands to show four columns, *emp_id*, *first*, *last* and *salary* as shown in Figure 10-4.

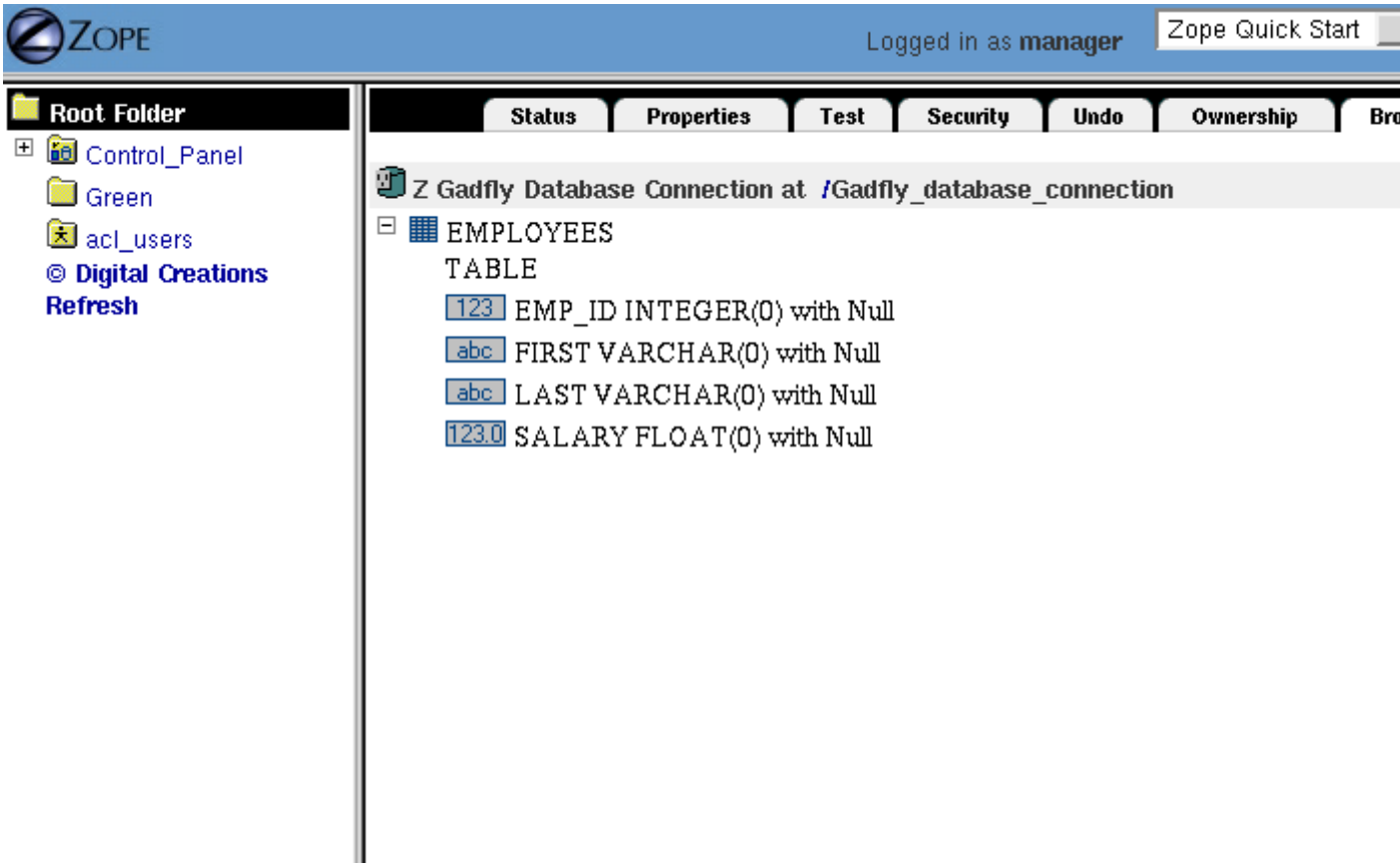


Figure 10-4 Browsing the Database Connection

This information is very useful when creating complex SQL applications with lots of large tables as it lets you discover the schemas of your tables. Not all databases support browsing of tables.

Now that you've created a database connection and have defined a table, you can create Z SQL Methods to operate on your database.

Using Z SQL Methods

Z SQL Methods are Zope object that execute SQL code through a Database Connection. All Z SQL Methods must be associated with a Database Connection. Z SQL Methods can both query databases and change data. Z SQL Methods can also contain more than one SQL command.

Next, you need to create a new Z SQL Method called *hire_employee* that inserts a new employee in the *employees* table. When a new employee is hired this method is called and a new record is inserted in the *employees* table that contains the information about the new employee. Select *Z SQL Method* from the *Add List*. This will take you to the add form for Z SQL Methods, as shown in Figure 10-5.

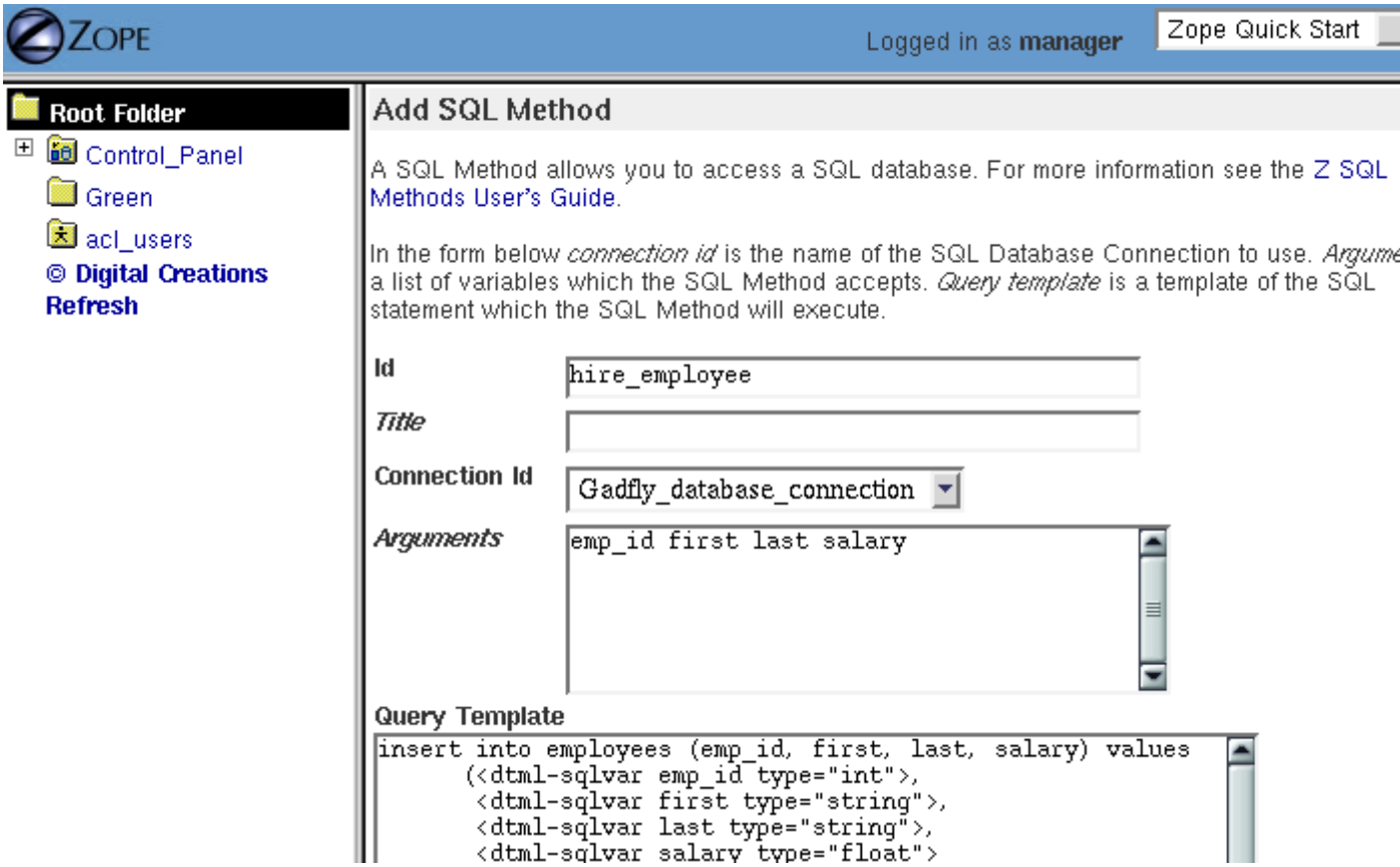


Figure 10-5 The Add form for Z SQL Methods

As usual, you must specify an *id* and *title* for the Z SQL Method. In addition you need to select a Database Connection to use with this Z SQL Methods. Give this new method the id *hire_employee* and select the *Gadfly_database_connection* that you created in the last section.

Next you can specify *arguments* to the Z SQL Method. Just like Scripts, Z SQL Methods can take arguments. Arguments are used to construct SQL statements. In this case your method needs four arguments, the employee id number, the first name, the last name and the employee's salary. Type "emp_id first last salary" into the *Arguments* field. You can put each argument on its own line, or you can put more than one argument on the same line separated by spaces. You can also provide default values for argument just like with Python Scripts. For example, `empid=100` gives the `empid` argument a default value of 100.

The last form field is the *Query template*. This field contains the SQL code that is executed when the Z SQL Method is called. In this field, enter the following code:

```
insert into employees (emp_id, first, last, salary) values
(<dtml-sqlvar emp_id type="int">,
 <dtml-sqlvar first type="string">,
 <dtml-sqlvar last type="string">,
 <dtml-sqlvar salary type="float">
)
```

Notice that this SQL code also contains DTML. The DTML code in this template is used to insert the values of the arguments into the SQL code that gets executed on your database. So, if the *emp_id* argument had the value *42*, the *first* argument had the value *Bob* your *last* argument had the value *Uncle* and the *salary* argument had the value *50000.00* then the query template would create the following SQL code:

```
insert into employees (emp_id, first, last, salary) values
(42,
 'Bob',
 'Uncle',
 50000.00
)
```

The query template and SQL-specific DTML tags are explained further in the next section.

You have your choice of three buttons to click to add your new Z SQL Method. The *Add* button will create the method and take you back to the folder containing the new method. The *Add and Edit* button will create the method and make it the currently selected object in the *Workspace*. The *Add and Test* button will create the method and take you to the method's *Test* view so you can test the new method. To add your new Z SQL Method, click the *Add* button.

Now you have a Z SQL Method that inserts new employees in the *employees* table. You'll need another Z SQL Method to query the table for employees. Create a new Z SQL Method with the id *list_all_employees*. It should have no arguments and contain the SQL code:

```
select * from employees
```

This simple SQL code selects all the rows from the *employees* table. Now you have two Z SQL Methods, one to insert new employees and one to view all of the employees in the database. Let's test your two new methods by inserting some new employees in the *employees* table and then listing them. To do this, click on the *hire_employee* Method and click the *Test* tab. This will take you to the *Test* view of the Method, as shown in Figure 10-6.

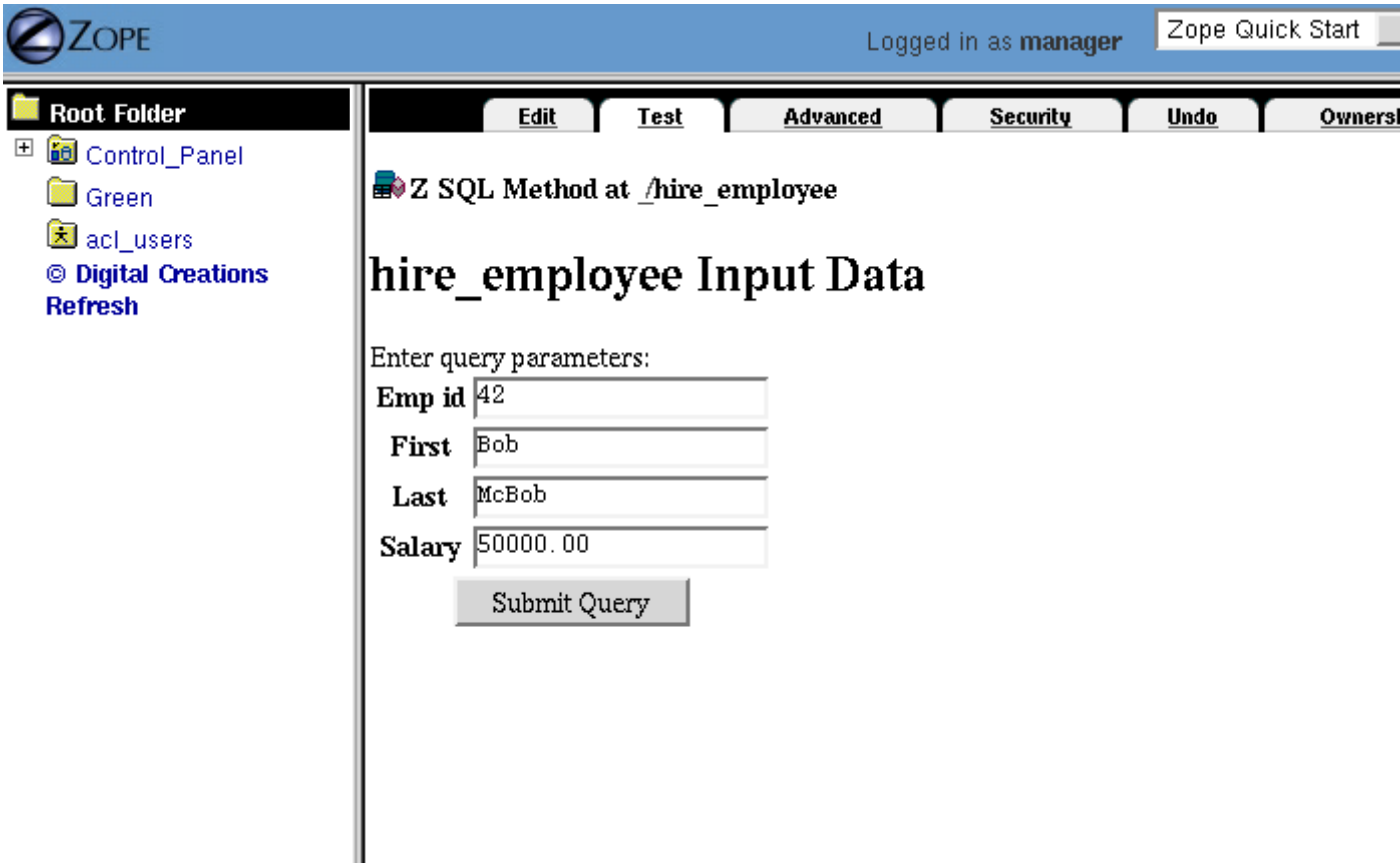


Figure 10-6 The hire_employee Test view

Here, you see a form with four input boxes, one for each argument to the *hire_employee* Z SQL Method. Zope automatically generates this form for you based on the arguments of your Z SQL Method. Because the *hire_employee* Method has four arguments, Zope creates this form with four input boxes. You can test the method by entering an employee number, a first name, a last name, and a salary for your new employee. Enter the employee id "42", "Bob" for the first name, "McBob" for the last name and a salary of "50000.00". Then click the *Test* button. You will then see the results of your test.

The screen says *This statement returned no results*. This is because the *hire_employee* method only inserts new information in the table, it does not select any information out of the table, so no records were returned. The screen also shows you how the query template get rendered into SQL. As expected, the *sqlvar* DTML tags rendered the four arguments into valid SQL code that your database executed. You can add as many employees as you'd like by repeatedly testing this method.

To verify that the information you added is being inserted into the table, select the *list_all_employees* Z SQL Method and click on its *Test* tab.

This view says *This query requires no input*, indicating the *list_all_employees* does not have any argument and thus, requires no input to execute. Click on the *Submit Query* button to test the method.

The *list_all_employees* method returns the contents of your *employees* table. You can see all the new employees that you added. Zope automatically generates this tabular report screen for you. Next we'll show how you can create your own user interface to your Z SQL Methods to integrate them into your web site.

Calling Z SQL Methods

Querying a relational database returns a sequence of results. The items in the sequence are called *result rows*. SQL query results are always a sequence. Even if the SQL query returns only one row, that row is the only item contained in a list of results. Hence, Z SQL Methods *always* return a sequence of results which contains zero or more results records.

The items in the sequence of results returned by a Z SQL Method are called *Result objects*. Result objects can be thought of as rows from the database table turned into Zope objects. These objects have attributes that match the schema of the database results.

An important difference between result objects and other Zope objects is that result objects do not get created and permanently added to Zope. Result objects are not persistent. They exist for only a short period of time; just long enough for you to use them in a result page or to use their data for some other purpose. As soon as you are done with a request that uses result objects they go away, and the next time you call a Z SQL Method you get a new set of fresh result objects.

Result objects can be used from DTML to display the results of calling a Z SQL Method. For example, add a new DTML Method to your site called *listEmployees* with the following DTML content:

```
<dtml-var standard_html_header>

  <ul>
  <dtml-in list_all_employees>
    <li><dtml-var emp_id>: <dtml-var last>, <dtml-var first>
      makes <dtml-var salary fmt=dollars-and-cents> a year.
    </li>
  </dtml-in>
  </ul>

<dtml-var standard_html_footer>
```

This method calls the *list_all_employees* Z SQL Method from DTML. The *in* tag is used to iterate over each Result object returned by the *list_all_employees* Z SQL Method. Z SQL Methods always return a list of objects, so you will almost certainly use them from the DTML *in* tag unless you are not interested in the results or if the SQL code will never return any results, like *hire_employee*.

The body of the *in* tag is a template that defines what gets rendered for each Result object in the sequence returned by *list_all_employees*. In the case of a table with three employees in it, *listEmployees* might return HTML that looks like this:

```
<html>
  <body>
```

```
  <ul>
    <li>42: Roberts, Bob
      makes $50,000 a year.
    </li>
    <li>101: leCat, Cheeta
      makes $100,000 a year.
    </li>
    <li>99: Junglewoman, Jane
      makes $100,001 a year.
    </li>
  </ul>

  </body>
</html>
```

The *in* tag rendered an HTML list item for each Result object returned by *list_all_employees*.

Next we'll look at how to create user interfaces in order to collect data and pass it to Z SQL Methods.

Providing Arguments to Z SQL Methods

So far, you have the ability to display employees with the the *listEmployees* DTML Method which calls the *list_all_employees* Z SQL Method. Now let's look at how to build a user interface for the *hire_employee* Z SQL Method. Recall that the *hire_employee* accepts four arguments, *emp_id*, *first*, *last*, and *salary*. The *Test* tab on the *hire_employee* method lets you call this method, but this is not very useful for integrating into a web application. You need to create your own input form for your Z SQL Method or call it manually from your application.

The Z Search Interface can create an input form for you automatically. In Chapter 11, "Searching and Categorizing Content", you used the Z Search Interface to build a form/action pair of methods that automatically generated an HTML search form and report screen that queried the Catalog and returned results. The Z Search Interface also works with Z SQL Methods to build a similar set of search/result screens.

Select *Z Search Interface* from the add list and specify *hire_employee* as the *Searchable object*. Enter the value "hireEmployee" for the *Report Id* and "hireEmployeeForm" for the *Search Id* and click *Add*.

Click on the newly created *hireEmployeeForm* and click the *View* tab. Enter an employee_id, a first name, a last name, and salary for a new employee and click *Submit*. Zope returns a screen that says "There was no data matching this query". Because the report form generated by the Z Search Interface is meant to display the result of a Z SQL Method, and the *hire_employee* Z SQL Method does not return any results; it just inserts a new row in the table. Edit the *hireEmployee* DTML Method a little to make it more informative. Select the *hireEmployee* Method. It should contain the following long stretch of DTML:

```
<dtml-var standard_html_header>

<dtml-in hire_employee size=50 start=query_start>

   <dtml-if sequence-start>
```

```
      <dtml-if previous-sequence>

        <a href="<dtml-var URL><dtml-var sequence-query
                 >query_start=<dtml-var
                 previous-sequence-start-number>">
        (Previous <dtml-var previous-sequence-size> results)
        </a>

      </dtml-if previous-sequence>

      <table border>
        <tr>
        </tr>

   </dtml-if sequence-start>

        <tr>
        </tr>

   <dtml-if sequence-end>

      </table>
      <dtml-if next-sequence>

         <a href="<dtml-var URL><dtml-var sequence-query
            >query_start=<dtml-var
            next-sequence-start-number>">
         (Next <dtml-var next-sequence-size> results)
         </a>

      </dtml-if next-sequence>

   </dtml-if sequence-end>

<dtml-else>

  There was no data matching this <dtml-var title_or_id> query.

</dtml-in>

<dtml-var standard_html_footer>
```

This is a pretty big piece of DTML! All of this DTML is meant to dynamically build a batch-oriented tabular result form. Since we don't need this, let's change the *hireEmployee* method to be much simpler:

```
<dtml-var standard_html_header>

<dtml-call hire_employee>

<h1>Employee <dtml-var first> <dtml-var last> was Hired!</h1>

<p><a href="listEmployees">List Employees</a></p>

<p><a href="hireEmployeeForm">Back to hiring</a></p>

<dtml-var standard_html_footer>
```

Now view *hireEmployeeForm* and hire another new employee. Notice how the *hire_employee* method is called from the DTML *call* tag. This is because we know there is no output from the *hire_employee* method. Since there are no results to iterate over, the method does not need to be called with the *in* tag. It can be called simply with the *call* tag.

Now you have a complete user interface for hiring new employees. Using Zope's security system, you can now restrict access to this method to only a certain group of users whom you want to have permission to hire new employees. Keep in mind, the search and report screens generated by the Z Search Interface are just guidelines that you can easily customize to suite your needs.

Next we'll take a closer look at precisely controlling SQL queries. You've already seen how Z SQL Methods allow you to create basic SQL query templates. In the next section you'll learn how to make the most of your query templates.

Dynamic SQL Queries

A Z SQL Method query template can contain DTML that is evaluated when the method is called. This DTML can be used to modify the SQL code that is executed by the relational database. Several SQL specific DTML tags exist to assist you in the construction of complex SQL queries. In the next sections you'll learn about the *sqlvar*, *sqltest*, and *sqlgroup* tags.

Inserting Arguments with the *Sqlvar* Tag

It's pretty important to make sure you insert the right kind of data into a column in a database. You database will complain if you try to use the string "12" where the integer 12 is expected. SQL requires that different types be quoted differently. To make matters worse, different databases have different quoting rules.

In addition to avoiding errors, SQL quoting is important for security. Suppose you had a query that makes a select:

```
select * from employees
  where emp_id=<dtml-var emp_id>
```

This query is unsafe since someone could slip SQL code into your query by entering something like *12; drop table employees* as an *emp_id*. To avoid this problem you need to make sure that your variables are properly quoted. The *sqlvar* tag does this for you. Here is a safe version of the above query that uses *sqlvar*:

```
select * from employees
  where emp_id=<dtml-sqlvar emp_id type=int>
```

The *sqlvar* tag operates similarly to the regular DTML *var* tag in that it inserts values. However it has some tag attributes targeted at SQL type quoting, and dealing with null values. The *sqlvar* tag accepts a number of arguments:

name

The *name* argument is identical to the name argument for the *var* tag. This is the name of a Zope variable or Z SQL Method argument. The value of the variable or argument is inserted into the SQL Query Template. A *name* argument is required, but the "name=" prefix may be omitted.

type

The *type* argument determines the way the *sqlvar* tag should format the value of the variable or argument being inserted in the query template. Valid values for type are *string*, *int*, *float*, or *nb*. *nb* stands for non-blank and means a string with at least one character in it. The *sqlvar* tag *type* argument is required.

optional

The *optional* argument tells the *sqlvar* tag that the variable or argument can be absent or be a null value. If the variable or argument does not exist or is a null value, the *sqlvar* tag does not try to render it. The *sqlvar* tag *optional* argument is optional.

The *type* argument is the key feature of the *sqlvar* tag. It is responsible for correctly quoting the inserted variable. See Appendix A for complete coverage of the *sqlvar* tag.

You should always use the *sqlvar* tag instead of the *var* tag when inserting variables into a SQL code since it correctly quotes variables and keeps your SQL safe.

Equality Comparisons with the *Sqltest* Tag

Many SQL queries involve equality comparison operations. These are queries that ask for all values from the table that are in some kind of equality relationship with the input. For example, you may wish to query the *employees* table for all employees with a salary *greater than* a certain value.

To see how this is done, create a new Z SQL Method named *employees_paid_more_than*. Give it one argument, *salary*, and the following SQL template:

```
select * from employees
  where <dtml-sqltest salary op=gt type=float>
```

Now click *Add and Test*. The *op* tag attribute is set to *gt*, which stands for *greater than*. This Z SQL Method will only return records of employees that have a higher salary than what you enter in this input form. The *sqltest* builds the SQL syntax necessary to safely compare the input to the table column. Type "10000" into the *salary* input and click the *Test* button. As you can see the *sqltest* tag renders this SQL code:

```
select * from employees
  where salary > 10000
```

The *sqltest* tag renders these comparisons to SQL taking into account the type of the variable and the particularities of the database. The *sqltest* tag accepts the following tag parameters:

name

The name of the variable to insert.

type
: The data type of the value to be inserted. This attribute is required and may be one of *string*, *int*, *float*, or *nb*. The nb data type stands for "not blank" and indicates a string that must have a length that is greater than 0. When using the nb type, the *sqltest* tag will not render if the variable is an empty string.

column
: The name of the SQL column, if different than the *name* attribute.

multiple
: A flag indicating whether multiple values may be provided. This lets you test if a column is in a set of variables. For example when *name* is a list of strings "Bob" , "Billy" , `<dtml-sqltest name type="string" multiple>` renders to this SQL: `name in ("Bob", "Billy").`

optional
: A flag indicating if the test is optional. If the test is optional and no value is provided for a variable then no text is inserted. If the value is an empty string, then no text will be inserted only if the type is *nb*.

op
: A parameter used to choose the comparison operator that is rendered. The comparisons are: *eq* (equal to), *gt* (greater than), *lt* (less than), *ge* (greater than or equal to), *le* (less than or equal to), and *ne* (not equal to).

See Appendix A for more information on the *sqltest* tag. If your database supports additional comparison operators such as *like* you can use them with *sqlvar*. For example if *name* is the string "Mc%", the SQL code:

```
<dtml-sqltest name type="string" op="like">
```

would render to:

```
name like 'Mc%'
```

The *sqltest* tag helps you build correct SQL queries. In general your queries will be more flexible and work better with different types of input and different database if you use *sqltest* rather than hand coding comparisons.

Creating Complex Queries with the *Sqlgroup* Tag

The *sqlgroup* tag lets you create SQL queries that support a variable number of arguments. Based on the arguments specified, SQL queries can be made more specific by providing more arguments, or less specific by providing less or no arguments.

Here is an example of an unqualified SQL query:

```
select * from employees
```

Here is an example of a SQL query qualified by salary:

```
select * from employees
where(
  salary > 100000.00
)
```

Here is an example of a SQL query qualified by salary and first name:

```
select * from employees
where(
  salary > 100000.00
  and
  first in ('Jane', 'Cheetah', 'Guido')
)
```

Here is an example of a SQL query qualified by a first and a last name:

```
select * from employees
where(
  first = 'Old'
  and
  last = 'McDonald'
)
```

All three of these queries can be accomplished with one Z SQL Method that creates more specific SQL queries as more arguments are specified. The following SQL template can build all three of the above queries:

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqltest salary op=gt type=float optional>
<dtml-and>
  <dtml-sqltest first op=eq type=nb multiple optional>
<dtml-and>
  <dtml-sqltest last  op=eq type=nb multiple optional>
</dtml-sqlgroup>
```

The *sqlgroup* tag renders the string *where* if the contents of the tag body contain any text and builds the qualifying statements into the query. This *sqlgroup* tag will not render the *where* clause if no arguments are present.

The *sqlgroup* tag consists of three blocks separated by *and* tags. These tags insert the string *and* if the enclosing blocks render a value. This way the correct number of *ands* are included in the query. As more arguments are specified, more qualifying statements are added to the query. In this example, qualifying statements restricted the search with *and* tags, but *or* tags can also be used to expand the search.

This example also illustrates *multiple* attribute on *sqltest* tags. If the value for *first* or *last* is a list, then the right SQL is rendered to specify a group of values instead of a single value.

You can also nest *sqlgroup* tags. For example:

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqlgroup>
     <dtml-sqltest first op=like type=nb>
  <dtml-and>
     <dtml-sqltest last op=like type=nb>
  </dtml-sqlgroup>
<dtml-or>
  <dtml-sqltest salary op=gt type=float>
</dtml-sqlgroup>
```

Given sample arguments, this template renders to SQL like so:

```
select * from employees
where
( (first like 'A%'
   and
   last like 'Smith'
  )
  or
  salary > 20000.0
)
```

You can construct very complex SQL statements with the *sqlgroup* tag. For simple SQL code you won't need to use the *sqlgroup* tag. However, if you find yourself creating a number of different but related Z SQL Methods you should see if you can't accomplish the same thing with one method that uses the *sqlgroup* tag.

Advanced Techniques

So far you've seen how to connect to a relational database, send it queries and commands, and create a user interface. These are the basics of relational database conductivity in Zope.

In the following sections you'll see how to integrate your relational queries more closely with Zope and enhance performance. We'll start by looking at how to pass arguments to Z SQL Methods both explicitly and by acquisition. Then you'll find out how you can call Z SQL Methods directly from URLs using traversal to result objects. Next you'll find out how to make results objects more powerful by binding them to classes. Finally we'll look at caching to improve performance and how Zope handles database transactions.

Calling Z SQL Methods with Explicit Arguments

If you call a Z SQL Method without argument from DTML, the arguments are automatically collected from the environment. This is the technique that we have used so far in this chapter. It works well when you want to query a database from a search form, but sometimes you want

to manually or programmatically query a database. Z SQL Methods can be called with explicit arguments from DTML or Python. For example, to query the *employee_by_id* Z SQL Method manually, the following DTML can be used:

```
        <dtml-var standard_html_header>

          <dtml-in expr="employee_by_id(emp_id=42)">
            <h1><dtml-var last>, <dtml-var first></h1>

            <p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-
var
            first> makes <dtml-var salary fmt=dollars-and-cents> per
year.</p>
          </dtml-in>

        <dtml-var standard_html_footer>
```

Remember, the *employee_by_id* method returns only one record, so the body of the *in* tag in this method will execute only once. In the example you calling the Z SQL Method like any other method and passing it a keyword argument for *emp_id*. The same can be done easily from Python:

```
        ## Script (Python) "join_name"
        ##parameters=id
        ##
        for result in context.employee_by_id(emp_id=id):
            return result.last + ', ' + result.first
```

This script accepts an *id* argument and passes it to *employee_by_id* as the *emp_id* argument. It then iterates over the single result and joins the last name and the first name with a comma.

You can provide more control over your relational data by calling Z SQL Methods with explicit arguments. It's also worth noting that from DTML and Python Z SQL Methods can be called with explicit arguments just like you call other Zope methods.

Acquiring Arguments from other Objects

Z SQL can acquire information from other objects and be used to modify the SQL query. Consider Figure 10-7, which shows a collection of Folders in a organization's web site.

Figure 10-7 Folder structure of an organizational web site

Suppose each department folder has a *department_id* string property that identifies the accounting ledger id for that department. This property could be used by a shared Z SQL Method to query information for just that department. To illustrate, create various nested folders with different *department_id* string properties and then create a Z SQL Method with the id *requisition_something* in the root folder that takes three arguments, *description*, *quantity*, and *unit_cost*. and the following query template:

```
INSERT INTO requisitions
  (
    department_id, description, quantity, unit_cost
  )
VALUES
  (
    <dtml-sqlvar department_id type=string>,
    <dtml-sqlvar description type=string>,
    <dtml-sqlvar quantity type=int>,
    <dtml-sqlvar unit_cost type=float>
  )
```

Now, create a Z Search Interface with a *Search Id* of "requisitionSomethingForm" and the *Report id* of "requisitionSomething". Select the *requisition_something* Z SQL Method as the *Searchable Object* and click *Add*.

Edit the *requisitionSomethingForm* and remove the first input box for the *department_id* field. We don't want the value of *department_id* to come from the form, we want it to come from a property that is acquired.

Now, you should be able to go to a URL like:

```
http://example.org/Departments/Support/requisitionSomethingForm
```

and requisition some punching bags for the Support department. Alternatively, you could go to:

```
http://example.org/Departments/Sales/requisitionSomethingForm
```

And requisition some tacky rubber key-chains with your logo on them for the Sales department. Using Zope's security system as described in Chapter 7, "Users and Security", you can now restrict access to these forms so personnel from departments can requisition items just for their department and not any other.

The interesting thing about this example is that *department_id* was not one of the arguments provided to the query. Instead of getting the value of this variable from an argument, it *acquires* the value from the folder where the Z SQL Method is accessed. In the case of the above URLs, the *requisition_something* Z SQL Method acquires the value from the *Sales* and *Support* folders. This allows you to tailor SQL queries for different purposes. All the departments can share a query but it is customized for each department.

By using acquisition and explicit argument passing you can tailor your SQL queries to your web application.

Traversing to Result Objects

So far you've provided arguments to Z SQL Methods from web forms, explicit argument, and acquisition. You can also provide arguments to Z SQL Methods by calling them from the web with special URLs. This is called *traversing* to results objects. Using this technique you can walk directly up to result objects using URLs.

In order to traverse to result objects with URLs, you must be able to ensure that the SQL Method will return only one result object given one argument. For example, create a new Z SQL Method named *employee_by_id* that accepts one argument, *emp_id*, and has the following SQL Template:

```
select * from employees where
  <dtml-sqltest emp_id op=eq type=int>
```

This method selects one employee out of the *employees* table based on their employee id. Since each employee has a unique id, only one record will be returned. Relational databases can provide these kinds of uniqueness guarantees.

Zope provides a special URL syntax to access ZSQL Methods that always return a single result. The URL consists of the URL of the ZSQL Method followed by the argument name followed by the argument value. For example, *http://localhost:8080/employee_by_id/emp_id/42*. Note, this URL will return a single result object where as if you queried the ZSQL Method from DTML and passed it a single argument it would return a list of results that happend to only have one item in it.

Unfortunately the result object you get with this URL is not very interesting to look at. It has no way to display itself in HTML. You still need to display the result object. To do this, you can call a DTML Method on the result object. This can be done using the normal URL acquisition rules described in Chapter 10, "Advanced Zope Scripting". For example, consider the following URL:

```
http://localhost:8080/employee_by_id/emp_id/42/viewEmployee
```

Here we see the *employee_by_id* Z SQL Method being passed the *emp_id* argument by URL. The *viewEmployee* method is then called on the result object. Let's create a *viewEmployee* DTML Method and try it out. Create a new DTML Method named *viewEmployee* and give it the following content:

```
<dtml-var standard_html_header>

  <h1><dtml-var last>, <dtml-var first></h1>

  <p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-
var
```

```
          first> makes <dtml-var salary fmt=dollars-and-cents> per
year.</p>

        <dtml-var standard_html_footer>
```

Now when you go to the URL *http://localhost:8080/employee_by_id/emp_id/42/viewEmployee* the *viewEmployee* DTML Method is bound the result object that is returned by *employee_by_id*. The *viewEmployee* method can be used as a generic template used by many different Z SQL Methods that all return employee records.

Since the *employee_by_id* method only accepts one argument, it isn't even necessary to specify *emp_id* in the URL to qualify the numeric argument. If your Z SQL Method has one argument, then you can configure the Z SQL Method to accept only one extra path element argument instead of a pair of arguments. This example can be simplified even more by selecting the *employee_by_id* Z SQL Method and clicking on the *Advanced* tab. Here, you can see a check box called *Allow "Simple" Direct Traversal*. Check this box and click *Change*. Now, you can browse employee records with simpler URLs like *http://localhost:8080/employee_by_id/42/viewEmployee*. Notice how no *emp_id* qualifier is declared in the URL.

Traversal gives you an easy way to provide arguments and bind methods to Z SQL Methods and their results. Next we'll show you how to bind whole classes to result objects to make them even more powerful.

Binding Classes to Result Objects

A result object has an attribute for each column in results row. However, result objects do not have any methods, just attributes.

There are two ways to bind a method to a Result object. As you saw in the previous section, you can bind DTML and other methods to Z SQL Method Result objects using traversal to the results object coupled with the normal URL based acquisition bind mechanism described in Chapter 10, "Advanced Zope Scripting". You can also bind methods to Result objects by defining a Python class that gets *mixed in* with the normal, simple Result object class. These classes are defined in the same location as External Methods in the filesystem, in Zope's *Extensions* directory. Python classes are collections of methods and attributes. By associating a class with a Result object, you can make the Result object have a rich API and user interface.

Classes used to bind methods and other class attributes to Result classes are called *Pluggable Brains*, or just *Brains*. Consider the example Python class:

```
        class Employee:

          def fullName(self):
            """ The full name in the form 'John Doe' """
            return self.first + ' ' + self.last
```

When result objects with this Brains class are created as the result of a Z SQL Method query, the Results objects will have *Employee* as a base class. This means that the record objects will have all the methods defined in the *Employee* class, giving them behavior, as well as data.

To use this class, create the above class in the *Employee.py* file in the *Extensions* directory. Go the *Advanced* tab of the *employee_by_id* Z SQL Method and enter *Employee* in the *Class Name* field, and *Employee* in the *Class File* field and click *Save Changes*. Now you can edit the *employeeView* DTML Method to contain:

```
        <dtml-var standard_html_header>

          <h1><dtml-var fullName></h1>

          <p><dtml-var first>'s employee id is <dtml-var emp_id>.  <dtml-
var
          first> makes <dtml-var salary fmt=dollars-and-cents> per
year.</p>

        <dtml-var standard_html_footer>
```

Now when you go to the URL *http://localhost:8080/employee_by_id/42/viewEmployee* the *fullName* method is called by the *viewEmployee* DTML Method. The *fullName* method is defined in the *Employee* class of the *Employee* module and is bound to the result object returned by *employee_by_id*

Brains provide a very powerful facility which allows you to treat your relational data in a more object-centric way. For example, not only can you access the *fullName* method using direct traversal, but you can use it anywhere you handle result objects. For example:

```
        <dtml-in employee_by_id>
          <dtml-var fullName>
        </dtml-in>
```

For all practical purposes your Z SQL Method returns a sequence of smart objects, not just data.

This example only scratches the surface of what can be done with Brains classes. Python programming is beyond the scope of this book so we will only go a little farther here. However, you could create brains classes that accessed network resources, called other Z SQL Methods, performed all kinds of business logic.

Here's a more powerful example of brains. Suppose that you have an *managers* table to go with the *employees* table that you've used so far. Suppose also that you have a *manager_by_id* Z SQL Method that returns a manager id manager given an *emp_id* argument:

```
        select manager_id from managers where
          <dtml-sqltest emp_id type=int op=eq>
```

You could use this Z SQL Method in your brains class like so:

```
class Employee:

    def manager(self):
        """
        Returns this employee's manager or None if the
        employee does not have a manager.
        """
        # Calls the manager_by_id Z SQL Method.
        records=self.manager_by_id(emp_id=self.emp_id)
        if records:
            manager_id=records[0].manager_id
            # Return an employee object by calling the
            # employee_by_id Z SQL Method with the manager's emp_id
            return self.employee_by_id(emp_id=manager_id)[0]
```

This `Employee` class shows how methods can use other Zope objects to weave together relational data to make it seem like a collection of objects. The `manager` method calls two Z SQL Methods, one to figure out the emp_id of the employee's manager, and another to return a new Result object representing the manager. You can now treat employee objects as though they have simple references to their manager objects. For example you could add something like this to the *viewEmployee* DTML Method:

```
<dtml-if manager>
  <dtml-with manager>
    <p> My manager is <dtml-var first> <dtml-var last>.</p>
  </dtml-with>
</dtml-if>
```

As you can see brains can be both complex and powerful. When designing relational database applications you should try to keep things simple and add complexity slowly. It's important to make sure that your brains classes don't add lots of unneeded overhead.

Caching Results

You can increase the performance of your SQL queries with caching. Caching stores Z SQL Method results so that if you call the same method with the same arguments frequently, you won't have to connect to the database every time. Depending on your application, caching can dramatically improve performance.

To control caching, go to the *Advanced* tab of a SQL Method. You have three different cache controls as shown in Figure 10-8.

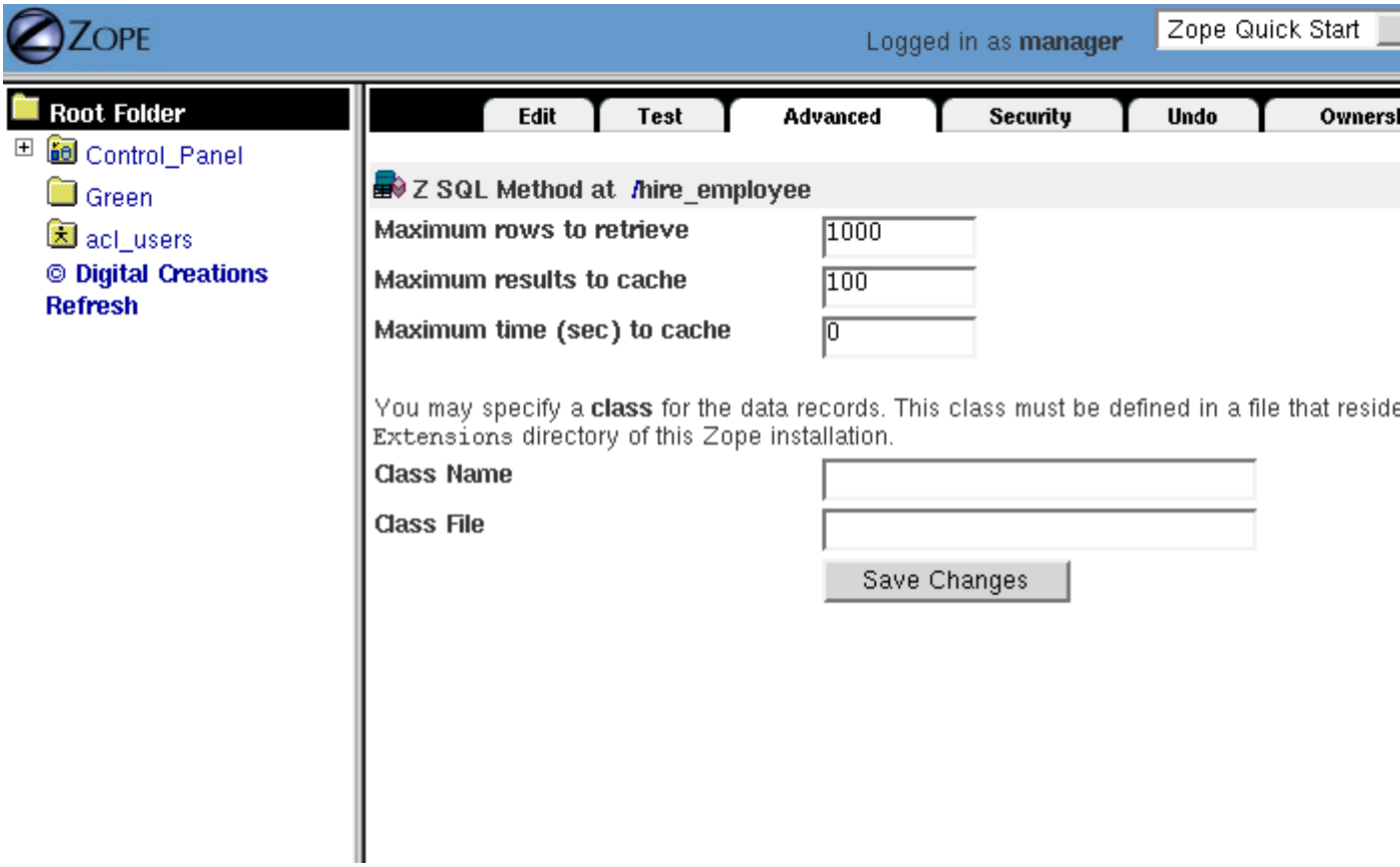


Figure 10-8 Caching controls for Z SQL Methods

The *Maximum number of rows received* field controls how much data to cache for each query. The *Maximum number of results to cache* field controls how many queries to cache. The *Maximum time (in seconds) to cache results* controls how long cached queries are saved for. In general, the larger you set these values the greater your performance increase, but the more memory Zope will consume. As with any performance tuning, you should experiment to find the optimum settings for your application.

In general you will want to set the maximum results to cache to just high enough and the maximum time to cache to be just long enough for your application. For site with few hits you should cache results for longer, and for sites with lots of hits you should cache results for a shorter period of time. For machines with lots of memory you should increase the number of cached results. To disable caching set the cache time to zero seconds. For most queries, the default value of 1000 for the maximum number of rows retrieved will be adequate. For extremely large queries you may have to increase this number in order to retrieve all your results.

Transactions

A transaction is a group of operations that can be undone all at once. As you saw in Chapter 1, "Introducing Zope", all changes done to Zope are done within transactions. Transactions ensure data integrity. When using a system that is not transactional and one of your web actions changes ten objects, and then fails to change the eleventh, then your data is now

inconsistent. Transactions allow you to revert all the changes you made during a request if an error occurs.

Imagine the case where you have a web page that bills a customer for goods received. This page first deducts the goods from the inventory, and then deducts the amount from the customers account. If the second operations fails for some reason you want to make sure the change to the inventory doesn't take effect.

Most commercial and open source relational databases support transactions. If your relational database supports transactions, Zope will make sure that they are tied to Zope transactions. This ensures data integrity across both Zope and your relational database. If either Zope or the relational database aborts the transaction, the entire transaction is aborted.

Summary

Zope allows you to build web applications with relational databases. Unlike many web application servers, Zope has its own object database and does not require the use of relational databases to store information.

Zope lets you use relational data just like you use other Zope objects. You can connect your relational data to business logic with scripts and brains, you can query your relational data with Z SQL Methods and presentation tools like DTML, and your can even use advanced Zope features like URL traversal, acquisition, undo and security while working with relational data.

Zopebuch: Inhaltsverzeichnis

Skalierbarkeit und ZEO

Bekommt eine Website mehr Anfragen als sie abarbeiten kann, wird sie langsam und gibt die Seiten nur zögernd heraus. Eine grosse Menge von Anfragen kann einen Server sogar dermassen überlasten, das Anfragen nicht mehr beantwortet werden können und der Server unter Umständen sogar unkontrolliert abstürzt. Dies ist nun nicht ein Problem was nur Zope betrifft, auch andere Serverdienste sind empfindlich gegen Überlast. Oft wird dieses Problem durch den Einsatz mehrerer Rechner mit identischen Serverkonfigurationen gelöst. Fällt nun einer der identischen Rechner aus, können auf eine einfache Art und Weise die noch zur Verfügung stehenden Rechner die Aufgabe der ausgefallenden Recheneinheit übernehmen.

Die Dienste auf mehrere Rechner zu verteilen hat seine Vor- und Nachteile. Wenn man beispielsweise fünf Rechner mit identischen Zope Installationen in Betrieb nehmen möchte, muss auch sichergestellt sein, dass auf allen Zope Instanzen die gleichen Informationen vorliegen. Dies ist nicht besonders schwer zu bewerkstelligen, wenn man als einziger User eine handvoll statischer Objekte anlegt und diese auf jeder Zope Instanz auf gleiche Art und Weise installiert. Auf einem Zope-Server in einer grossen Organisation mit mehreren Tausenden sich schnell änderen Objekten würde die manuelle Abgleichung der fünf verschiedenen Zope-Instanzen zu einer nicht mehr zu bewältigenden Aufgabe.

Um dieses Problem zu lösen hat die Zope Corporation die sogenannten "Zope Enterprise Objects" geschaffen - abgekürzt "ZEO" genannt. Dieses Kapitel gibt euch einen nur einen kurzen Einblick wie ZEO installiert wird. Viele andere tolle Sachen, die Ihr mit ZEO noch

machen könnt, werden hier nicht abgehandelt. Um mehr über ZEO zu erfahren, lest euch die die mitgelieferte Dokumentation durch. Das ZEO Forum ist ebenfalls eine gute Informationsquelle ("Anmerkung des Übersetzers: Aber wo bloss ???").

Was ist ZEO?

ZEO ist ein System, was es uns ermöglicht, eine Website auf mehreren Rechnern zu betreiben. Eine derartige Anordnung von Rechnern wird auch "Cluster" genannt. Mit einer solchen Anordnung und einer Lastverteilungskomponente kann eine Lastverteilung erzielt werden (englisch: "load balancing"). Weil Zope auf mehreren Rechnern zur Verfügung steht, können durch eine geeignete Lastverteilungskomponente die Anfragen auf mehrere Rechner verteilt werden, falls die Anfragelast das erforderlich macht. Ausserdem können im Falle eines Serverausfalles, die noch intakten Rechner die Anfragen bedienen. Mit einem derartigem Serveraufbau kann man also ganz in Ruhe den defekten Rechner austauschen oder reparieren, ohne das von aussen ein Serverausfall spürbar wird.

Mit ZEO kann Zope auf mehreren Rechnern betrieben werden. ZEO garantiert, dass alle beteiligten Zope Installationen jederzeit auf genau die selbe Datenbank zugreifen. ZEO benutzt dazu eine Client/Server- Architektur. Alle Clients verbinden sich mit einer zentralen Instanz, dem ZEO Speicher Server, so wie auch in Bild 11-1 zu sehen ist.

Die Begrifflichkeit kann verwirrend wirken, weil man normalerweise gewohnt ist, von Zope als einen Server zu sprechen. Wenn ZEO allerdings im Spiel ist, dann übernimmt Zope zugleich Serverfunktionaliät (für die Beantwortung von Webanfragen) sowie Klientfunktionalität (um Daten vom ZEO-Server zu erhalten.)

ZEO Clients und Server kommunizieren mit einem standardisierten Internetprotokoll. Auf diese Art und Weise ist es möglich, beide im selben Raum oder aber auch in einem anderem Land zu betreiben.

Wann empfiehlt es sich ZEO einzusetzen

ZEO kann viele Anfragen in einer ausfallsicheren Art und Weise beantworten. Wenn eure Website nicht Millionen von Anfragen bekommt, dann braucht Ihr ZEO vielleicht nicht. Es gibt keine feststehenden Regeln wann man ZEO einsetzen sollte und wann nicht, aber folgende Argumente würden für den Einsatz von ZEO sprechen:

- Eure Webseite bekommt deutlich mehr Anfragen als der Rechner zügig ausliefern kann. Zope ist ein hochgerüstetes System, und Zope kann Millionen von Anfragen beantworten (natürlich abhängig von der eingesetzten Hardware). Wenn diese Leistung nicht mehr ausreicht um eure Anforderungen zu erfüllen, dann sollte ZEO zum Einsatz kommen.
- Eure Website ist sehr wichtig und erfordert konstante 24/7 Verfügbarkeit. In diesem Falle erlaubt ZEO den Einsatz mehrerer sofortige einsatzfähiger Ersatzserver.
- Ihr wollt Eure Website global über mehrere Mirror-Sites anbieten und an jedem Standort dieselben Daten per ZEO-Client zugreifen und schreiben können.
- Ihr wollt einen ZEO Client nach Fehlern untersuchen (debuggen), während die anderen Zeo-Clients noch weiterhin Webseiten ausliefern.

All diese Punkte sind bereits ziemlich vorgeschrittene Themen. Einrichtung, Konfiguration und Betrieb eines derartigen Systems erfordert schon vorgeschrittene Administrationskenntnisse und Betriebsmittel. Die meisten Zope User werden sich mit ZEO nicht auseinandersetzen zu brauchen oder haben nicht die erforderlichen Kenntnisse, ein verteiltes System wie ZEO einzusetzen. ZEO macht zwar Spass, aber ist auch aufwändiger als ein ganz normaler einzelner Zope - Server.

Installation und Inbetriebnahme des ZEO Systems

Am häufigsten wird ZEO mit einem ZEO Server und mehreren ZEO Clients betrieben. Bevor Ihr beginnt eurer eigenes ZEO-System zu installieren und zu konfigurieren, beachtet folgende Punkte:

- Alle ZEO Client- und Serversysteme müssen die selbe Zope-Version in Betrieb nehmen. Es sollten nur die neusten ZEO und Zope-Version verwendet werden. Dies ist erforderlich, um einen einwandfreien Betrieb des Gesamtsystemes zu ermöglichen.
- Alle ZEO Client Systeme sollten die selben Produkte von Drittherstellern installiert haben, und die Versionen der Produkte sollten identisch sein.
- Falls Dein Zope System auf externe Betriebsmittel wie E-Mailserver oder Relationale Datenbanken zugreifen muss, muss allen ZEO-Clients der Zugriff auf diese Betriebsmittel ermöglicht werden.
- Langsame oder zeitweise gestörte Netzwerkverbindungen zwischen dem Serversystem und dem Clientsystem können die Leistung der ZEO-Clientsysteme beeinträchtigen. Die ZEO Clientsysteme sollten eine gute Verbindung zu dem Serversystem haben.

ZEO wird nicht mit Zope mitgeliefert, Ihr müsst es euch von der Produktseite bei Zope.org runterladen.

Die Installation von ZEO erfordert ein wenig Handarbeit. Um ZEO zu installieren, lade zunächst das gepackte Archiv ZEO-1.0.tgz von der Zope.org Website herunter und speichere diese Datei in das ZopeInstallationsverzeichnis. Nun entpacke das Archiv. Auf einem UNIX System kann das mit folgendem Befehl bewirkt werden:

```
tar -zxf ZEO-1.0.tgz
```

Auf einem Windows System kannst Du das Archiv mit Winzip entpacken. Bevor Ihr nun ZEO installiert, empfehlen wir noch dringend eine Sicherung des Zope System vorzunehmen.

Nun solltet Ihr ein ZEO-1.0 Verzeichnis haben. Als nächstes müsst Ihr noch ein paar Dateien vom Zope Stammverzeichnis in das Zope top level lib/python Verzeichnis kopieren. Unter Unix funktioniert dieses so:

```
cp -R ZEO-1.0/ZEO lib/python
```

Falls Du unter Windows arbeitest, kannst Du folgende DOS Befehle absetzen um eure ZEO-Files zu kopieren:

```
C:\...Zope\>xcopy ZEO-1.0\* lib\python /S
```

Danach legt Ihr noch eine Datei mit dem Namen custom_zodb.py im Zope Hauptverzeichnis an. In diese Datei schreibt Ihr dann folgendene Python Anweisungen:

```
import ZEO.ClientStorage
Storage=ZEO.ClientStorage.ClientStorage(('localhost',7700))
```

Damit ist der Zope Server zu einem ZEO Client konfiguriert worden. Falls dem ClientStorage ein Tuple üben wird, wie dieses Programm es macht, beachtet folgendes: Das Tuple hat zwei Elemente, eine Zeichenkette, welche die Serveradresse enthät, sowie einen Dienstzugangspunkt (Port) unter dem der Server seine Dienste anbietet. In diesem Beispiel zeigen wir, wie Client und Server auf demselbem Rechner laufen. Aus diesem Grunde haben wir den Rechnernamen auf "localhost" gesetzt.

Okay, nun ist ZEO auf einem einzigen Rechner soweit startklar. Versucht nun zunächst den Server zu starten. Geht hierzu in einem Terminalprogramm oder dem DOS-Eingabefenster in das Zope Hauptverzeichnis und tippt folgendenes ein:

```
python lib/python/ZEO/start.py -p 7700
```

Dies wird den ZEO Server so starten, dass er unter dem Dienstzugangspunkt(Port) mit der Nummer 7700 zu erreichen ist. Nun fehlt noch ein Zope, der als ZEO-Client diesen ZEO Server nutzt. Starte in einem anderem Fenster den Zope Server ganz normal mit dem z2.py Skript:

```
python z2.py -D

------
2000-10-04T20:43:11 INFO(0) client Trying to connect to server
------
2000-10-04T20:43:11 INFO(0) ClientStorage Connected to storage
------
2000-10-04T20:43:12 PROBLEM(100) ZServer Computing default pinky
------
2000-10-04T20:43:12 INFO(0) ZServer Medusa (V1.19) started at Wed Oct  4
15:43:12 2000
        Hostname: pinky.zopezoo.org
        Port:8080
```

Beachte, dass in dem obigen Beispiel, Zope zunächst darauf hinweist, dass Zope versucht sich mit dem (ZEO-) Server in Verbindung zu setzen (client Trying to connect to server) um anschliessend darüber zu informieren, dass er es tatsaechlich geschafft hat (ClientStorage Connected to storage). Da ja alles wie geplant funktioniert, kannst Du Dich jetzt wie gewohnt mit http://localhost:8080/manage (oder halt eine andere URL, auf der der Zope-Server zu erreichen ist.) in das Zope-Verwaltungstool einloggen.

Wie Du gesehen hast, sieht noch alles genauso aus wie vorher. Gehe nun in das Kontrollfeld (Control Panel) und klicke auf Datenbankverwaltung (Database Management). Hier wird angezeigt, das Zope mit einem ZEO Server verbunden ist.

ZEO auf einem einzigen Computer ist schonmal ein guter Einstieg, sich mit dem ZEO-System vertraut zu machen und zu sehen wie es funktioniert. ZEO auf einem einzigen Computer zu

betreiben, kann die Leistungsfähig der Website nicht verbessern, tatsächlich wird die Leistungsfähigkeit oftmals ein wenig reduziert. Um nun aber wirklich in den Genuss der Vorteile eines ZEO-Systems zu kommen, muss ZEO auf mehreren Rechnern installiert und funktionsfähig sein. Wie man ein solches System in Betrieb nimmt, erklären wir nun im nächsten Abschnitt.

Wie man ZEO auf mehreren Rechnern zum Einsatz bringt.

Die Vorgehensweise ZEO auf mehreren Rechnern zum Betrieb zu bringen, unterscheidet sich von der Inbetriebnahme auf einem einzelnem Rechner kaum. Im Allgemeinen gibt es zwei Arbeitsschritte. Im ersten Schritt startet man einen ZEO Server und im zweitem Schritt startet man einen oder mehrere ZEO Clientsysteme.

Mal angenommen, Ihr hättet vier Rechner. Der erste Rechner mit dem Namen "zooserver" wäre der Zeo Server, und die anderen drei Rechner mit den Namen "zeoclient1","zeoclient2","zeoclient3" wären die Zeo-Clients. Dann wäre der erste Schritt, den Server auf dem Rechner mit dem Namen "zooserver zu starten".

Um den ZEO Server so zu starten, dass er Aufträge; auf dem Dienstzugangspunkt (Port) mit der Nummer 9999 auf dem Rechner "zooserver" bearbeitet, starte den Server mit dem Skript start.py wie folgt:

```
python lib/python/ZEO/start.py -p 9999 -h zooserver.zopezoo.org
```

Dies wird den ZEO Server starten. Nun kannst Du die ZEO-Clients starten, indem Du auf jeden Rechner das Skript custom_zodb.py wie folgt anpasst:

```
import ZEO.ClientStorage
Storage=ZEO.ClientStorage.ClientStorage(('zooserver.zopezoo.org',9999))
```

Nun kannst Du auf jedem Client-Rechner das Skript z2.py, wie im letzten Abschnitt (Installation und Inbetriebnahme) erklärt, starten. Beachte, dass bei jedem Client-System die Parameter "host" und "port" (Dienstzugangspunkt) identisch sind. Wenn Du alle drei Rechner mit der gleichen Client-Konfiguration gestartet hast, wirst Du bemerken, dass alle drei Rechner dieselbe Zope-Website anzeigen. Dies kannst Du dadurch überprüfen, dass Du alle Rechner nacheinander auf dem Dienstzugangspunkt (Port) 8080 mit dem Webbrowser untersuchst.

Du möchtest bestimmt ZEO deswegen auf mehreren Rechnern in Betrieb nehmen, um Geschwindkeitsvorteile zu erzielen. Mehrere Rechner in Betrieb zu nehmen, sollte ja auch heissen, dass mehr Seiten pro Zeiteinheit ausgeliefert werden können. Eine geschickte Verteilung der Anfragelast auf mehrere Rechner erfordert jedoch ein wenig Feinarbeit in der Einrichtung des Gesamtsystems. Der nächste Abschnitt erklärt, warum und wie die Last auf mehrere Rechner verteilt werden kann.

Wie man die Last verteilt

Im letztem Beispiel hatten wir einen ZEO-Server und drei ZEO Clients mit den Bezeichnungen zeoclient1. zeoclient2 und zeoclient3. Die drei ZEO Clients sind mit dem ZEO-Server verbunden und jeder Client ist auf seine korrekte Arbeitsweise überprüft worden.

Nun haben wir also drei Rechner, die Inhalte an die Besucher der Website ausliefern können. Das nächste Problem ist nun die einkommenden Seitenanfragen auf alle drei Rechner zu verteilen. Die Besucher der Website sollen aber nur www.zopezoo.org kennen, es wäre viel zu umständlich, die Webseitenbesucher darauf hinzuweisen, dass es ebenfalls möglich ist, die Rechner zeoclient1, zeoclient2, oder zeoclient3 zu benutzen. Auch wäre eine gleichmässige Verteilung der Anfragelast nicht leicht zu bewerkstelligen. Im Grunde genommen wäre es aber viel einfacher den Vorgang der Lastverteilung zu automatisieren.

Es gibt eine Reihe von Lösungsansätzen für dieses Problem, einige davon sind einfach, die anderen sind fortgeschritten und dann gibt es auch auch noch eine kostenintensive Lösung. Im nächsten Abschnitt werden wir auf die üblichsten Formen, Webseitenanfragen auf mehrere Rechner zu verteilen, eingehen. Jede Lösung nutzt eine andere Technologie, während die einen Lösungen mit Hilfe freier oder kommerzieller Software aufbauen, bauen die anderen mit auf Spezialhardware auf.

Die Benutzer wählen einen Server aus

Die einfachste Methode eine Lastverteilung zu erreichen ist es den Besuchern eine Auswahl von Rechnern anzubieten, unter der sie einen ZEO-Client erreichen können. Diese Vorgehensweise erfordert keine zusäztliche Software oder Hardware, es ist nur erforderlich die Liste der verfügbaren ZEO-Clients stets aktuell zu halten. Mit dieser Methode entscheidet also der Benutzer, mit welchem ZEO-Client er in Kontakt treten wird.

Es sei angemerkt, dass diese Methode der Lastverteilung passiv erfolgt, daher hast Du keine Kontrolle darüber für welchen ZEO-Client der Kunde sich entscheidet. Wenn der Benutzer sich für keinen der Serveralternativen entscheidet, wird die Hauptlast weiterhin von dem Hauptrechner mit der Bezeichnung www.zopezoo.org getragen.

Wenn Du keinen Zugriff auf die gespiegelten Websites hast, ist das eine ganz brauchbare Loesung. Wenn eine gespiegelte Website ausfällt, kann der Besucher immer noch entscheiden zu der Website zurückzukehren, über die Du ja auch technischen Zugang hast, um eine andere gespiegelte Site auszuwählen.

Die Methode steigert schon deutlich die Leistungsfähigkeit der Website. Der Benutzer wählt zum Beispiel eine gespiegelte Website aus, die geographisch näher liegt und hat dadurch wahrscheinlich eine gute Antwortzeit bei der Navigation durch die Website. Wenn Ihr beispielsweise einen Server in Dortmund und einen in Sydney stehen hättet, dann wären die User aus Australien meist gut beraten den Server in Syndey auszuwählen.

Um diese Methode zum Einsatz zu bringen, lege ein neues Attribut mit dem Namen "mirror_servers" im Hauptverzeichnis an. In jeder Zeile dieser Property, schreibe eine URL für jeden ZEO client - ein Beispiel ist in Abbildung 11-2 zu sehen.

Figure of property with URLs to mirrors

Figure 11-2 Figure of property with URLs to mirrors

Füge nun etwas DTML zu Deiner Website hinzu, um die Liste der Website-Spiegel anzuzeigen:

Dieses DTML listet alle Spiegel auf, aus denen der Besucher wählen kann. Bei diesem Model ist es gut die Computer so zu benennen, dass die Besucher in der Wahl ihres Spiegels unterstützt werden. Zum Beispiel, wenn Du die Belastung geographisch aufteilst, benenne die Computer nach dem Namen der Länder.

Falls Du den Usern nicht einen beliebigen Server auswählen lassen willst, kommt noch folgende Mö,glichkeit in Frage: eine Methode index_html nimmt zufällig die Weiterleitung vor. Zum Beispiel könntest Du folgendes DTML-Programm in Deiner www.zopezoo.org verwenden:

```
<dtml-call>
```

This code will redirect any visitors to www.zopezoo.org to a random mirror server. Dieses Programm leitet jeden Besucher der Site www.zopezoo.org zu einem zufälligen Spiegel Server weiter.

Lastverteilung mit Round-Robin DNS.

Das "Domain Name System" oder kurz "DNS" ist ein Verzeichnisdienst ähnlich einem Telefonbuch, der Computernamen (wie zB.: "www.zope.org") in numerische Adressen abbilden kann. So ist es beispielsweise auch möglich, dass einem Rechnernamen mehrere numerische Adressen zugeordnet wird. Durch diesen Trick, der auch als "Round-Robin DNS" bezeichnet wird, wird wie unten in der Abbildung 11-3 auch zu sehen ist, auf eine einfache Art und Weise eine Lastverteilung erreicht.

Figure 11-3 Lastverteilung mit Round-Robin DNS.

Immer wenn www.zopezoo.org im DNS-Verzeichnisdienst angefragt wird, gibt der zuständige Nameserver (zB. Bind oder tinydns) eine der drei Internet-Adressen der Rechner zeoclient1, zeoclient2 oder zeoclient3 heraus und zwar immer in abwechselnder Reihenfolge nacheinander. Auf diese Art und Weise wird die Anfragelast unter den verschiedenen ZEO-Clients verteilt.

Dies ist keineswegs eine optimale Lastverteilungsstrategie, weil DNS-Informationen von anderen Nameservern zwischengespeichert werden. Sobald ein Besucher also die Internet-Adresse für www.zopezoo.org aufgelöst hat, surft er für einen gewissen Zeitraum auf dem selben ZEO-Client. Im allgemeinen ist die dadurch erzielte Lastverteilung durchaus akzeptabel, weil sich die Summe aller Anfragen in der Regel durchaus auf alle ZEO-Clients verteilt.

Eine Schattenseite dieser Lösung ist, daß manche Nameserver die IP-Adresse der Website www.zopezoo.org für Stunden, wenn nicht sogar Tage, auf denselben ZEO-Client abbilden. Wenn ein ZEO-Client ausserhalb Deines Wirkungsbereich liegt und dieser aus irgendeinem Grunde ausfällt, dann wird für einen Bruchteil 1/N von Internetnutzern (wobei N die Anzahl

der ZEO-Clients ist) Deine Website nicht erreichbar sein, bis der für sie zuständige Nameserver seine gespeicherten DNS-Informationen erneut aktualisiert.

Deinen zuständigen DNS-Server so einzurichten, das die DNS-Auflösung im Round-Robin ausgeführt wird, ist eine ziemlich fortgeschrittene Technik, die wir in diesem Buch nicht mehr abhandeln werden. Eine gute Einführung in diese Technik, kann aber in der Dokumentation des Apache-Webservers gefunden werden. Lastverteilung mit Round-Robin DNS ist eine nützliche und preiswerte Methode, aber ist doch auch mit Nachteilen verbunden. DNS-Namensdienste können merkwürdige Speicherstrategien haben, der Wunsch nach Verteilung der Last auf mehreren ZEO-Server ist halt von der Strategie der Namensdienste Anfragelast zu verteilen, abhängig geworden. Der nächste Abschnitt erklärt eine komplexere aber auch effektivere Methode Last zu verteilen. Diese Methode wird als "Layer 4 Switchung" (Schicht 4 Lastverteilung) bezeichnet.

Lastverteilung mit Hilfe eines Layer 4 Switches

Mit "Layer 4 Switching" wird die Last transparent, daher für die Surfer unmerklich auf eine Anzahl Rechner umgelenkt. Die Technik ist schon sehr fortgeschritten und wird in diesem Handbuch auch nicht weiter abgehandelt, aber es lohnt sich auf die verschiedenen Produkte hinzuweisen, die das "Layer 4 switching" für euch leisten können.

Beim "Layer 4 Switching" verteilt ein Switch die hereinkommenden Anfragen auf eine Gruppe von ZEO Clienten. Der Switch nimmt die Verteilung nach der ihm gegebenen Konfiguration vor, die Arbeitsweise wird in Abbildung 11-4 dargestellt.

Skizze: Lastverteilung mit einem Layer 4 Switch

Ab hier fette Baustelle

There are hardware and software Layer 4 switches. There are a number of software solutions, but one in general that stands out is the Linux Virtual Server (LVS). This is an extension to the free Linux operating system that lets you turn a Linux computer into a Layer 4 switch. More information on the LVS can be found on its web site.

Es gibt Layer 4 Switches in Hardware- und Softwareausführung. Es gibt eine Reihe von Hardware-Lösungen die für sich beanspruchen eine höhere Leistung zu erbringen wie die reinen Softwarelösungen.(like LVS???)

There are also a number of hardware solutions that claim higher performance than software based solutions like LVS. Cisco Systems has a hardware router called LocalDirector that works as a Layer 4 switch, and Alteon also makes a popular Layer 4 switch.

Wie man mit einer für das Gesamtsystem kritischen Einzel-Kompententen umgeht (Single Point of Failure)

Ohne ZEO ist Dein komplettes Zope System eine kritische Einzelkomponente. Mit ZEO ist es möglich die Gefahr eines Komplettausfalles auf mehrere Rechner zu verteilen, daher der von dem Gesamtsystem erbrachte Dienst ist nur dann bedroht, wenn alle Rechner auf einmal ausfallen. Das Gesamtsystem kann also noch weiterarbeiten, wenn nur ein Rechner

ausf&aumll;t und seinen Dienst einstellt - die restlichen Systeme übernehmen dann einfach die durch den Ausfall erzeugte Mehrarbeit.

Zum Zeitpunkt der Texterstellung, waren ZEO-System nicht komplett ausfallsicher, weil bei Ausfall des zentralen ZEO-Servers, das komplette ZEO-System zum Stillstand kommt - daher es gibt keine Möglichkeit in diesem Fall den zentralen ZEO-Server zu ersetzen. Die nachfolgend beschriebenen Methoden versuchen die Risiken eines Ausfalls durch Verteilen des ZEO-Systems auf mehrere Rechner zu mildern.

Eine weitverbreitete Methode ist es, das Ausfallrisiko des ZEO-Servers durch Einsatz hochwertiger Hardware- Komponenten, regelmässige Sicherungen und den Einsatz leicht zu ersetzender, billiger Standardrechner für die ZEO-Clients, zu verringern.

By investing the bulk of your infrastructure budget on making your ZEO server rock solid (redundant power supplies, RAID, and other fail-safe methods) you can be pretty well assured that your ZEO server will remain up, even if a handful of your inexpensive ZEO clients fail.

Dadurch das der ZEO-Server auf teurer und hochwertiger Hardware läuft (redundante Stromversorgung, RAID und andere Sicherheitsvorkehrungen) kann man schon recht sicher sein, das der ZEO-Server stabil läuft und gegebenenfalls im Falle einer Stöhrung schnell wieder zu Einsatz kommen kann.

Like Layer 4 switching, there are a number of products, software and hardware, that help you mitigate this kind of risk. One popular software solution for linux is called fake. Fake is a Linux based utility that can make a backup computer take over for a failed primary computer by "faking out" network addresses. When used in conjunction with monitoring utilities like mon or heartbeat, fake can guarantee almost 100% up-time of your ZEO server and Layer 4 switches. Using fake in this way is beyond the scope of this book.

Anmerkung eines unbekannten Benutzers: Why cann't we program ZEO so that it will make one of the client ZEOs take over the failed ZEO server? Warum können wir ZEO nicht so umprogrammieren, dass einer der ZEO-Clients die Rolle eines ausfallenden ZEO-Servers übernehmen kann ?

So far, we've explained these techniques for mitigating a single point of failure: Bis jetzt haben wir folgende Techniken beschrieben

- Various tools (mirrors, round-robin DNS, Layer 4 switching) can be used to multiplex requests across multiple computers.
- ZEO can be used to distribute your database (ZEO server) to multiple ZEO clients.
- fake, and other tools can be used to provide redundant servers and Layer 4 switches.

The final piece of the puzzle is the ZEO server itself, and where it stores its information. If your primary ZEO server fails, how can your backup ZEO server ensure it has the most recent information that was contained in the primary server? As usual, there are several ways to solve this problem, and they are covered in the next section.

Der ZEO Server im Detail

Bevor wir die Details eines ZEO Server darstellen, werden wir noch einige Details über die Funktion des Zopespeichers im Allgemeinen ausführen.

Zope speichert seine Objekte oder andere Informationen nicht unmittelbar auf Festplatte. Stattdessen benutzt Zope eine Speicherkomponente, die sich um die Einzelheiten eines Speichervorgangs kümmert.

This is a very flexible model, because Zope no longer needs to be concerned about opening files, or reading and writing from databases, or sending data across a network (in the case of ZEO). Each particular storage takes care of that task on Zope's behalf.

For example, a plain, stand-alone Zope system can be illustrated in Figure 11-5.

Zope connected to a filestorage

Figure 11-5 Zope connected to a filestorage Comment

You can see there is one Zope application which plugs into a FileStorage. This storage, as its name implies, saves all of its information to a file on the computer's filesystem.

When using ZEO, you simple replace the FileStorage with a ClientStorage, as illustrated in Figure 11-6.

Zope with a Client Storage and Storage server

Figure 11-6 Zope with a Client Storage and Storage server Comment

Instead of saving objects to a file, a ClientStorage sends objects over a network connection to a Storage Server. As you can see in the illustration, the Storage Server uses a FileStorage to save that information to a file on the ZEO server's filesystem.

Storages are interchangeable and easy to implement. Because of their interchangeable nature, ZEO Storage Servers can use ZEO ClientStorages to pass on object data to yet another ZEO Storage Server. This is illustrated in Figure 11-7.

Multi-tiered ZEO system

Figure 11-7 Multi-tiered ZEO system

Here, you can see a number of ZEO clients funnel down through three ZEO servers, which in turn act as ZEO clients themselves and funnel down into the final, central ZEO server than saves its information in a FileStorage. Now, that central ZEO server is the single point of failure in the system. If any of your other clients, or intermediate servers fail, the system will still continue to work, but if the central server fails, then you need an alternative.

Using fake you can have a back-up storage server strategy, but this method is not very well proven and hasn't been explored by the authors. In the future, ZEO will have a "multiple-server" feature, that allows a group of storage servers to act as a quorum, so if one or more storage servers fail, the remaining servers in the quorum can continue to serve objects.

Es gibt zahlreiche Vorteile für ein derartiges Vorgehen, insbesondere falls Sie eine stark über das Netz verteilte Objektdatenbank aufbauen wollen. Wie bei allen Systemen mit Vorteilen, gibt es auch hier einige Nachteile. Diese Nachteile werden im folgenden Abschnitt untersucht.

ZEO Fallstricke

Größtenteils unterscheidet sich der Betrieb mit ZEO nicht vom Betrieb eines normalenZopeservers, aber es gibt doch einige Dinge zu beachten.

Zum einem dauert es länger bis Informationen in die Zope object Datenbank weggeschrieben werden. Dies verlangsamt zwar die Benutzung eines Zope nicht (weil Zope diese Operation nebenläufig im Hintergrund erledigt), aber es erhöht die Wahrscheinlichkeit einen "ConflictError" zu erhalten. Ein derartiger "ConflictError" passiert wenn beispielsweise zwei ZEO-Clients gleichzeitig auf ein Object schreibend zugreifen wollen. Einer der beiden ZEO Clients gewinnt diesen "Konfikt" dann und kann ganz normal weiterarbeiten. Der andere ZEO-Client hingegen verliert diesen "Konfikt" und muss seinen Schreibvorgang wiederholen.

Derartike Konflikte sollten so selten wie möglich vorkommen, da sie das System ausbremsen können. Während es ganz normal sein kann, hin und wieder einen derartiken Konflikt zu haben (aufgrund der vielen nebenläufigen Aufgaben, die ein Zope bearbeitet), ist es nicht wünschenswerte eine Menge dieser Fehlerfehler zu haben. Ein schlimmer Fall wäre beispielsweise ein ZEO Client, der in sehr kurzen Intervallen wiederholt versucht, auf ein Objekt schreibend zuzugreifen. In diesem Fall wird es eine Anhäufung von "ConflictErrors" geben und infolgedessen eine große Menge Wiederholungsversuche. Wenn ein ZEO client dreimal erfolglos versucht hat in die Datenbank zu schreiben, weil er jedesmal einen "ConflictError" ausglöst hat, dann wir der Schreibvorgang dieses ZEO-Clients nicht geschrieben und die Daten werden nicht geschrieben.

Weil es bei Verwendung von ZEO länger dauert bis Daten geschrieben werden, sind die Chancen einen "ConflictError" zu bekommen auch höher als wenn man ein Zope ohne ZEO betreiben würde. Aus diesem Grunde ist es ein ZOPE mit ZEO bei Schreibvorgängen fehleranfälliger. Dieses Problem sollten Sie beim Design Ihrer Netzwerkes oder Ihrer Anwendung im Blickfeld behalten. Als eine Daumenregel gilt: die Wahrscheinlichkeit einen "ConflictError" zu bekommen erhöht sich mit zunehmender Anzahl regelmässiger Schreibvorg&aum;nge. Auf der einen Seite verringern schnellere und zuverlässigere Netzwerkverbindungen und Computer die Chancen einen "ConflictError" zu bekommen. Auf der anderen Seite hingegen können die meisten der Probleme mit "ConflictErrors" vermieden werden, wenn man diese beide Faktoren berücksichtigt.

Es ist noch wichtig zu erwähnen, daß zwischen ZEO-Server und ZEO-Client zur Zeitpunkt der Erstellung dieses Dokumentes keine abhörsichere Verbindungen aufgebaut werden, daher der komplette Datenaustausch geschieht ungeschützt über das Netzwerk. Aus diesem Grunde solltet Ihr das ZEO-System nicht einfach ohne Sicherungsmassnahmen ins &oum;ffentlich zugreifbare Internet stellen. Falls Ihr doch auf diese dumme Idee kommen solltet, braucht Ihr natuer;ich nicht zu wundern, wenn irgendjemand irgendwelche Daten in euren ZEO-Server schreibt!!! - Dies kann schon sehr gravierende Auswirkungen auf euren Serverbetrieb haben !!!

Schliesslich sei noch erwähnt, daß es zwischen ZEO-Server und ZEO-Client keine verschlüsselten, und daher abhörsicheren Verbindungen gibt. Dies ist kein unlösbares Problem, da Du ja auch andere Werkzeuge wie Firewalls dazu einsetzen kannst, um Deine ZEO-Server abzuschirmen. Wenn Du hingegen keine Möglichkeit siehst, eine Firewall einzusetzten, Du aber trotzden den Datenaustausch zwischen den ZEO-Systemen über ein unsicheres Netzwerk, wie z.B. das öffentliche Internet-System gewährleisten willst, können wir Dir nur wärmstens den Einsatz von Programmen wie OpenSSH und stunnel empfehlen.

Diese ermöglichen einen verschlüselten Datenverkehr zwischen ZEO Server und den ZEO Clients. Wie diese Programme zu installieren und einzurichten sind werden wir an dieser Stelle nicht beschreiben - die wüde den Rahmen dieses Handbuches zu sehr strapazieren. Es sei nur auf die gute Dokumentation auf den Herstellerseiten der Programme im Internet verwiesen. Wenn Ihr wissen wollt wie eine Firewall funktioniert und einzurichten ist, können wir euch das Buch "Linux Firewalls" von Robert Ziegler empfehlen, welches vom Verlag "New Riders" herausgegeben wurde. (Anmerkung des Übersetzers: lieber mal eine deutsches Buch empfehlen ?)

Schlussbemerkungen

In diesem Kapitel haben wir einen Blick auf ZEO geworfen, und haben dabei erfahren wie ZEO die Kapazitäten eines Webauftrittes verbesser kann. Ausserdem haben wir nicht nur ZEO auf einem Rechner zum Laufen gebracht um und mit dessen Funktionsweise vertraut zu machen, sondern wir haben ZEO auch auf mehreren Rechnern zum Einsatz gebracht. Weiterhin haben wir uns mit verschiedenen Lastverteilungsverfahren beschäftigt, mit denen wir die Last unsere Webseitenbesucher auf mehrere Rechner zu verteilen.

ZEO is nicht die Antwort auf alle Probleme, und wie jedes andere System, welches mit vielen Rechnern im Verbund arbeitet, wird eine zusätliche Komplexität hinter den Kullisen der Website aufgebaut. Die Komplexität wird besonders dann stark spürbar, wenn viele dynamisch erzeugte Webseiten an das Publikum ausgeliefert werden sollen.

Zopebuch: Inhaltsverzeichnis

Dieses Dokument ist momentan noch nicht vollständig übersetzt.

Übersetzung von Lucia

Kapitel 14: Zope erweitern

Sie können Zope durch das Erstellen eigener Arten von Objekten erweitern und auf Ihre eigenen Bedürfnisse anpassen. Neue Arten von Objekten werden in Zope durch *Products* (Produkte) installiert . Produkte sind Erweiterungen von Zope, die von der Zope-Gemeinde und vielen anderen Drittherstellern entwickelt werden. Es gibt hunderte verschiedene Produkte, viele dienen sehr speziellen Anwendungen. Eine komplette Liste an Produkten findet man unter "Download" bei Zope.org.

Produkte können auf zwei verschiedene Arten entwickelt werden, *über das Web* durch Benutzung von ZClasses (Z-Klassen) und durch die Programmiersprache Python. Produkte können sogar eine Mischung von beidem, den Web-Produkten und Python Code, sein. Dieses Kapitel beschäftigt sich mit der Erstellung neuer Produkte über das Web, ein Thema das wir schon kurz in Kapitel 11, "Suchen und Katalogisieren von Inhalt" angeschnitten haben. Ein Produkt komplett in Python zu programmieren wäre hier außerhalb des gesteckten Zieles, für spezielle Dokumentation zur Produkt-Entwicklung besuchen Sie deshalb bitte Zope.org.

Dieses Kapitel zeigt Ihnen wie Sie :

- neue Produkte erzeugen können
- ??? ZClasses in Produkten definieren ???Define ZClasses in Products

- Python in ZClasses integrieren
- Produkte anderen Zope-Benutzern zugänglich machen

Im nächsten Abschnitt werden wir erste Schritte im Anpassen von Zope an Ihre Bedürfnisse machen. Dort lernen Sie, wie neue Zope-Produkte erzeugt werden.

Zope Produkte erstellen

Produkte werden bei ZOPE im *Product Management* folder (Produkt Management-Ordner) im "Control Panel" (Systemsteuerung) gespeichert. Klicken Sie auf das *Control_Panel* im Stammordner und klicken Sie dann *Products*. Sie sind nun auf dem Bildschirm, der unter Abbildung 12-1 gezeigt wird.

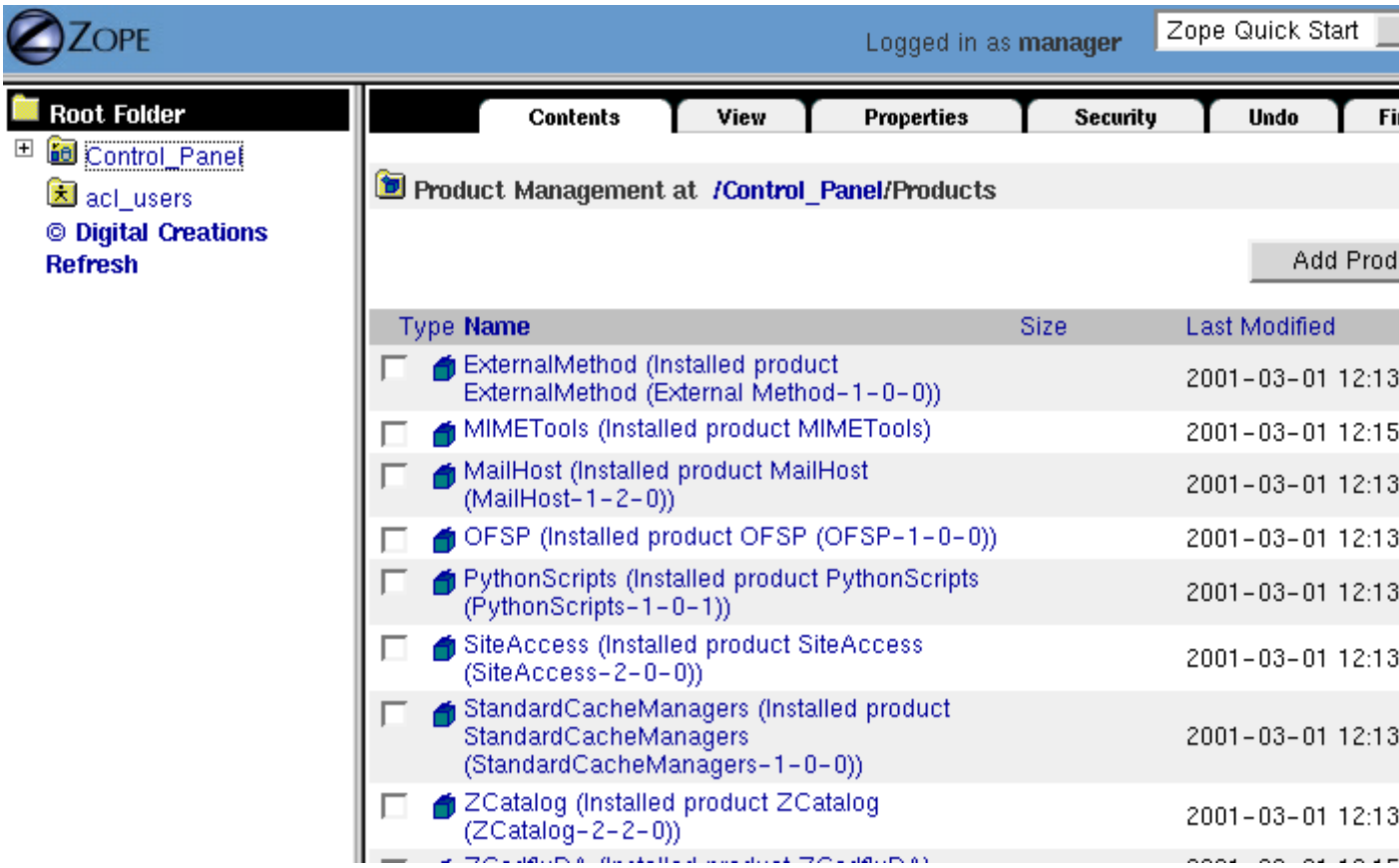


Abbildung 12-1 Installierte Produkte ??? Installed Products

Jedes blaue Päckchen zeigt ein installiertes Produkt an. Von diesem Bildschirm aus können Sie diese Produkte verwalten. Einige Produkte sind standardmäßig in Zope eingebunden oder wurden von Ihnen oder Ihrem Administrator installiert. Diese Produkte haben ein Symbol, das ein *geschlossenes* Päckchen anzeigt. Siehe hierzu Abbildung 12-1. Produkte mit einem geschlossenen Päckchen können nicht über das Web-Interface bearbeitet werden. Sie können durch klicken auf diese Produkte Informationen erhalten, sie aber nicht verändern.

Sie können außerdem Ihre eigegen Produkte erzeugen, die sie dann auch über das Web verändern *können*. Produkte lassen Sie neue Arten von Objekten in Zope erzeugen. Diese durch das Web verwaltbaren Produkte haben Symbole, die ein offenes Päckchen anzeigen. Wenn Sie den Beispielen in Kapitel 11 "Inhalt durchsuchen ud kathegorisieren" gefolgt sind, dann haben Sie nun ein *News/Neuigkeiten* Produkt mit offenem Päckchen.

Wieso wollen wir eigentlich Produkte erzeugen? Nehmen wir an, alle Betreuer des Zoos wollen einen unkomplizierten Weg, um einfache Online-Ausstellungen über den Zoo zu erzeugen. Die Ausstellungen müssen alle dasselbe Format haben und eine gleiche Informations-Struktur haben. Außerdem soll jede Ausstellung einem speziellen Tier zugeordnet sein.

Um dieses zu erreichen, könnten Sie eine Ausstellung zu einem Tier erstellen und diese dann für jede weitere Ausstellung kopieren und einfügen. Dies wäre allerdings ein schwieriger und manueller Prozess, da alle Informationen und Eigenschaften für jede neue Ausstellung geändert werden müsste. Außerdem gibt es vielleicht einmal tausende von Ausstellungen.

Um diese Problemstellung weiter auszubauen, gehen wir davon aus, daß Sie zu jeder Ausstellung die Information haben möchten, ob sich das Tier vom Aussterben bedroht ist oder nicht. Mit "kopieren und einfügen" würden Sie jede Ausstellung einzeln ändern müssen. "Kopieren und Einfügen" würde sich also eindeutig nicht für einen großen Zoo anwenden lassen und dies würde ziemlich teuer werden.

Sie müssen außerdem sicherstellen, daß jede Ausstellung einfach zu handhaben ist. Die Betreuer der einzelnen Ausstellungen sollten diejenigen sein, die die Informationen zur Verfügung stellen. Keiner der Betreuer hat aber eine große Ahnung von Zope oder wie man Webseiten gestaltet. Sicherlich werden Sie nicht deren Zeit verschwenden wollen, um dies zu erlernen. Die Betreuer sollen nichts weiter zu tun brauchen als Informationen über Ihre Interessen in ein einfaches Formular einzutragen, "veröffentlichen" zu klicken und wieder weggehen zu können.

Mit der Erzeugung eines Zope Produkts erreichen Sie dieses Ziel einfach und schnell. Sie können einfach handhabbare Objekte erzeugen, die Ihre Betreuer benutzen können. Sie können Seitenvorlagen für Ausstellungen definieren, deren einmalige Änderung alle Ausstellungsseiten ??? verändert / betrifft. All diese Dinge können Sie durch die Erstellung von Produkten tun.

Ein einfaches Produkt erzeugen

Durch die Benutzung von Produkten können Sie das Problem der Erstellung und Verwaltung von Ausstellungen lösen. Lassen Sie uns mit dem Beispiel, wie man ein einfaches Produkt erstellt. beginnen. Es soll uns ermöglichen, Informationen über die Ausstellungen zu sammeln und eine maßgeschneiderte Ausstellung zu erzeugen. Weiter hinten im Kapitel werden Sie komplexere und leistungsfähigere Wege zur Benutzung von Produkten kennenlernen.

??? Der größte Wert eines Zope Produktes liegt darin, daß es erlaubt Objekte an einem zentralen Platz zu erzeugen und man über die "Product Add" Liste Zugang zu den Objekten hat. Dies gibt Ihnen die Möglichkeit, globale Dienste zu erstellen und diese mittels eines Standard Teiles??? des Zope Management-Interfaces zugänglich zu machen. This gives you the ability to build global services and make them available via a standard part of the Zope

management interface. Mit anderen Worten: Produkte erlauben Ihnen, Zope Ihren Wünschen anzupassen.

Beginnen Sie, indem Sie zum *Products* Ordner im *Control Panel*gehen. Klicken Sie auf den *Add Product* Knopf im *Product Management* Ordner, um ein neues Produkt zu erzeugen. Dies wird sie zum "Produkt hinzufügen" (Product add) Formular bringen. Geben Sie die id "ZooAusstellung" ein und klicken Sie *Generate*. Sie werden nun Ihr neues Produkt im *Product Management* Ordner sehen.Es sollte ein geöffnetes blaues Päckchen zu sehen sein. Der geöffnete Deckel zeigt an, daß Sie das Produkt anklicken und durch das Web administrieren können.

Wählen Sie das *ZooAusstellung* Produkt aus. Dadurch werden Sie auf den Produkt Management Bildschirm gelangen.

Bis auf einige wenige Unterschiede sieht der Management-Bildschirm für ein Produkt genauso aus wie ein Ordner und er verhält sich auch so:

1. Es gibt eine neue Ansicht, die sich *Distribution* nennt, zu sehen ganz auf der rechten Seite. Diese gibt Ihnen die Möglichkeit, Ihr Produkt zu packen und an andere weiterzugeben. Dieser Punkt wird später abgehandelt.
2. Wenn Sie die Add-Liste anklicken, werden Sie einige neue Typen von Objekten finden, die Sie zufügen können, einbegriffen *ZClass*, *Factory* und *Permission*.
3. Der Ordner, der mit einem Fragezeichen versehen st, ist der Hilfe Ordner zum Produkt *ZooAusstellung* Dieser Ordner kann *Hilfe-Themen* beinhalten, die anderen die Benutzung des Produktes erklären.
4. Es gibt außerdem eine neue Ansicht namens *Define Permissions*, die die Benutzerrechte zu diesem Produkt definieren. Diese Ansicht ist momentan nicht wichtig für unser Beispiel, da es sich um eine fortgeschrittene Anwendung handelt

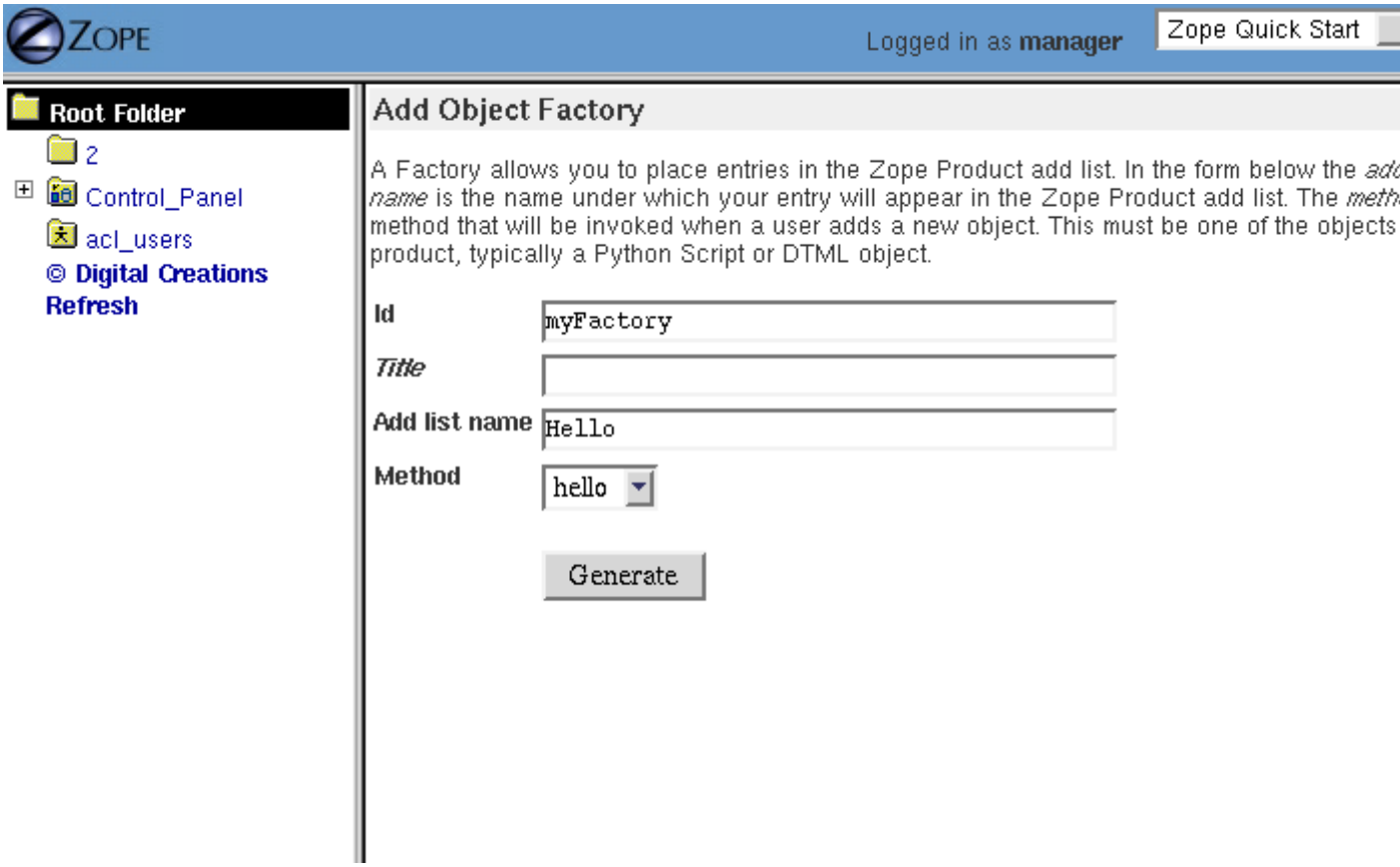
Erzeugen Sie in der *Contents* Ansicht eine DTML Methode namens *Hallo* mit diesem Inhalt:

```
<dtml-var standard_html_header>

<h2>Hallo vom Zoo Ausstellungs Produkt</h2>

<dtml-var standard_html_footer>
```

Diese Methode erlaubt Ihnen, Ihr Produkt zu testen. Als nächstes erzeugen wir eine Fabrik (engl. Factory). Wählen Sie aus der Produktauswahlliste *Zope Factory* aus. Sie gelangen dann zu einem Formular zur Erzeugung einer Fabrik, wie in Bild 12-2 gezeigt.



Fabriken stellen eine Brücke zwischen Ihrer Produktauswahlliste und Ihrem Produkt her. Geben Sie Ihrer Fabrik die ID *meineFabrik*. In das *Add list name* Feld geben Sie *Hallo* und in der Auswahl *Method* wählen Sie *hallo*. Klicken sie nun *Generate*, danach klicken Sie auf die neue Fabrik und ändern die *Permission* auf *Add Document, Images, and Files* und bestätigen diese Änderungen mit *Save Changes*. Diese Änderung teilt Zope mit, daß Sie die Rechte zum *Hinzufügen von Dokumenten, Bildern und Dateien* haben müssen, um die Fabrik benutzen zu können. Herzlichen Glückwunsch, Sie haben soeben das Zope-Management-Interface Ihren Wünschen angepasst. Gehen Sie zum Stammordner und klicken Sie die Produktauswahlliste an. Sie werden feststellen, daß diese nun einen Eintrag namens *Hallo* enthält. Wählen Sie *Hallo* von der Produktauswahlliste aus und Ihre *hallo* Methode wird aufgerufen.

Eine der üblichsten Anwendungen die man mit Fabriken verbindet, ist das Kopieren von Objekten in den aktuellen Ordner. Mit anderen Worten: Die Methoden können Zugang zum Ort von welchem sie aufgerufen werden erlangen und dort dann Arbeiten in diesem Ordner ausführen, inklusive Objekte in ihn zu kopieren ??? In other words your methods can get access to the location from which they were called and can then perform operations on that Folder including copy objects into it.??? Nur weil Sie alle möglichen verrückten Sachen mit Fabriken und Produkten anstellen können, sollten Sie diese übrigens nicht tun. Im Allgemeinen geht ein Anwender davon aus, daß er, wenn er in der Produktauswahlliste ein Produkt ausgewählt hatte, zu einem Erstellformular gelangt, in dem er die ID und ein neues Objekt festlegt. Als nächstes erwartet er, daß ein neues Objekt mit der zuvor festgelegten ID erstellt wird wenn er *hinzufügen (Add)* gewählt hatte. Schauen wir nun, wie wir diesen Erwartungen gerecht werden.

Zuerst erstellen wir einen neuen Ordner mit dem Namen *AusstellungsVorlage* in Ihrem Produkt Ordner (ZooAusstellung). Dieser wird als Vorlage für Ihre Ausstellungen dienen. Nun erstellen wir ebenfalls im Produkt Ordner eine DTML Methode namens *Erstellformular* und ein Python Skript namens *erstellen* Diese Objekte werden neue Instanzen im Ausstellungsprojekt erzeugen. Gehen Sie jetzt zu Ihrer Fabrik zurück und ändern Sie sie, so daß bei *Add list name ZooAusstellung* steht und die Methode *Erstellformular* ist.

Wenn nun jemand *Zooausstellung* von der Produktauswahlliste wählt, wird die Methode *Erstellformular*durchgeführt. Diese Methode soll Informationen über die ID und den Titel der Ausstellung sammeln. Wenn der Benutzer *hinzufügen* klickt, soll das Skript *erstellen* aufgerufen werden welches den Ordner *AusstellungsVorlage* in den aufrufenden Ordner kopiert und ihm die angegebene ID gibt. Der nächste Schritt ist, das *Erstellformular* mit diesem Inhalt zu editieren:

```
<dtml-var manage_page_header>

        <h2>Fügen Sie eine Zoo Ausstellung hinzu</h2>

        <form action="add" method="post">
        id <input type="text" name="id"><br>
        title <input type="text" name="title"><br>
        <input type="submit" value=" Add ">
        </form>

      <dtml-var manage_page_footer>
```

Zugegebenermaßen ist dies ein eher kahles Erstellformular. Es sammelt kaum Daten und es sagt dem Benutzer nicht, was eine Zoo Ausstellung ist und wieso er welche hinzfügen sollte. Wenn Sie Ihre eigenen Webanwendung gestalten, werden Sie mehr tun wollen als in diesem Beispiel.

Beachten Sie bitte, daß diese Methode keine Standard HTML Kopf- und Fußzeilen enthält. Es ist eine Zope-Konvention daß Zope Management-Bildschirme nicht dieselben Kopf- und Fußzeilen wie Ihre herkömmlichen Seite benutzen. Stattdessen benutzen Management-Bildschirme `manage_page_header` und `manage_page_footer`. ???? Die Kopf- und Fußzeilen der Management-Ansicht stellen sicher, daß diese ein einheitliches Aussehen und eine einheitliche Benutzung haben.

Außerdem stellen Sie fest, dass die Action für das Formular das *erstellen* Skript ist. Fügen Sie nun den folgenden Inhalt in das *erstellen* Skript ein:

```
      ## Script (Python) "add"
      ##parameters=id ,title, REQUEST=None
      ##
      """
      Kopieren Sie die AusstellungsVorlage in den aufrufenden Ordner
      """

      # Duplizieren Sie die Vorlage und geben Sie ihm die neue ID Dies wird
in den
      # aktuellen Kontext plaziert (der Platz von dem die Fabrik aus
aufgerufen wurde).
      exhibit=context.manage_clone(container.exhibitTemplate,id)
```

```
      # Ändern des Titel des Duplikates
      exhibit.manage_changeProperties(title=title)

      # Wenn durch das Web aufgerufen wurde, zurück zum Kontext umleiten.
???If we were called through the web, redirect back to the context
      if REQUEST is not None:
          try: u=context.DestinationURL()
          except: u=REQUEST['URL1']
          REQUEST.RESPONSE.redirect(u+'/manage_main?update_menu=1')
```

Dieses Skript vervielfältigt die *AusstellungsVorlage* und kopiert sie mit der angegebenen ID zum aktuellen Ordner. Dann ändert es die Eigenschaften des *Titels* der neuen Ausstellung. Am Ende gibt es den Management Bildschirm des aktuellen Ordners zurück indem es *manage_main* aufruft. ??? Finally it returns the current folder's main management screen by calling *manage_main*.

Herzlichen Glückwunsch, Sie haben soeben Zope durch die Erstellung eines neuen Produktes erweitert. Sie haben einen Weg geschaffen, Objekte mithilfe der Produktauswahlliste in Zope zu kopieren. Trotzdem leidet diese Lösung noch unter einem Problem, das wir schon früher in diesem Kapitel anbgeschnitten hatten. Obwohl Sie die AusstellungsVorlage an einem zentralen Ort editieren können, ist es trotzdem noch immer nur eine Vorlage. Wenn Sie also eine neue Eigenschaft zur Vorlage hinzufügen, wird das keine der bereits existierenden Ausstellungen beeinflussen. Um existierende Ausstellungen zu ändern, müssen Sie diese von Hand ändern.

ZKlassen (ZClasses) bringen Sie einen Schritt weiter, indem sie Ihnen eine zentrale Vorlage ermöglichen, die einen neuen Typen von Objekt definieren. Wenn Sie diese Vorlage dann ändern, ändern sich gleichzeitig auch automatisch alle Objekte dieses Types. Diese zentrale Vorlage nennt man ZKlasse (ZClass). Im nächsten Abschnitt werden Sie lernen, wie eine ZKlasse erstellt wird, die eine neue ZKlasse namens *Ausstellung* definiert.

ZKlassen (ZClasses) erzeugen

ZKlassen sind Werkzeuge die Ihnen helfen, neue Typen von Objekten zu bilden, indem sie eine *Klasse* definieren. Eine Klasse ist wie ein Entwurfsplan für Objekte. Wenn Sie eine Klasse definieren, legen Sie fest, wie ein Objekt bei seiner Erstellung sein wird. EIne Klasse kann Methoden, Eigenschaften und andere Attribute definieren.

Objekte, die Sie über eine bestimmte Klasse erstellen, werden *Instanzen* dieser Klasse genannt. So gibt es beispielsweise nur eine Klasse namens *Ordner*, Sie können aber mehrere Ordner-Instanzen in Ihrer Anwendung haben.

Instanzen besitzen dieselben Methoden und Eigenschaften wie ihre Klasse. Wenn Sie die Klasse ändern, werden alle Instanzen diese Änderung ausgeben/reflektieren??? Anders als die Vorlagen, die wir im letzten Abschnitt erstellt haben, übernehmen Klassen fortlaufend die Kontrolle über Instanzen. Beachten Sie aber bitte, daß diese Kontrolle nur in eine Richtung funktioniert: Wenn Sie Änderung in einer Instanz machen, beeinflussen diese nicht die Klasse oder andere Instanzen.

ZKlassen entsprechen den Formatvorlagen in Textverarbeitungsprogrammen. Die meisten Textverarbeitungen liefern ein Set an vordefinierten Vorlagen, die man zum Erstellen einer bestimmten Art von Dokument benutzen kann, so zum Beispiel zur Erstellung eines Lebenslaufes. Es mag vielleicht Hunderttausende von Lebensläufen geben, die auf der Lebenslauf-Vorlage von Microsoft basieren, aber es gibt dafür nur eine Vorlage. Ein Zope-Objekt verhält sich zu einer ZKlasse wie ein einzelner Lebenslauf zu einer Formatvorlage für Lebensläufe.

ZKlasse können Sie durch Benutzung der Zope Management-Oberfläche über das Web erstellen. Klassen können auch in Python geschrieben werden, aber dies wird in diesem Buch nicht behandelt.

ZKlassen können Attribute von anderen Klassen erben. Diese Vererbung erlaubt es Ihnen, eine neue Klasse zu definieren, die auf einer anderen Klasse basiert. Gehen wir beispielsweise davon aus, Sie möchten eine neue Art von Dokument mit speziellen für Sie interessanten Eigenschaften erstellen. Statt alle Funktionalität dieses Dokumentes von Grund auf neu zu erstellen, können Sie diese einfach von der *DTML Dokument*-Klasse *erben* und nur die neuen Informationen an denen Sie interessiert sind, hinzufügen.

Mithilfe von Vererbung können Sie auch verallgemeinernde Beziehungen zwischen Klassen erstellen. Zum Beispiel könnten Sie eine Klasse namens `Tier` erstellen, die Informationen beinhaltet, die allenTiere gemeinsam sind. Dann bilden Sie die Klassen *Reptil* und *Säugetier* die sich beide von der Klasse *Tier* vererben. Um dies noch weiter fortzuführen, könnten Sie mithilfe von Vererbung zwei zusätzliche Klassen *Eidechse* und *Schlange* bilden, wie in Abbildung 12-3 gezeigt.

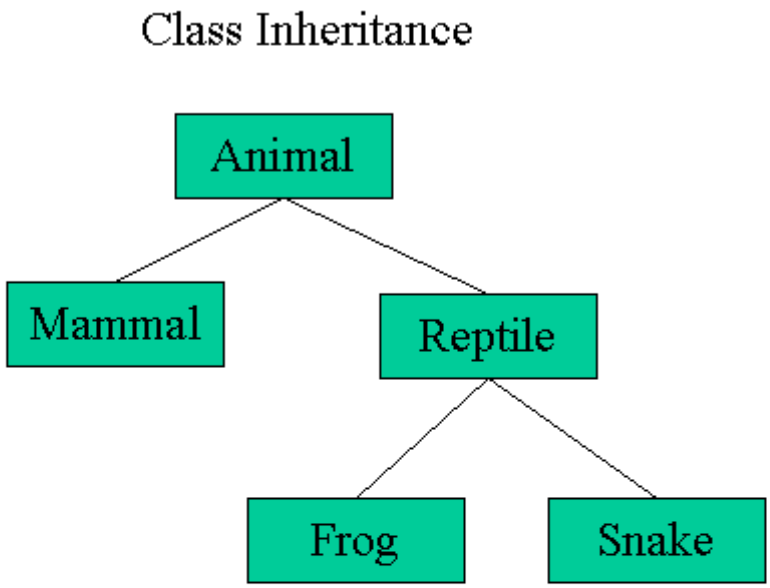


Figure 12-3 Beispiel für Vererbung in Klassen

ZKlassen können von den meisten Klassen, die wir in diesem Buch verwenden, erben. Zusätzlich können ZKlassen von anderen ZKlassen im selben Produkt erben. Wir werden diese und andere Techniken in diesem Kapitel anwenden.

Bevor wir mit dem nächsten Beispiel weitermachen, sollten Sie das existierende Produkt *ZooAusstellung* in Ihrem Zope Produkt Ordner umbenennen, zum Beispiel zu *ZooVorlage*, damit dieses nicht mit diesem Beispiel in Konflikt gerät. Erstellen Sie nun ein neues Produkt im Produkt Ordner namens *ZooAusstellung*.

Wählen Sie nun *ZClass* aus der Auswahlliste in der Inhaltsansicht der *ZooAusstellung* aus. Dieses Formular ist komplex und hat jede Menge Elemente. Wir werden diese Schritt für Schritt durchgehen:

Id
: Dies ist der Name der zu erzeugenden Klasse. Wählen Sie für dieses Beispiel den Namen *ZooAusstellung*.

Meta Type

Der Meta Typ eines Objektes ist ein Name für den Typ des Objektes. Dies sollte ein kurzer Begriff, der aussagt, was das Objekt tut, sein. Wählen Sie hier den Meta Typen "ZooAusstellung".

Base Classes

Base classes / Basis Klassen definieren eine Sequenz an Klassen von denen Sie Attribute vererben wollen. Ihre neue Klasse kann als ??? ss to inherit attributes from. Your new class can be thought of as *extending* or being *derived from* the functionality of your base classes. ??? Sie können eine oder mehrere Klassen von der Liste auf der linken Seite auswählen und durch drücken des Knopfes -> in Ihre Liste der Basis Klassen einfügen. Der Knopf <- entfernt die Basis Klassen, die Sie rechts ausgewählt hatten. Wählen Sie momentan keine Basis Klasse aus. Später werden wir auif einige der interessanteren Basis Klassen, wie z.B. *ObjectManager* eingehen.

Create constructor objects?

Sie werden diese Option normalerweise angekreuzt lassen, es sei denn Sie möchten form/action Konstruktor Paare und ein Fabrik Objekt selbst herstellen. Wenn Sie möchten, daß Zope diese Aufgabe für Sie übernimmt, lassen Sie dieses Kästchen angekreuzt. Somit wird dieses Formular fünf Objekte erstellen: eine Klasse, eine Konstruktor Formular, eine Konstruktor Aktion, einmal Zugriffsrechte und eine Fabrik. Wir lassen also in diesem Beispiel die Box angekreuzt.

Include standard Zope persistent object base classes?

Diese Option sollte angekreuzt bleiben, es Sei denn Sie möchten nicht, daß Ihr Objekt in der Datenbank gespeichert wird. Dies ist eine Option für fortgeschrittene Benutzer. Lassen Sie dieses Kästchen angekreuzt. ??? This is an advanced option and should only be used for Pluggable Brains. ???

Klicken Sie nun *Add*. Danach kommen Sie automatisch zur Übersicht des Produktes *ZooAusstellung*. Sie werden nun fünf neue Objekte vorfinden, wie in Abbildung 12-4 gezeigt.

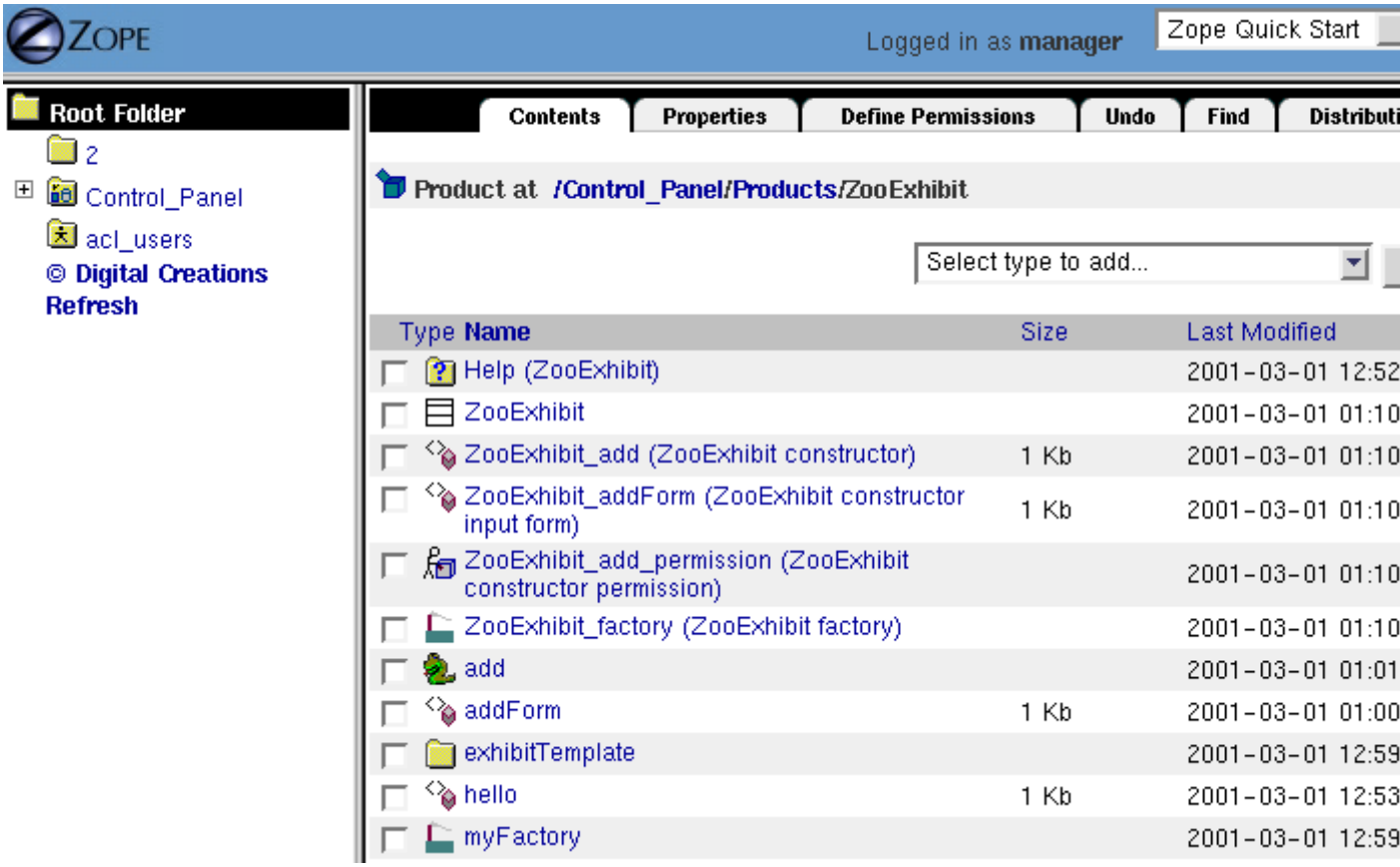


Figure 12-4 Produkt mit einer ZKlasse

Die fünf Objekte die Zope erstellt hat sind alle so konfiguriert, daß sie problemlos funktionieren, Sie brauchen diese momentan also nicht ändern. Hier eine kurz Beschreibung von jedem neu erstellten Objekt:

ZooAusstellung
: Die ist die ZKlasse selbst. Sein Symbol ist eine weisse Box mit zwei roten Linien. Dies ist das traditionelle Symbol für eine *Klasse*.

2003-03-03 Lucia
: This DTML Method is the constructor form for the ZClass. It is a simple form that accepts an id and title. You can customize this form to accept any kind of input your new object requires. The is very similar to the add form we created in the first example.

ZooExhibit_add
: This DTML Method gets called by the constructor form, *ZooExhibit_addForm*. This method actually creates your new object and sets its *id* and *title*. You can customize this form to do more advanced changes to your object based on input parameters from the *ZooExhibit_addForm*. This has the same functionality as the Python script we created in the previous example.

ZooExhibit_add_permission
: The curious looking stick-person carrying the blue box is a *Permission*. This defines a permission that you can associate with adding new *ZooExhibit* objects. This lets you protect the ability to add new Zoo exhibits. If you click on this Permission, you can see the name of this new permission is "Add ZooExhibits".

ZooExhibit_factory
: The little factory with a smokestack icon is a *Factory* object. If you click on this object, you can change the text that shows up in the add list for this object in the *Add list name* box. The *Method* is the method that gets called when a user selects the *Add list name* from the add list. This is usually the constructor form for your object, in this case, *ZooExhibit_addForm*. You can associate the Permission the user must have to add this object, in this case, *ZooExhibit_add_permission*. You can also specify a regular Zope permission instead.

That's it, you've created your first ZClass. Click on the new ZClass and click on its *Basic* tab. The *Basic* view on your ZClass lets you change some of the information you specified on the ZClass add form. You cannot change the base classes of a ZClass. As you learned earlier in the chapter, these settings include:

meta-type
: The name of your ZClass as it appears in the product add list.

class id
: A unique identifier for your class. You should only change this if you want to use your class definition for existing instances of another ZClass. In this case you should copy the class id of the old class into your new class.

icon
: The path to your class's icon image. There is little reason to change this. If you want to change your class's icon, upload a new file with the *Browse* button.

At this point, you can start creating new instances of the *ZooExhibit* ZClass. First though, you probably want a common place where all exhibits are defined, so go to your root folder and select *Folder* from the add list and create a new folder with the id "Exhibits". Now, click on the *Exhibits* folder you just created and pull down the Add list. As you can see, *ZooExhibit* is now in the add list.

Go ahead and select *ZooExhibit* from the add list and create a new Exhibit with the id "FangedRabbits". After creating the new exhibit, select it by clicking on it.

As you can see your object already has three views, *Undo*, *Ownership*, and *Security*. You don't have to define these parts of your object, Zope does that for you. In the next section, we'll add some more views for you to edit your object.

Creating Views of Your ZClass

All Zope objects are divided into logical screens called *Views*. Views are used commonly when you work with Zope objects in the management interface, the tabbed screens on all Zope objects are views. Some views like *Undo*, are standard and come with Zope.

Views are defined on the *Views* view of a ZClass. Go to your *ZooExhibit* ZClass and click on the *Views* tab. The *Views* view looks like Figure 12-5.

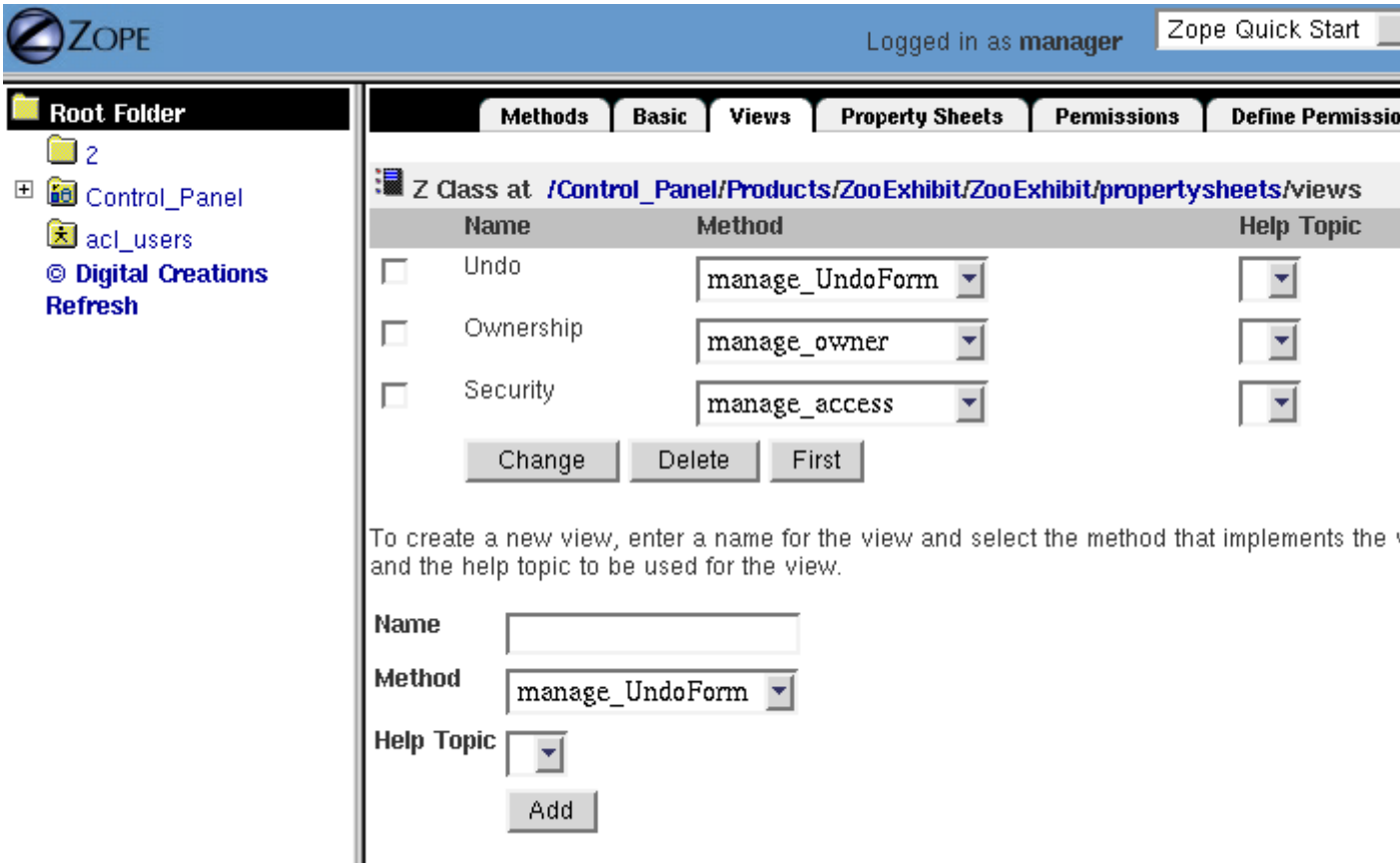


Figure 12-5 The Views view.

On this view you can see the three views that come automatically with your new object, *Undo*, *Ownership*, and *Security*. They are automatically configured for you as a convenience, since almost all objects have these interfaces, but you can change them or remove them from here if you really want to (you generally won't).

The table of views is broken into three columns, *Name*, *Method*, and *Help Topic*. The *Name* is the name of the view and is the label that gets drawn on the view's tab in the management interface. The *Method* is the method of the class or property sheet that gets called to render the view. The *Help Topic* is where you associate a *Help Topic* object with this view. Help Topics are explained more later.

Views also work with the security system to make sure users only see views on an object that they have permission to see. Security will be explained in detail a little further on, but it is good to know at this point that views now only divide an object management interfaces into logical chunks, but they also control who can see which view.

The *Method* column on the Methods view has select boxes that let you choose which method generates which view. The method associated with a view can be either an object in the *Methods* view, or a Property Sheet in the *Property Sheets* view.

Creating Properties on Your ZClass

Properties are collections of variables that your object uses to store information. A Zoo Exhibit object, for example, would need properties to contain information about the exhibit, like what animal is in the exhibit, a description, and who the caretakers are.

Properties for ZClasses work a little differently than properties on Zope objects. In ZClasses, Properties come in named groups called *Property Sheets*. A Property Sheet is a way of organizing a related set of properties together. Go to your *ZooExhibit* ZClass and click on the *Property Sheets* tab. To create a new sheet, click *Add Common Instance Property Sheet*. This will take you to the Property Sheet add form. Call your new Property Sheet "ExhibitProperties" and click *Add*.

Now you can see that your new sheet, *ExhibitProperties*, has been created in the *Property Sheets* view of your ZClass. Click on the new sheet to manage it, as shown in Figure 12-6.

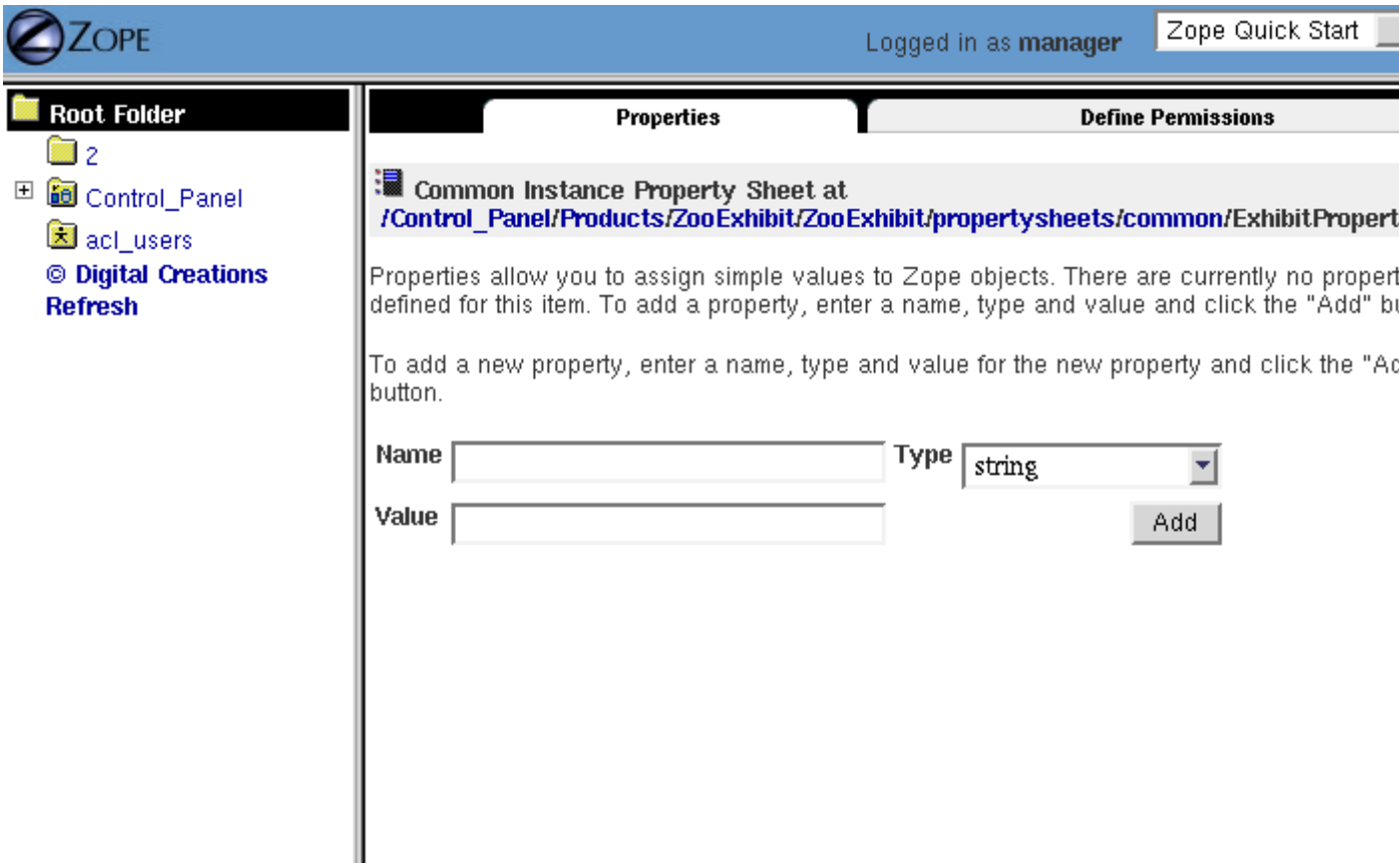


Figure 12-6 A Property Sheet

As you can see, this sheet looks very much like the *Properties* view on Zope objects. Here, you can create new properties on this sheet. Properties on Property Sheets are exactly like Properties on Zope objects, they have a name, a type, and a value.

Create three new properties on this sheet:

animal
: This property should be of type *string*. It will hold the name of the animal this exhibit features.

description
: This property should be of type *text*. It will hold the description of the exhibit.

caretakers
: This property should be of type *lines*. It will hold a list of names for the exhibit caretakers.

Property Sheets have two uses. As you've seen with this example, they are a tool for organizing related sets of properties about your objects, second to that, they are used to generate HTML forms and actions to edit those set of properties. The HTML edit forms are generated automatically for you, you only need to associate a view with a Property Sheet to see the sheet's edit form. For example, return to the ZooExhibit ZClass and click on the *Views* tab and create a new view with the name *Edit* and associate it with the method *propertysheets/ExhibitProperties/manage_edit*.

Since you can use Property Sheets to create editing screens you might want to create more than one Property Sheet for your class. By using more than one sheet you can control which properties are displayed together for editing purposes. You can also separate private from public properties on different sheets by associating them with different permissions.

Now, go back to your *Exhibits* folder and either look at an existing *ZooExhibit* instance or create a new one. As you can see, a new view called *Edit* has been added to your object, as shown in Figure Figure 12-7.

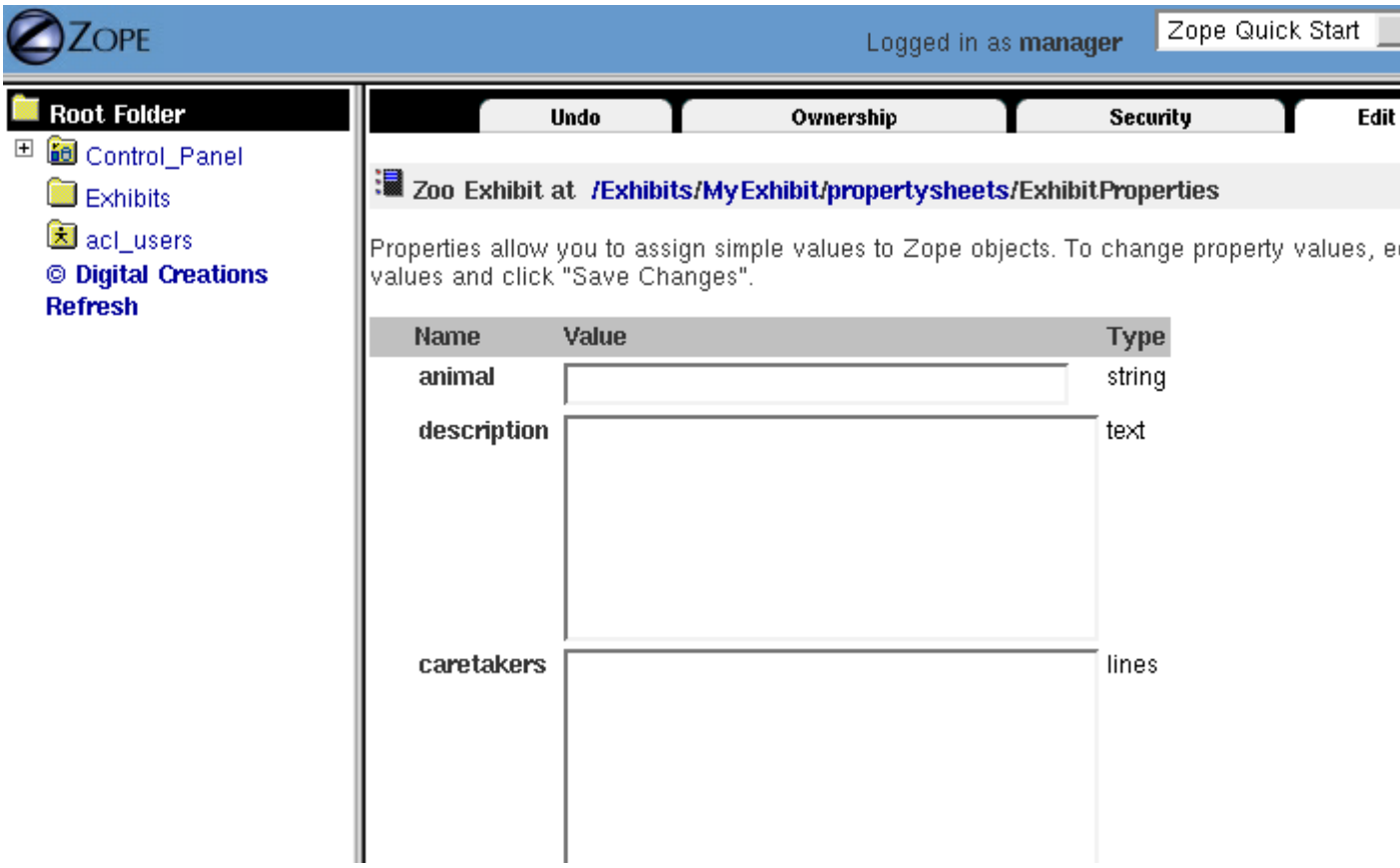


Figure 12-7 A ZooExhibit Edit view

This edit form has been generated for you automatically. You only needed to create the Property Sheet, and then associate that sheet with a View. If you add another property to the *ExhibitProperties* Property Sheet, all of your instances will automatically get a new updated edit form, because when you change a ZClass, all of the instances of that class inherit the change.

It is important to understand that changes made to the class are reflected by all of the instances, but changes to an instance are *not* reflected in the class or in any other instance. For example, on the *Edit* view for your *ZooExhibit* instance (*not* the class), enter "Fanged Rabbit" for the *animal* property, the description "Fanged, carnivorous rabbits plagued early medieval knights. They are known for their sharp, pointy teeth." and two caretakers, "Tim" and "Somebody Else". Now click *Save Changes*.

As you can see, your changes have obviously effected this instance, but what happened to the class? Go back to the *ZooExhibit* ZClass and look at the *ExhibitProperties* Property Sheet. Nothing has changed! Changes to instances have no effect on the class.

You can also provide default values for properties on a Property Sheet. You could, for example, enter the text "Describe your exhibit in this box" in the *description* property of the *ZooExhibit* ZClass. Now, go back to your *Exhibits* folder and create a *new* , *ZooExhibit* object and click on its *Edit* view. Here, you see that the value provided in the Property Sheet is the default value for the instance. Remember, if you change this instance, the default value of the property in the Property Sheet is *not* changed. Default values let you set up useful information in the ZClass for properties that can later be changed on an instance-by-instance basis.

You may want to go back to your ZClass and click on the *Views* tab and change the "Edit" view to be the first view by clicking the *First* button. Now, when you click on your instances, they will show the Edit view first.

Creating Methods on your ZClass

The *Methods* View of your ZClass lets you define the methods for the instances of your ZClass. Go to your *ZooExhibit* ZClass and click on the *Methods* tab. The *Methods* view looks like Figure 12-8.

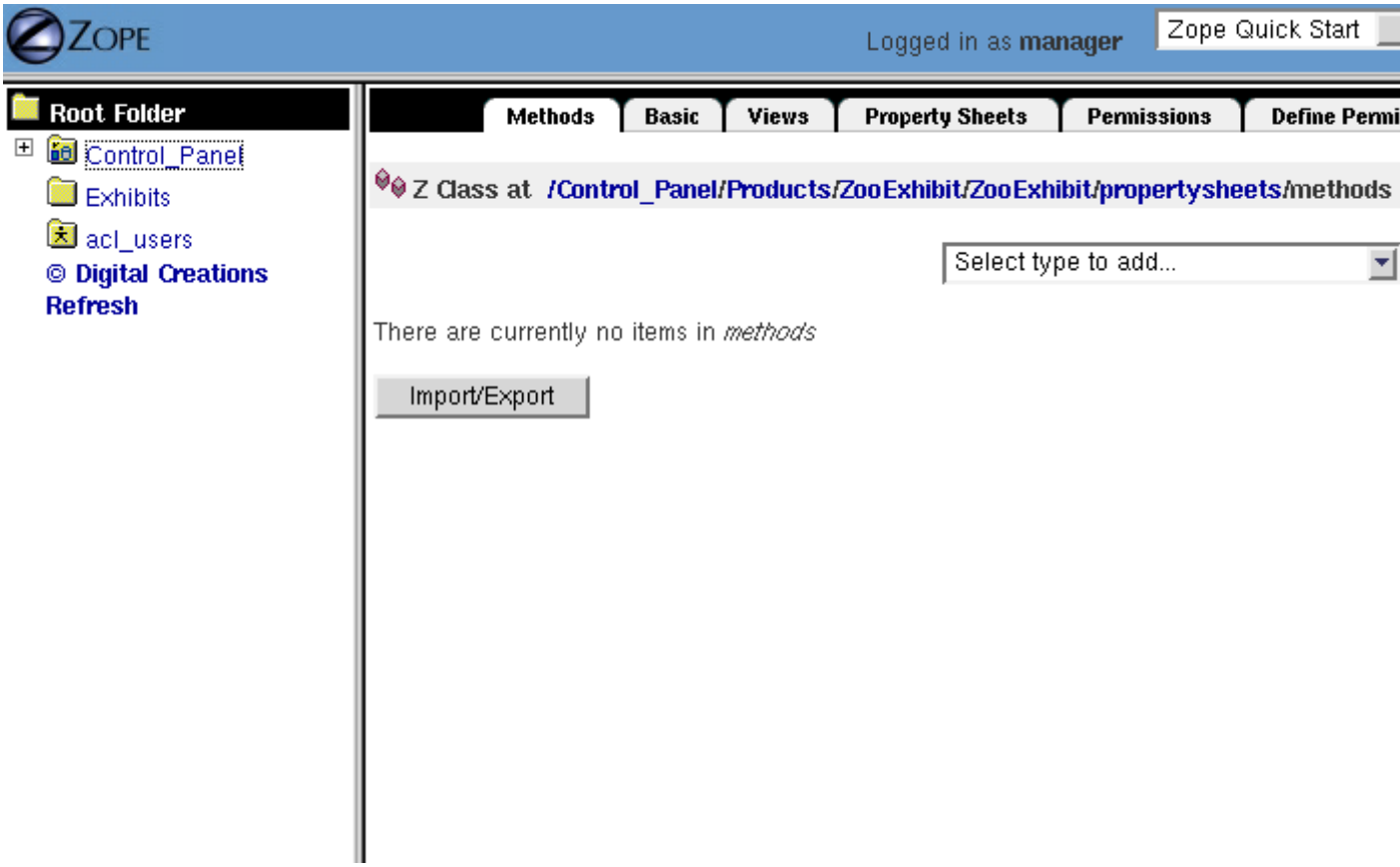


Figure 12-8 The Methods View

You can create any kind of Zope object on the *Methods* view, but generally only callable objects (DTML Methods and Scripts, for example) are added.

Methods are used for several purposes:

Presentation

When you associate a view with a method, the method is called when a user selects that view on an instance. For example, if you had a DTML Method called *showAnimalImages*, and a view called *Images*, you could associate the *showAnimalImages* method with the *Images* view. Whenever anyone clicked on the *Images* view on an instance of your ZClass, the *showAnimalImages* method would get called.

Logic

Methods are not necessarily associated with views. Methods are often created that define how you can work with your object.

For example, consider the *isHungry* method of the *ZooExhibit* ZClass defined later in this section. It does not define a view for a *ZooExhibit*, it just provide very specific information about the *ZooExhibit*. Methods in a ZClass can call each other just like any other Zope methods, so logic methods could be *used* from a presentation method, even though they don't *define* a view.

Shared Objects

As was pointed out earlier, you can create any kind of object on the *Methods* view of a ZClass. All instances of your ZClass will *share* the objects on the Methods view. For example, if you create a *Z Gadfly Connection* in the Methods view of your ZClass, then all instances of that class will share the same Gadfly connection. Shared objects can be useful to your class's logic or presentation methods.

A good example of a presentation method is a DTML Method that displays a Zoo Exhibit to your web site viewers. This is often called the *public interface* to an object and is usually associated with the *View* view found on most Zope objects.

Create a new DTML Method on the *Methods* tab of your *ZooExhibit* ZClass called *index_html*. Like all objects named *index_html*, this will be the default representation for the object it is defined in, namely, instances of your ZClass. Put the following DTML in the *index_html* Method you just created:

```
<dtml-var standard_html_header>

  <h1><dtml-var animal></h1>

  <p><dtml-var description></p>

  <p>The <dtml-var animal> caretakers are:<br>
    <dtml-in caretakers>
      <dtml-var sequence-item><br>
    </dtml-in>
  </p>

<dtml-var standard_html_footer>
```

Now, you can visit one of your *ZooExhibit* instances directly through the web, for example, *http://www.zopezoo.org/Exhibits/FangedRabbits/* will show you the public interface for the Fanged Rabbit exhibit.

You can use Python-based or Perl-based Scripts, and even Z SQL Methods to implement logic. Your logic objects can call each other, and can be called from your presentation methods. To create the *isHungry* method, first create two new properties in the *ExhibitProperties* property sheet named "last_meal_time" that is of the type *date* and "isDangerous" that is of the type *boolean*. This adds two new fields to your Edit view where you can enter the last time the animal was fed and select whether or not the animal is dangerous.

Here is an example of an implementation of the *isHungry* method in Python:

```
## Script (Python) "isHungry"
##
"""
Returns true if the animal hasn't eaten in over 8 hours
"""
from DateTime import DateTime
if (DateTime().timeTime()
    - container.last_meal_time.timeTime() >  60 * 60 * 8):
```

```
    return 1
else:
    return 0
```

The `container` of this method refers to the ZClass instance. So you can use the `container` in a ZClass instance in the same way as you use `self` in normal Python methods.

You could call this method from your *index_html* display method using this snippet of DTML:

```
<dtml-if isHungry>
  <p><dtml-var animal> is hungry</p>
</dtml-if>
```

You can even call a number of logic methods from your display methods. For example, you could improve the hunger display like so:

```
<dtml-if isHungry>

  <p><dtml-var animal> is hungry.

  <dtml-if isDangerous>

    <a href="notify_hunger">Tell</a> an authorized
    caretaker.

  <dtml-else>

    <a href="feed">Feed</a> the <dtml-var animal>.

  </dtml-if>

  </p>

</dtml-if>
```

Your display method now calls logic methods to decide what actions are appropriate and creates links to those actions. For more information on Properties, see Chapter 3, "Using Basic Zope Objects".

ObjectManager ZClasses

If you choose *ZClasses:ObjectManager* as a base class for your ZClass then instances of your class will be able to contain other Zope objects, just like Folders. Container classes are identical to other ZClasses with the exception that they have an addition view *Subobjects*.

From this view you can control what kinds of objects your instances can contain. For example if you created a FAQ container class, you might restrict it to holding Question and Answer objects. Select one or more meta-types from the select list and click the *Change* button. The *Objects should appear in folder lists* check box control whether or not instances of your container class are shown in the Navigator pane as expandable objects.

Container ZClasses can be very powerful. A very common pattern for web applications is to have two classes that work together. One class implements the basic behavior and hold data. The other class contains instances of the basic class and provides methods to organize and list the contained instances. You can model many problems this way, for example a ticket manager can contain problem tickets, or a document repository can contain documents, or an object router can contain routing rules, and so on. Typically the container class will provide methods to add, delete, and query or locate contained objects.

ZClass Security Controls

When building new types of objects, security can play an important role. For example, the following three Roles are needed in your Zoo:

Manager
: This role exists by default in Zope. This is you, and anyone else who you want to be able to completely manage your Zope system.

Caretaker
: After you create a *ZooExhibit* instance, you want users with the *Caretaker* role to be able to edit exhibits. Only users with this role should be able to see the *Edit* view of a *ZooExhibit* instance.

Anonymous
: This role exists by default in Zope. People with the *Anonymous* role should be able to view the exhibit, but not manage it or change it in any way.

As you learned in Chapter 7, "Users and Security", creating new roles is easy, but how can you control who can create and edit new *ZooExhibit* instances? To do this, you must define some security policies on the *ZooExhibit* ZClass that control access to the ZClass and its methods and property sheets.

Controlling access to Methods and Property Sheets

By default, Zope tries to be sensible about ZClasses and security. You may, however, want to control access to instances of your ZClass in special ways.

For example, Zoo Caretakers are really only interested in seeing the *Edit* view (and perhaps the *Undo* view, which we'll show later), but definitely not the *Security* or *Ownership* views. You don't want Zoo caretakers changing the security settings on your Exhibits; you don't even want them to *see* those aspects of an Exhibit, you just want to give them the ability to edit an exhibit and nothing else.

To do this, you need to create a new *Zope Permission* object in the *ZooExhibit* Product (*not* the ZClass, permissions are defined in Products only). To do this, go to the *ZooExhibit* Product and select *Zope Permission* from the add list. Give the new permission the *Id* "edit_exhibit_permission" and the *Name* "Edit Zoo Exhibits" and click *Generate*.

Now, select your *ZooExhibit* ZClass, and click on the *Permissions* tab. This will take you to the *Permissions* view as shown in Figure Figure 12-9.

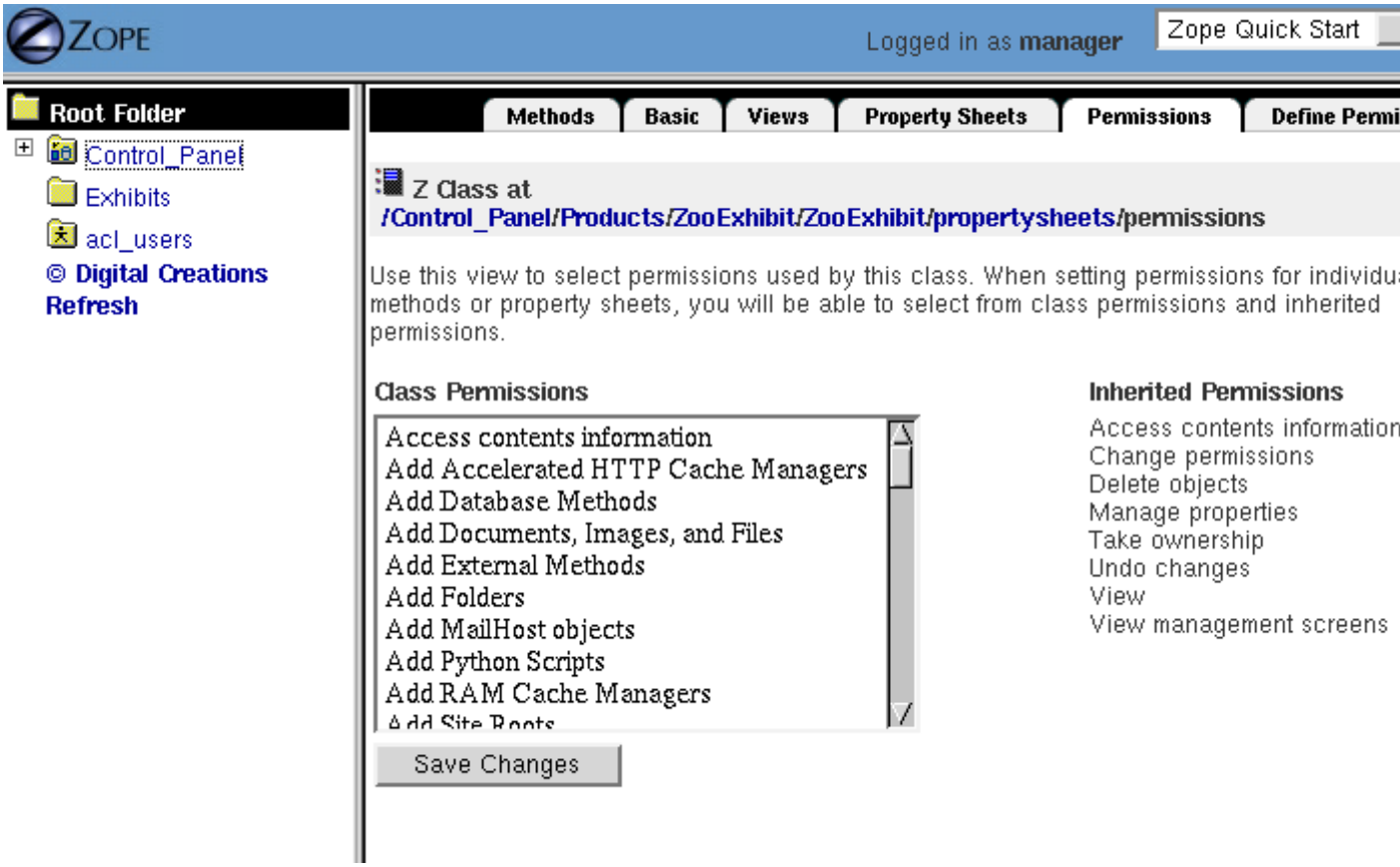


Figure 12-9 The Permissions view

This view shows you what permissions your ZClass uses and lets you choose additional permissions to use. On the right is a list of all of the default Zope permissions your ZClass inherits automatically. On the left is a multiple select box where you can add new permissions to your class. Select the *Edit Zoo Exhibits* permission in this box and click *Save Changes*. This tells your ZClass that it is interested in this permission as well as the permissions on the right.

Now, click on the *Property Sheets* tab and select the *ExhibitProperties* Property Sheet. Click on the *Define Permissions* tab.

You want to tell this Property Sheet that only users who have the *Edit Zoo Exhibits* permission you just created can manage the properties on the *ExhibitProperties* sheet. On this view, pull down the select box and choose *Edit Zoo Exhibits*. This will map the *Edit Zoo Exhibits* to the *Manage Properties* permission on the sheet. This list of permissions you can select from comes from the ZClass *Permissions* view you were just on, and because you selected the *Edit Zoo Exhibits* permission on that screen, it shows up on this list for you to select. Notice that all options default to *disabled* which means that the property sheet cannot be edited by anyone.

Now, you can go back to your *Exhibits* folder and select the *Security* view. Here, you can see your new Permission is on the left in the list of available permission. What you want to do now is create a new Role called *Caretaker* and map that new Role to the *Edit Zoo Exhibits* permission.

Now, users must have the *Caretaker* role in order to see or use the *Edit* view on any of your *ZooExhibit* instances.

Access to objects on your ZClass's *Methods* view are controlled in the same way.

Controlling Access to instances of Your ZClass

The previous section explained how you can control access to instances of your ZClass's Methods and Properties. Access control is controlling who can create new instances of your ZClass. As you saw earlier in the chapter, instances are created by Factories. Factories are associated with permissions. In the case of the Zoo Exhibit, the *Add Zoo Exhibits* permission controls the ability to create Zoo Exhibit instances.

Normally only Managers will have the *Add Zoo Exhibits* permission, so only Managers will be able to create new Zoo Exhibits. However, like all Zope permissions, you can change which roles have this permissions in different locations of your site. It's important to realize that this permission is controlled separately from the *Edit Zoo Exhibits* permission. This makes it possible to allow some people such as Caretakers to change, but not create Zoo Exhibits.

Providing Context-Sensitive Help for your ZClass

On the *View* screen of your ZClass, you can see that each view can be associated with a *Help Topic*. This allows you to provide a link to a different help topics depending on which view the user is looking at. For example, let's create a Help Topic for the *Edit* view of the *ZooExhibit* ZClass.

First, you need to create an actual help topic object. This is done by going to the *ZooExhibit* Product which contains the *ZooExhibit* ZClass, and clicking on the *Help* folder. The icon should look like a folder with a blue question mark on it.

Inside this special folder, pull down the add list and select *Help Topic*. Give this topic the id "ExhibitEditHelp" and the title "Help for Editing Exhibits" and click *Add*.

Now you will see the *Help* folder contains a new help topic object called *ExhibitEditHelp*. You can click on this object and edit it, it works just like a DTML Document. In this document, you should place the help information you want to show to your users:

```
<dtml-var standard_html_header>

  <h1>Help!</h1>

  <p>To edit an exhibit, click on either the <b>animal</b>,
  <b>description</b>, or <b>caretakers</b> boxes to edit
  them.</p>

<dtml-var standard_html_footer>
```

Now that you have created the help topic, you need to associate with the *Edit* view of your ZClass. To do this, select the *ZooExhibit* ZClass and click on the *Views* tab. At the right, in the same row as the *Edit* view is defined, pull down the help select box and select

ExhibitEditHelp and click *Change*. Now go to one of your ZooExhibit instances, the *Edit* view now has a *Help!* link that you can click to look at your Help Topic for this view.

In the next section, you'll see how ZClasses can be cobined with standard Python classes to extend their functionality into raw Python.

Using Python Base Classes

ZClasses give you a web managable interface to design new kinds of objects in Zope. In the beginning of this chapter, we showed you how you can select from a list of *base classes* to subclass your ZClass from. Most of these base classes are actually written in Python, and in this section you'll see how you can take your own Python classes and include them in that list so that your ZClasses can extend their methods.

Writing Python base classes is easy, but it involves a few installation details. To create a Python base class you need access to the filesystem. Create a directory inside your *lib/python/Products* directory named *AnimalBase*. In this directory create a file named *Animal.py* with these contents:

```
class Animal:
    """
    A base class for Animals
    """

    _hungry=0

    def eat(self, food, servings=1):
        """
        Eat food
        """
        self._hungry=0

    def sleep(self):
        """
        Sleep
        """
        self._hungry=1

    def hungry(self):
        """
        Is the Animal hungry?
        """
        return self._hungry
```

This class defines a couple related methods and one default attribute. Notice that like External Methods, the methods of this class can access private attributes.

Next you need to register your base class with Zope. Create an *__init__.py* file in the *AnimalBase* directory with these contents:

```
from Animal import Animal

def initialize(context):
```

```
"""
Register base class
"""
context.registerBaseClass(Animal)
```

Now you need to restart Zope in order for it find out about your base class. After Zope restarts you can verify that your base class has been registered in a couple different ways. First go to the Products Folder in the Control Panel and look for an *AnimalBase* package. You should see a closed box product. If you see broken box, it means that there is something wrong with your *AnimalBase* product.

Click on the *Traceback* view to see a Python traceback showing you what problem Zope ran into trying to register your base class. Once you resolve any problems that your base class might have you'll need to restart Zope again. Continue this process until Zope successfully loads your product. Now you can create a new ZClass and you should see *AnimalBase:Animal* as a choice in the base classes selection field.

To test your new base class create a ZClass that inherits from *AnimalBase:Animal*. Embellish you animal however you wish. Create a DTML Method named *care* with these contents:

```
<dtml-var standard_html_header>

<dtml-if give_food>
  <dtml-call expr="eat('cookie')">
</dtml-if>

<dtml-if give_sleep>
  <dtml-call sleep>
</dtml-if>

<dtml-if hungry>
  <p>I am hungry</p>
<dtml-else>
  <p>I am not hungry</p>
</dtml-if>

<form>
<input type="submit" value="Feed" name="give_food">
<input type="submit" value="Sleep" name="give_sleep">
</form>

<dtml-var standard_html_footer>
```

Now create an instance of your animal class and test out its *care* method. The care method lets you feed your animal and give it sleep by calling methods defined in its Python base class. Also notice how after feeding your animal is not hungry, but if you give it a nap it wakes up hungry.

As you can see, creating your own Products and ZClasses is an involved process, but simple to understand once you grasp the basics. With ZClasses alone, you can create some pretty complex web applications right in your web browser.

In the next section, you'll see how to create a *distribution* of your Product, so that you can share it with others or deliver it to a customer.

Distributing Products

Now you have created your own Product that lets you create any number of exhibits in Zope. Suppose you have a buddy at another Zoo who is impressed by your new online exhibit system, and wants to get a similar system for his Zoo.

Perhaps you even belong to the Zoo keeper's Association of America and you want to be able to give your product to anyone interested in an exhibit system similar to yours. Zope lets you distribute your Products as one, easy to transport package that other users can download from you and install in their Zope system.

To distribute your Product, click on the *ZooExhibit* Product and select the *Distribution* tab. This will take you to the *Distribution* view.

The form on this view lets you control the distribution you want to create. The *Version* box lets you specify the version for your Product distribution. For every distribution you make, Zope will increment this number for you, but you may want to specify it yourself. Just leave it at the default of "1.0" unless you want to change it.

The next two radio buttons let you select whether or not you want others to be able to customize or redistribute your Product. If you want them to be able to customize or redistribute your Product with no restrictions, select the *Allow Redistribution* button. If you want to disallow their ability to redistribute your Product, select the *Disallow redistribution and allow the user to configure only the selected objects:* button. If you disallow redistribution, you can choose on an object by object basis what your users can customize in your Product. If you don't want them to be able to change anything, then don't select any of the items in this list. If you want them to be able to change the *ZooExhibit* ZClass, then select only that ZClass. If you want them to be able to change everything (but still not be able to redistribute your Product) then select all the objects in this list.

Now, you can create a distribution of your Product by clicking *Create a distribution archive*. Zope will now automatically generate a file called *ZooExhibit-1.0.tar.gz*. This Product can be installed in any Zope just like any other Product, by unpacking it into the root directory of your Zope installation.

Don't forget that when you distribute your Product you'll also need to include any files such as External Method files and Python base classes that your class relies on. This requirement makes distribution more difficult and for this reason folks sometimes try to avoid relying on Python files when creating through the web Products for distribution.

Macht es Sinn die Namen der Klassen und Methoden zu übersetzen?? (ZooExhibit --> ZooAusstellung), wenn diese auf den Screenshots noch auf englisch stehen? Ich finde das ist ein wenig kontraproduktiv, da damit dem Leser evtl. die Zusammenhänge nicht klar werden

Zopebuch: Inhaltsverzeichnis

DTML ist die *Document Template Markup Language*, eine praktische Darstellungs- und Vorlagensprache, die in Zope integriert ist. Dieser Anhang ist eine Referenz zu allen DTML-Tags und Ihrer Funktionsweise.

call: Eine Methode aufrufen

Mit Hilfe des `call`-Tags können Sie eine Methode aufrufen, ohne die Ergebnisse in den DTML-Output einzufügen.

Syntax

`call` Tag Syntax:

```
<dtml-call Variable|expr="Ausdruck">
```

Wenn das call-Tag eine Variable verwendet, werden die Argumente automatisch von DTML weitergegeben, genau wie beim `var`-Tag. Ist die Methode in einem Ausdruck beschrieben, müssen Sie die Argumente selbst übergeben.

Beispiele

Aufruf über einen Variablennamen:

```
<dtml-call UpdateInfo>
```

Dieser Code ruft das `UpdateInfo`-Objekt auf und gibt die Argumente automatisch weiter.

Aufruf über einen Ausdruck:

```
<dtml-call expr="RESPONSE.setHeader('content-type', 'text/plain')">
```

Beachten Sie auch

var-Tag

Kommentar: Kommentiert DTML

Mit dem comment-Tag können sie DTML mit Kommentaren dokumentieren. Sie können es auch dazu verwenden, DTML-Tags vorübergehend auszuschalten, indem Sie sie auskommentieren.

Syntax

`comment` Tag Syntax:

```
<dtml-comment>
</dtml-comment>
```

Das `comment` Tag ist ein Block-Tag. Die Inhalte des Blocks werden weder ausgeführt, noch werden sie in den DTML-Output eingefügt.

Beispiele

DTML dokumentieren:

```
<dtml-comment>
  Dieser Inhalt wird nicht ausgeführt und erscheint nicht im
Output.
</dtml-comment>
```

DTML auskommentieren:

```
<dtml-comment>
  Diese DTML ist abgeschaltet und wird nicht ausgeführt.
  <dtml-call someMethod>
</dtml-comment>
```

Funktionen: DTML-Funktionen

DTML-Dienstprogrammfunktionen stellen einige in Python integrierte sowie einige DTML-spezifische Funktionen zur Verfügung.

Funktionen

abs(Zahl)
: Gibt den absoluten Wert einer Zahl zurück. Das Argument kann ein plain-Integer, ein long-Integer oder eine Gleitkommazahl sein. Ist das Argument eine komplexe Zahl, wird der Betrag der Zahl zurückgegeben.

chr(integer)
: Gibt einen String von einem Character Länge zurück, dessen ASCII-Code der Integer ist, beispielsweise gibt `chr(97)` den String `a` zurück. Dies ist die Umkehrun von ord(). Das Argument muss im Zahlenraum von 0 bis einschliesslich 255 liegen; ein `ValueError` wird auftreten, falls der integer ausserhalb dieses Bereiches ist.

DateTime()
: Gibt ein Zope `DateTime`-Objekt zurück, sofern Konstruktorargumente übergeben wurden. Bitte beachten Sie die DateTime API-Referenz für weitere Informationen zu Konstruktorargumenten.

divmod(Zahl, Zahl)
: Benutzt zwei Zahlen als Argumente und gibt ein Zahlenpaar zurück, das aus dem Quotienten und dem Restbetrag besteht, wenn eine long-Division ausgeführt wird. Bei gemischten Zahlentypen werden die Regeln für binäre arithmetische Operanden angewendet. Für plain und long-Integers ist das Ergebnis wie bei `(a / b, a % b)`.

Bei Gleitkommazahlen ist das Ergebnis gleich `(q, a % b)`, wobei *q* normalerweise `math.floor(a / b)` beträgt, es kann jedoch um 1 reduziert sein. In jedem Fall ist 'q * b + a % b' sehr nahe an *a*, sofern `a % b` nicht Null ist, hat die Berechnung das selbe Vorzeichen wie *b* und `0 <= abs(a % b) < abs(b).`

float(Zahl)

Wandelt einen String oder eine Zahl in eine Gleitkommazahl um. Wenn das Argument ein String ist, muss es eine positive Dezimalzahl oder eine Gleitkommazahl, die in Leerzeichen eingebettet sein kann, enthalten; dies verhält sich wie `string.atof(number)`. Ansonsten kann das Argument ein long oder integer sein, in diesem Fall wird eine Gleitkommazahl mit dem selben Wert (innerhalb Pythons Gleitkommazahlpräzisierung) zurückgeliefert.

getattr(Objekt, String)

Gibt den Namen der Attribute eines Objektes zurück. Der Name muss ein String sein. Ist der String der Name eines der Attribute des Objektes, so wird der Wert des Attributes zurügeliefert. Beispielsweise ist `getattr(x, "foobar")` gleich `x.foobar`. Wenn das genannte Attribut nicht existiert, wird, falls vorhanden, der Defaultwert zurückgeliefert, ansonsten wird ein `AttributeError` ausgegeben.

getitem(Variable, render=0)

Gibt den Wert einer DTML-Variable zurück. Wenn `render` wahr ist, wird die Variable gerendert. Bitte beachten Sie die `render`-Funktion.

hasattr(Objekt, String)

Die Argumente sind hier ein Objekt und ein String. Das Ergebnis ist 1, wenn der String der Name eines der Attribute des Objektes ist, andernfalls 0. (Dies wird durch den Aufruf von getattr(Objekt, Name) und die Überprüfung implementiert, ob eine Exception geworfen wird oder nicht.)

hash(Objekt)

Gibt den Hash-Wert eines Objektes zurück, sofern es einen besitzt. Hash-Werte sind Integers. Sie werden genutzt, um schnell Dictionary-Schlüssel während einer Dictionary-Suche zu vergleichen. Numerische Werte die gleich sind haben auch den selben Hash-Wert (selbst wenn es sich um verschiedene Zahlentypen handelt, z.B. 1 und 1.0).

has_key(Variable)

Gibt true zurück, wenn der DTML-Namensraum die gesuchte Variable enthält.

hex(integer)

Konvertiert einen integer (egal welcher Grösse) in einen hexadezimalen String um. Das Ergebnis ist ein gültiger Python-Ausdruck. Bitte beachten Sie: dies liefert immer ein vorzeichenloses Literal, z.B. ergibt `hex(-1)` auf einem 32-bit Rechner `0xffffffff`. Wird die Rückberechnung auf einem Rechner mit der selben Wortlänge ausgeführt, wird als Ergebnis -1 ausgegeben, bei einer anderen Wortlänge könnte jedoch eine grosse positive Zahl ausgegeben werden oder aber ein `OverflowError` auftreten.

int(Zahl)

Wandelt einen String oder eine Zahl in einen integer um. Ist das Argument ein String, muss er eine positive Dezimalzahl (einen Python-integer) enthalten, die in Leerzeichen eingebettet sein kann; dies verhält sich wie 'string.atoi(number[, radix]'). Der `radix`-Parameter wird als Basis für die Umwandlung genutzt und kann ein integer mit einem Wert von 2 bis 36 sein. Wird `radix` benutzt und die Zahl ist kein String, wird ein `TypeError` ausgegeben. Andernfalls kann das Argument ein integer oder eine Gleitkommazahl sein. Die Umwandlung von Gleitkommazahlen in integers wird von der C-Semantik definiert; normalerweise rundet die Umwandlung ab.

len(Sequenz)

Gibt die Länge (die Zahl der Elemente) eines Objektes zurück. Das Argument kann eine Sequenz (String, Tupel oder Liste) oder eine Zuordnung (Dictionary) sein.

max(s)

Mit dem einzelnen Argument s gibt dieser Ausdruck das grösste Element einer nicht(!)-leeren Sequenz (z.B. String, Tupel oder Liste) zurück. Bei mehr als einem Argument wird das grösste Element der Argumentenliste zurückgegeben.

min(s)

Mit dem einzelnen Argument s wird hier das kleinste Element einer nicht(!)-leeren Sequenz (z.B. String, Tupel, Liste) zurückgegeben. Bei mehr als einem Argument wird das kleinste Element der Argumentenliste zurückgegeben.

namespace([name=value]...)

Gibt ein neues DTML-Namensraum-Objekt zurück. Die Schlüsselwort - Argument-Paare `name=value` werden in den neuen Namensraum eingefügt.

oct(integer)

Wandelt einen integer (jeglicher Grösse) in einen Oktal-String um. Das Ergebnis ist ein gültiger Python-Ausdruck. Bitte beachten Sie: diese Umwandlung ergibt immer ein vorzeichenloses Literal, beispielsweise ergibt `oct(-1)` auf einem 32-bit-Rechner `037777777777`. Wird die Berechnung auf einem Rechner mit der selben Wortlänge durchgeführt, wird das Literal als -1 berechnet; bei einer anderen Wortlänge könnte eine grosse positive Zahl oder ein `OverflowError` ausgegeben werden.

ord(Character)

Gibt den ASCII-Wert eines Strings, der aus genau einem Character besteht, aus. Beispielsweise ergibt `ord("a")` den integer 97. Dies ist das Gegenteil von `chr()`.

pow(x, y [,z])

Gibt *x* hoch *y* zurück; falls auch *z* angegeben ist, wird *x* hoch *y*, modulo *z* ausgegeben (diese Berechnung ist besser geeignet als 'pow(x, y) % z'). Die Argumente müssen numerisch sein. Bei gemischten Zahlentypen werden die Regeln für binäre arithmetische Operanden angewandt. Der tatsächliche Zahlentyp bestimmt auch den Zahlentyp des Ergebnisses; ist das Ergebnis in diesem Typ nicht auszudrücken, wird eine Exception ausgegeben, z.B. `pow(2, -1)` oder `pow(2, 35000)` sind nicht erlaubt.

range([start,] stop [,step])

Dies ist eine vielseitige Funktion um Listen, die arithmetische Reihen enthalten, zu erstellen. Die Argumente müssen vom Typ integer sein. Wird kein step-Argument angegeben, wird der Defaultwert 1 verwendet. Wird kein start-Argument angegeben, wird der Defaultwert 0 verwendet. Der komplette Ausdruck gibt eine Liste von integers zurück: '[Startwert, Startwert + step, Startwert + 2 * step, ...]'. Ist der Wert für step positiv, ist das letzte Element der Reihe das grösste und um 'Startwert + i *step' kleiner als* stop*; ist der Wert für* step *negativ, ist das letzte Element das kleinste und um 'start + i* step' grösser als *stop*. Der Wert für *step* darf NICHT 0 sein, sonst wird ein `ValueError` ausgegeben.

round(x [,n])

Gibt die Gleitkommazahl *x*, gerundet auf *n* Nachkommastellen aus. Wird n nicht angegeben, wird der Defaultwert 0 verwendet. Das Ergebnis ist wiederum eine Gleitkommazahl. Werte werden auf das nächste Vielfache von 10 hoch -n gerundet; sind zwei Werte gleich nach an x, wird von 0 weggerundet. So ist zum Beispiel round(0.5) 1.0, round(-0.5) dagegen nicht 0, sondern -1.0.

render(Objekt)

Rendert das `Objekt`. Für DTML-Objekte berechnet dieser Ausdruck den DTML-Code mit dem aktuellen Namensraum. Für andere Objekte ist dies das gleiche wie `str(object)`.

reorder(s [,with] [,without])

Ordnet die Elemente in s anhand der Reihenfolge, die in `with` definiert wird, ohne die Elemente, die in `without` enthalten sind, einzuschliessen. Elemente, die in s enthalten sind, aber in with nicht erwähnt werden, werden entfernt. s, with und without sind alle entweder Sequenzen von Strings oder Sequenzen von Schlüssel-Wert-Tupeln, die nach den Schlüsseln geordnet werden. Diese Funktion ist nützlich, um geordnete Auswahllisten zu erstellen.

SecurityCalledByExecutable()
: Gibt wahr zurück, wenn das aktuelle Objekt (z.B. ein DTML-Dokument oder eine DTML-Methode) von einer ausführbaren Anweisung (beispielsweise ein anderes DTML-Dokument oder eine DTML-Methode, ein Skript oder eine SQL-Anweisung) aufgerufen wird.

SecurityCheckPermission(permission, object)
: Überprüft, ob der Sicherheitskontext das gegebene Recht im Bezug auf das angegebene Objekt erlaubt. Beispielsweise gibt 'SecurityCheckPermission("Add Documents, Images, and Files", this())' wahr zurück wenn der aktuelle Benutzer berechtigt ist, Dokumente, Bilder und Dateien an der angegebenen Position zu erstellen.

SecurityGetUser()
: Gibt das aktuelle User-Objekt zurü. Dies ist normalerweise das selbe wie das `REQUEST.AUTHENTICATED_USER` Objekt. Jedoch ist das `AUTHENTICATED_USER` Objekt unsicher, da es ersetzt werden kann.

SecurityValidate([object] [,parent] [,name] [,value])
: Gibt true zurück wenn der aktuelle User auf den Wert zugreifen kann. `object` ist das Objekt, in dem auf den Wert zugegriffen wird, `parent` ist der Behälter, in dem der Wert liegt und `name` ist der Name, mit dem auf den Wert zugegriffen wird (z.B. "getattr", wenn über diesen Namen auf den Wert zugegriffen wird). Sie können einige dieser Argumente vernachlässigen, auf alle Fälle sollten Sie aber alle Argumente, die Ihnen zur Verfügung stehen, angeben.

SecurityValidateValue(object)
: Gibt true zurück, wenn das Objekt für den aktuellen Benutzer zugänglich ist. Diese Funktion erfüllt dieselbe Aufgabe wie der Aufruf von `SecurityValidate(None, None, None, Objekt)`.

str(object)
: Gibt einen String zurück, der eine für den Ausdruck optimierte Darstellung eines Objektes enthält. Bei Angabe von Strings werden diese unverändert zurückgegeben.

test(condition, result [,condition, result]... [,default])
: Nimmt eine oder mehrere Bedingung-Ergebnis-Tupel an und gibt das Ergebnis der ersten erfüllten Bedingung zurück. Es wird nur ein Ergebnis zurückgegeben, selbst wenn mehr als eine Bedingung erfüllt ist. Ist keine der Bedingungen erfüllt und ein Defaultwert ist vorgegeben, so wird dieser zurückgegeben. Ist keine der Bedingungen erfüllt und auch kein Defaultwert angegeben, wird nichts zurückgegeben.

unichr(Zahl)
: Gibt einen Unicode-String zurück, der den Wert der Zahl als Unicode-Character darstellt. Dies ist die Umkehrung von ord() für Unicode-Characters.

unicode(string[, encoding[, errors]])
: Verschlüsselt einen String unter Benutzung des Codes für die Verschlüsselung. Die Fehlerbehandlung wird gemäss der Fehler ausgeführt. Im Normalfall wird UTF-8 in strict mode verschlüsselt, was bedeutet, dass Verschlüsselungsfehler einen ValueError hervorrufen.

Attribute

None
: Das `None`-Objekt entspricht dem in Python integrierten `None`. Dies wird normalerweise benutzt, um einen Null- oder einen falschen Wert zu beschreiben.

Bitte beachten Sie auch

`string` module

`random` module

`math` module

`sequence` module

In Python integrierte Funktionen

if: Testbedingungen

Das `if`-Tag erlaubt es Ihnen, Bedingungen zu testen und verschiedene Aktionen, die von den Bedingungen abhängen, durchzuführen. Das `if`-Tag spiegelt Pythons `if/elif/else`-Ausdrücke zum Testen von Bedingungen wider.

Syntax

If Tag Syntax:

```
<dtml-if Bedingungsvariable|expr="Bedingungsausdruck">
[<dtml-elif Bedingungsvariable|expr="Bedingungsausdruck">]
 ...
[<dtml-else>]
</dtml-if>
```

Das `if`-Tag ist ein Block-Tag. Das `if`-Tag und optionale `elif`-Tags nehmen eine Bedingungsvariable oder einen Bedingungsausdruck auf, aber nicht beides zusammen. Wird der Bedingungsname oder der Ausdruck als true zurückgegeben, wird der `if`-Block ausgeführt. True bedeutet nicht 0, einen leeren String oder eine leere Liste. Falls die Bedingungsvariable nicht gefunden wird, wird die Bedingung als false angenommen.

Ist die Anfangsbedingung falsch, wird jede der `elif`-Bedingungen nacheinander getestet. Wenn eine der `elif`-Bedingungen wahr ist, wird ihr Block ausgeführt. Schliesslich wird der optionale `else`-Block ausgeführt falls keine der `if`- oder `elif`-Bedingungen true ergeben hat. Es wird immer nur ein Block ausgeführt.

Beispiele

Eine Variable testen:

```
<dtml-if snake>
```

```
  Die snake-Variable ist true
</dtml-if>
```

Ausdrucksbedingungen testen:

```
<dtml-if expr="num > 5">
  num ist grösser als 5
<dtml-elif expr="num < 5">
  num ist kleiner als 5        <dtml-else>
  num ist genau 5
</dtml-if>
```

Bitte beachten Sie auch:

Python Tutorial: If-Statements

in: Sequenzwiederholungen

Das `in`-Tag gibt Ihnen umfassende Kontrolle über Sequenzwiederholungen und die Ausführung von batch-Prozessen.

Syntax

`in` Tag Syntax:

```
<dtml-in SequenzVariable|expr="SequenzAusdruck">
[<dtml-else>]
</dtml-in>
```

Der `in`-Block wird einmal für jedes Element in der Sequenzvariable oder dem Sequenzausdruck wiederholt. Das aktuelle Element wird während jeder Ausführung des `in`-Tags in den DTML-Namensraum eingefügt. Sind in der Sequenzvariablen bzw. im Sequenzausdruck keine Elemente enthalten, wird der optionale `else`-Block ausgeführt.

Attribute

mapping
: Eine Wiederholung, die auf der Zuordnung von Objekten und nicht auf ihren Instanzen beruht. Dies erlaubt es dem User, Werte der zugeordneten Objekte als DTML-Variablen zu benutzen.

reverse
: Dreht die Sequenz um.

sort=string
: Sortiert die Sequenz anhand des angegebenen Attributnamens.

start=int
: Die Nummer des ersten Elementes das angezeigt werden soll, die Folge beginnt bei 1.

end=int

Die Nummer des letzten Elementes das angezeigt werden soll, die Folge beginnt bei 1.

size=int
: Die Grösse des Batch.

skip_unauthorized
: Verhindert, dass eine Fehlermeldung erscheint, wenn ein nicht autorisiertes Element entdeckt wird.

orphan=int
: Die benötigte Mindestgrösse des Batch. Sie bestimmt, wie Sequenzen in Batches aufgeteilt werden. Tritt ein Batch auf, das kleiner ist als die Grösse von orphan, wird keine Aufteilung ausgeführt, woraus sich ein Batch ergibt das grösser ist als die eigentliche Batch-Grösse.

  Beispiel: Ist die Sequenzgrösse 12, die Batchgrösse 10 und die Grösse von orphan 3, so ergibt sich daraus ein Batch mit allen 12 Elementen, da aus einer Aufteilung der Elemente in zwei Batches ein Batch resultieren würde, das kleiner ist als die Grösse von orphan.

  Der Defaultwert ist 0.

overlap=int
: Die Zahl der Elemente, die zwischen Batches überhängen sollen. Per Default ist kein overlap eingestellt.

previous
: Wiederholt einmal, falls es ein früheres Batch gibt. Setzt die Batchvariablen für die frühere Sequenz.

next
: Wiederholt einmal, falls es ein nächstes Batch gibt. Setzt die Variablen für die nächste Sequenz.

prefix=string
: Stellt Versionen der Tagvariablen zur Verfügung, die mit der gegebenen Vorsilbe statt "sequence" anfangen und die Unterstriche(_) statt Bindestriche (-) benutzen. Diese Vorsilbe muss mit einem Buchstaben beginnen und darf nur alphanumerische Zeichen und Unterstriche (_) enthalten.

sort_expr=Ausdruck
: Sortiert die Sequenz anhand eines Attributes, das vom Wert des Ausdrucks benannt wird. Dies gibt Ihnen die Möglichkeit, nach verschiedenen Attributen zu sortieren.

reverse_expr=Ausdruck
: Dreht die Sequenz um, falls der Ausdruck true ergibt. Dies gibt Ihnen die Möglichkeit, die Sequenz anhand einer von Ihnen erstellten Auswahl umzudrehen.

Tag-Variablen

Current Item-Variablen

Diese Variablen beschreiben das aktuelle Element.

sequence-item
: Das aktuelle Element.

sequence-key

Der aktuelle Schlüssel. Wenn Sie eine Wiederholung über Tupel der Form `(Schlüssel,Wert)` laufen lassen, interpretiert das `in`-Tag sie als `(Sequenz-Schlüssel, Sequenz-Element)`.

sequence-index
: Sequenz-Index, startet mit 0 beim aktuellen Element.

sequence-number
: Die Sequenznummer, startet mit 1 beim aktuellen Element.

sequence-roman
: Der Sequenz-Index des aktuellen Elements in kleingeschriebenen römischen Ziffern.

sequence-Roman
: Der Sequenz-Index des aktuellen Elements in grossgeschriebenen römischen Ziffern.

sequence-letter
: Der Sequenz-Index des aktuellen Elements in Kleinbuchstaben.

sequence-Letter
: Der Sequenz-Index des aktuellen Elements in Grossbuchstaben.

sequence-start
: Gibt true zurück, wenn das aktuelle Element das erste der Sequenz ist.

sequence-end
: Gibt true zurück, wenn das aktuelle Element das letzte in der Sequenz ist.

sequence-even
: Gibt true zurück, wenn der Index des aktuellen Elements gerade ist.

sequence-odd
: Gibt true zurück, wenn der Index des aktuellen Elements ungerade ist.

sequence-length
: Die L&aumlnge der Sequenz.

sequence-var-*Variable*
: Eine Variable im aktuellen Element. Beispielsweise ist `sequence-var-title` die `title`-Variable des aktuellen Elements. Normalerweise können Sie auf diese Variablen direkt zugreifen, da das aktuelle Element in den DTML-Namensraum eingefügt wird. Allerdings können diese Variablen nützlich sein, wenn sie die vorige und nächste Batch-Information anzeigen wollen.

sequence-index-*Variable*
: Der Index einer Variable des aktuellen Elements.

Zusammenfassungsvariablen

Diese Variablen fassen Informationen über numerische Elementvariablen zusammen. Um sie zu verwenden müssen Sie eine Wiederholung über Objekte laufen lassen (wie Datenbankergebnislisten), die numerische Variablen enthalten.

total-*Variable*
: Die Summe aller Ausprägungen einer Elementvariable.

count-*Variable*
: Die Zahl der Ausprägungen einer Elementvariable.

min-*Variable*
: Der kleinste Wert einer Elementvariable.

max-*Variable*
: Der grösste Wert einer Elementvariable.

mean-*Variable*
: Der Mittelwert einer Elementvariable.

variance-*Variable*
: Die Abweichung einer Elementvariable von ihrem Mittelwert mit einem Grad Freiraum.

variance-n-*Variable*
: Die Abweichung einer Elementvariable von ihrem Mittelwert mit n Graden Freiraum.

standard-deviation-*Variable*
: Die Standard-Abweichung einer Elementvariable mit einem Grad Freiraum.

standard-deviation-n-*Variable*
: Die Standard-Abweichung einer Elementvariable mit n Graden Freiraum.

Gruppierungsvariablen

Diese Variablen geben Ihnen die Möglichkeit, Veränderungen an den aktuellen Elementvariablen zu verfolgen.

first-*Variable*
: Gibt true zurück, wenn das aktuelle Element das erste mit einem bestimmten Wert für eine bestimmte Variable ist.

last-*Variable*
: Gibt true zurück, wenn das aktuelle Element das letzte mit einem bestimmten Wert für eine bestimmte Variable ist.

Batch-Variablen

sequence-query
: Der query-String ohne die `start`-Variable. Sie können diese Variable verwenden, um Links zu nächsten und vorigen Batches zu erstellen.

sequence-step-size
: Die Grösse des Batch.

previous-sequence
: Gibt true zurück, wenn das aktuelle Batch nicht das erste ist. Bitte beachten Sie, dass diese Variable bei einer Wiederholung nur im ersten Durchgang true zurückgibt.

previous-sequence-start-index
: Der Start-Index des letzten Batches.

previous-sequence-start-number
: Die Startnummer des letzten Batches. Bitte beachten Sie, dass dies das selbe ist wie `previous-sequence-start-index` + 1.

previous-sequence-end-index
: Der End-Index des letzten Batches.

previous-sequence-end-number
: Die Endnummer des letzten Batches. Bitte beachten Sie, dass dies das selbe ist wie `previous-sequence-end-index` + 1.

previous-sequence-size
: Die Grösse des letzten Batches.

previous-batches
: Eine Sequenz von Zuordnungsobjekten mit Informationen über alle früheren Batches. Jedes dieser Objekte hat folgende Schlüssel: `batch-start-index`, `batch-end-index` und `batch-size`.

next-sequence

Gibt true zurück, wenn der vorige Batch nicht der letzte ist. Bitte beachten Sie: diese Variable ist bei einer Wiederholung nur im ersten Durchgang true.

next-sequence-start-index
: Der Start-Index der nächsten Sequenz.

next-sequence-start-number
: Die Startnummer der nächsten Sequenz. Bitte beachten Sie, dass dies das selbe ist wie `next-sequence-start-index` + 1.

next-sequence-end-index
: Die Endnummer der nächsten Sequenz.

next-sequence-end-number
: Die Endnummer der nächsten Sequenz. Bitte beachten Sie, dass dies das selbe ist wie `next-sequence-end-index` + 1.

next-sequence-size
: Die Grösse des nächsten Indexes.

next-batches
: Eine Sequenz von Zuordnungsobjekten mit Informationen über alle folgenden Batches. Jedes Zuordnungsobjekt hat folgende Schlüssel: `batch-start-index`, `batch-end-index` und `batch-size`.

Beispiele

Eine Wiederholung über Sub-Objekte laufen lassen:

```
<dtml-in objectValues>
  title: <dtml-var title><br>
</dtml-in>
```

Eine Wiederholung über zwei Sets von Objekten unter Zuhilfenahme von Vorsilben laufen lassen:

```
<dtml-let rows="(1,2,3)" cols="(4,5,6)">
  <dtml-in rows prefix="row">
    <dtml-in cols prefix="col">
      <dtml-var expr="row_item * col_item"><br>
      <dtml-if col_end>
        <dtml-var expr="col_total_item * row_mean_item">
      </dtml-if>
    </dtml-in>
  </dtml-in>
</dtml-let>
```

Eine Wiederholung über eine Liste von `(Schlüssel, Wert)` Tupeln laufen lassen:

```
<dtml-in objectItems>
  id: <dtml-var sequence-key>, title: <dtml-var title><br>
</dtml-in>
```

Die Farbe von Datenbankfeldern verändern:

```
        <table>
        <dtml-in objectValues>
        <tr <dtml-if sequence-odd>bgcolor="#EEEEEE"
            <dtml-else>bgcolor="#FFFFFF"
            </dtml-if>
          <td><dtml-var title></td>
        </tr>
        </dtml-in>
        </table>
```

Grundlegende Batchverarbeitung:

```
        <p>
        <dtml-in largeSequence size=10 start=start previous>
          <a href="<dtml-var absolute_url><dtml-var sequence-
query>start=<dtml-var previous-sequence-start-number>">Previous</a>
        </dtml-in>

        <dtml-in largeSequence size=10 start=start next>
          <a href="<dtml-var absolute_url><dtml-var sequence-
query>start=<dtml-var next-sequence-start-number>">Next</a>
        </dtml-in>
        </p>

        <p>
        <dtml-in largeSequence size=10 start=start>
          <dtml-var sequence-item>
        </dtml-in>
        </p>
```

Dieses Beispiel erstellt *Previous*- und *Next*-Links um sich zwischen verschiedenen Batches zu bewegen. Bitte beachten Sie, dass Sie, wenn Sie `sequence-query` verwenden, keine der GET-Variablen verlieren, wenn Sie sich zwischen den Batches bewegen.

let: Definiert DTML-Variablen

Das `let`-Tag definiert Variablen im DTML-Namensraum.

Syntax

`let` Tag Syntax:

```
        <dtml-let [Name=Variable][Name="Ausdruck"]...>
        </dtml-let>
```

Das `let`-Tag ist ein Block-Tag. Variablen werden von Tag-Argumenten definiert. Definierte Variablen werden in den DTML-Namensraum eingefügt, während der `let`-Block ausgeführt wird. Das `let`-Tag kann ein oder mehrere Attribte mit beliebigen Namen haben. Wenn die Attribute in Anführungszeichen definiert werden, werden Sie als Ausdrücke betrachtet, andernfalls werden Sie anhand des angegebenen Namens gesucht. Attribute werden der Reihe

nach abgearbeitet, so dass solche, die weiter hinten in der Reihe stehen, frühere referenzieren und/oder überschreiben können.

Beispiele

Grundlegende Verwendung:

```
<dtml-let name="'Bob'" ids=objectIds>
  name: <dtml-var name>
  ids: <dtml-var ids>
</dtml-let>
```

Verwendung des `let`-Tags mit dem `in`-Tag:

```
<dtml-in expr="(1,2,3,4)">
  <dtml-let num=sequence-item
            index=sequence-index
            result="num*index">
    <dtml-var num> * <dtml-var index> = <dtml-var result>
  </dtml-let>
</dtml-in>
```

Dies ergibt:

```
1 * 0 = 0
2 * 1 = 2
3 * 2 = 6
4 * 3 = 12
```

Bitte beachten Sie auch:

with-Tag

mime: Formatiert Daten mit MIME

Mit Hilfe des `mime`-Tags können Sie mit MIME kodierte Daten erzeugen. Dies wird hauptsächlich für das Formatieren von e-mails innerhalb des `sendmail`-Tags verwendet.

Syntax

`mime` Tag Syntax:

```
<dtml-mime>
[<dtml-boundry>]
...
</dtml-mime>
```

Das `mime`-Tag ist ein Block-Tag. Der Block kann von einem oder mehreren `boundry`-Tags aufgeteilt werden, um eine multi-part MIME Nachricht zu erstellen. `mime`-Tags können verschachtelt werden. Das `mime`-Tag wird am meisten innerhalb des `sendmail`-Tags genutzt.

Attribute

Sowohl das `mime`-Tag als auch das `boundry`-Tag haben die selben Attribute.

encode=string
: Der MIME Content-Transfer-Encoding Header mit dem Defaultwert `base64`. Gültige Verschlüsselnungsoptionen beinhalten `base64`, `quoted-printable`, `uuencode`, `x-uuencode`, `uue`, `x-uue` und `7bit`. Wird das `encode`-Attribute auf `7bit` gesetzt, wird an diesem Block keine Codierung vorgenommen und es wird angenommen, dass es sich bei den Daten um ein gültiges MIME-Format handelt.

type=string
: MIME Content-Type Header.

type_expr=string
: MIME Content-Type Header als Variablenausdruck. Sie können nicht gleichzeitig `type` und `type_expr` benutzen.

name=string
: MIME Content-Type Header-Name.

name_expr=string
: MIME Content-Type Header-Name als Variablenausdruck. Sie können nicht gleichzeitig `name` und `name_expr` benutzen.

disposition=string
: MIME Content-Disposition Header.

disposition_expr=string
: MIME Content-Disposition Header als Variablenausdruck. Sie können nicht gleichzeitig `disposition` und `disposition_expr` benutzen.

filename=string
: MIME Content-Disposition Header-Dateiname.

filename_expr=string
: MIME Content-Disposition Header-Dateiname als Variablenausdruck. Sie können nicht gleichzeitig `filename` und `filename_expr` benutzen.

skip_expr=string
: Ein Variablenausdruck der, wenn er true ergibt, den Block überspringt. Sie können dieses Attribut verwenden, um anhand einer persönlichen Auswahl MIME-Blocks einzufügen.

Beispiele

Einen Anhang senden:

```
<dtml-sendmail>
To: <dtml-recipient>
Subject: Resume
<dtml-mime type="text/plain" encode="7bit">

Hi, please take a look at my resume.
```

```
        <dtml-boundary type="application/octet-stream"
disposition="attachment"
        encode="base64" filename_expr="resume_file.getId()"><dtml-var
expr="resume_file.read()"></dtml-mime>
        </dtml-sendmail>
```

Bitte beachten Sie auch:

Python Library: mimetools

raise: Löst eine Exception aus

Das `raise`-Tag gibt eine Exception aus, die sich an Pythons `raise`-Ausdruck orientiert.

Syntax

`raise` Tag Syntax:

```
        <dtml-raise ExceptionName|ExceptionAusdruck>
        </dtml-raise>
```

Das `raise`-Tag ist ein Block-Tag. Es gibt eine Exception aus. Exceptions können Exceptionklassen oder Strings sein. Die Inhalte des Tags werden als Fehlervariable weitergegeben.

Beispiele

Einen KeyError ausgeben:

```
        <dtml-raise KeyError></dtml-raise>
```

Einen HTTP-404-Fehler ausgeben:

```
        <dtml-raise NotFound>Web Page Not Found</dtml-raise>
```

Bitte beachten Sie auch:

try-Tag

Python Tutorial: Errors and Exceptions

Python Built-in Exceptions

return: Gibt Daten zurück

Das `return`-Tag beendet die Ausführung von DTML und gibt Daten zurück. Es spiegelt Pythons `return`-Statement wider.

Syntax

`return` Tag Syntax:

```
<dtml-return ReturnVariable|expr="ReturnAusdruck">
```

Beendet die Ausführung von DTML und gibt eine Variable oder einen Ausdruck zurück. Der DTML-Output wird nicht ausgegeben. Normalerweise ist ein return-Ausdruck sinnvoller als eine return-Variable, denn in Skripten ist dieses Tag weitgehend überholt.

Beispiele

Eine Variable ausgeben:

```
<dtml-return result>
```

Ein Python-Dictionary ausgeben:

```
<dtml-return expr="{'hi':200, 'lo':5}">
```

sendmail: Sendet emails per SMTP

Das `sendmail`-Tag sendet eine e-mail unter Verwendung von SMTP.

Syntax

`sendmail` Tag Syntax:

```
<dtml-sendmail>
</dtml-sendmail>
```

Das `sendmail`-Tag ist ein Block-Tag. Es benötigt entweder ein `mailhost`- oder ein `smtphost`-Argument, aber nicht beides. Der Tag-Block wird als e-mail gesendet. Der Anfang des Blocks beschreibt die e-mail Header. Die Header werden durch eine leere Zeile vom mail-Körper getrennt. Wahlweise können die `To`, `From` und `Subject` Header mit Tag-Argumenten versehen werden.

Attribute

mailhost

Der Name eines Zope-MailHost-Objektes, das benutzt wird, um die e-mail zu verschicken. Sie können nicht gleichzeitig einen mailhost und einen smtphost definieren.

smtphost
: Der Name eines SMTP-Servers, der verwendet wird, um e-mails zu verschicken. Sie können nicht gleichzeitig einen mailhost und einen smtphost definieren.

port
: Wird das smtphost-Attribut verwendet, wird das port-Attribut dazu benutzt, um eine Portnummer zu definieren, zu der eine Verbindung hergestellt wird. Wird es nicht definiert, wird Port 25 verwendet.

mailto
: Die Empfängeradresse oder eine Liste von Empfängeradressen, die durch Kommas getrennt werden. Dies kann auch im `To` Header definiert werden.

mailfrom
: Die Absenderadresse. Die kann auch im `From` Header definiert werden.

subject
: Die Überschrift der e-mail. Dies kann auch im `Subject` Header definiert werden.

Beispiele

Eine e-mail über einen Mail-Host versenden:

```
<dtml-sendmail mailhost="mailhost">
To: <dtml-var recipient>
From: <dtml-var sender>
Subject: <dtml-var subject>

Dear <dtml-var recipient>,

Your order number <dtml-var order_number> is ready.
Please pick it up at your soonest convenience.
</dtml-sendmail>
```

Bitte beachten Sie auch:

RFC 821 (SMTP Protocol)

mime-Tag

sqlgroup: Formatiert komplexe SQL-Ausdrücke

Das `sqlgroup`-Tag formatiert komplexe wahr/falsch-SQL-Ausdrücke. Sie können es mit dem `sqltest`-Tag zusammen verwenden, um dynamische SQL-Abfragen zu erstellen, die sich selber auf die Umbegung zuschneiden. Dieses Tag wird in SQL-Methoden verwendet.

Syntax

`sqlgroup` Tag Syntax:

```
<dtml-sqlgroup>
[<dtml-or>]
[<dtml-and>]
...
</dtml-sqlgroup>
```

Das `sqlgroup`-Tag ist ein Block-Tag. Es wird in Blöcke mit einem oder mehreren optionalen `or`- und `and`-Tags unterteilt. `sqlgroup`-Tags können verschachtelt werden, um komplexere Logiken zu erreichen.

Attribute

required=boolean
: Zeigt an, ob die Gruppe benötigt wird. Wird sie nicht benötigt und ist leer, wird sie aus dem DTML-Output ausgeschlossen.

where=boolean
: Ist hier true gesetzt, wird der String "where" eingeschlossen. Dies ist beim Grossteil der `sqlgroup`-Tags in einer SQL-`select`-Abfrage nützlich.

Beispiele

Beispielvewendung:

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqltest salary op="gt" type="float" optional>
<dtml-and>
  <dtml-sqltest first type="nb" multiple optional>
<dtml-and>
  <dtml-sqltest last type="nb" multiple optional>
</dtml-sqlgroup>
```

Ist `first` auf den Wert `Bob` gesetzt und `last` auf den Wert `Smith, McDonald`, wird folgendes berechnet:

```
select * from employees
where
(first='Bob'
 and
 last in ('Smith', 'McDonald')
)
```

Ist `salary` auf den Wert 50000 und `last` auf den Wert `Smith` gesetzt, wird folgendes berechnet:

```
select * from employees
where
(salary > 50000.0
 and
 last='Smith'
```

```
)
```

Verschachtelte `sqlgroup`-Tags:

```
select * from employees
<dtml-sqlgroup where>
  <dtml-sqlgroup>
     <dtml-sqltest first op="like" type="nb">
  <dtml-and>
     <dtml-sqltest last op="like" type="nb">
  <dtml-sqlgroup>
<dtml-or>
  <dtml-sqltest salary op="gt" type="float">
</dtml-sqlgroup>
```

Werden Beispielargumente eingefügt, berechnet diese Vorlage in SQL ausgedrückt folgendes:

```
select * form employees
where
(
  (
   name like 'A*'
   and
   last like 'Smith'
   )
 or
 salary > 20000.0
)
```

Bitte beachten Sie auch:

sqltest-Tag

sqltest: Formatiert SQL-Bedinungstests

Das `sqltest`-Tag fügt einen Bedingungstest in einen SQL-Code ein. Es testet eine Spalte gegen eine Variable. Dieses Tag wird in SQL-Methoden verwendet.

Syntax

`sqltest` Tag Syntax:

```
<dtml-sqltest Variable|expr="Variablenausdruck">
```

Das `sqltest`-Tag besteht aus einem Element. Es fügt einen SQL-Bedingungstest ein. Es wird in SQL-Abfragen verwendet. Das `sqltest`-Tag filtert die eingegebene Variable korrekt. Die genannte Variable oder der Variablenausdruck wird gegen eine SQL-Spalte getestet, indem die beschriebene Vergleichsoperation durchgeführt wird.

Attribute

type=string
: Der Typ der Variable. Gültige Typen umfassen: `string`, `int`, `float` und `nb`. `nb` bedeutet "nicht-leerer" String und sollte statt `string` verwendet werden, es sei denn, Sie suchen nach Leer-Werten. Das type-Attribut wird benötigt und wird verwendet, um die eingegebene Variable korrekt zu filtern.

column=string
: Der Name der SQL-Spalte, gegen die getestet wird. Dieses Attribut hat als Defaultwert den Namen der Variable.

multiple=boolean
: Ist dieser Wert true, kann die Variable eine Sequenz von Werten sein, die gegen die Spalte getestet werden können.

optional=boolean
: Ist dieser Wert true, dann ist der Test optional und wird nicht durchgeführt, wenn die Variable leer ist oder nicht existiert.

op=string
: Der Vergleichsoperator. Gültige Vergleichsoperatoren umfassen:
eq
equal to (genauso gross wie)
gt
greater than (grösser als)
lt
less than (kleiner als)
ne
not equal to (nicht gleich)
ge
greater than or equal to (grösser als oder gleich)
le
less than or equal to (kleiner als oder gleich)

  Der Defaultwert des Vergleichsoperators ist equal to. Wird der Vergleichsoperator nicht erkannt wird der Defaultwert verwendet. So können Sie Vegleiche wie `like` einsetzen.

Beispiele

Grundlegende Verwendung:

```
select * from employees
  where <dtml-sqltest name type="nb">
```

Ist die `name`-Variable auf `Bob` gesetzt, so wird folgendes berechnet:

```
select * from employees
  where name = 'Bob'
```

Mehrere Werte:

```
select * from employees
  where <dtml-sqltest empid type=int multiple>
```

Ist die `empid`-Variable auf `(12,14,17)` gesetzt, so wird folgendes berechnet:

```
select * from employees
  where empid in (12, 14, 17)
```

Bitte beachten Sie auch:

sqlgroup-Tag

sqlvar-Tag

sqlvar: Fügt SQL-Variablen ein

Das `sqlvar`-Tag fügt auf sicherem Weg Variablen in SQL-Code ein. Dieses Tag wird in SQL-Methoden verwendet.

Syntax

`sqlvar` Tag Syntax:

```
<dtml-sqlvar Variable|expr="Variablenausdruck">
```

Das `sqlvar`-Tag besteht nur aus einem Element. Wie auch das `var`-Tag sucht das `sqlvar`-Tag eine Variable und fügt sie ein. Anders jedoch als beim var-Tag, werden hier die Formatierungsoptionen auf den SQL-Code zugeschnitten.

Attribute

type=string
: Der Typ der Variable. Gültige Typen umfassen: `string`, `int`, `float` und `nb`. `nb` bedeutet "nicht-leerer" String und sollte statt `string` verwendet werden, es sei denn, Sie wollen leere Strings verwenden. Das type-Attribt wird benötigt und wird verwendet, um die eingefügte Variable korrekt zu filtern.

optional=boolean
: Ist der Wert auf true gesetzt und die Variable ist null oder existiert nicht, wird nichts eingefügt.

Beispiele

Grundlegende Verwendung:

```
select * from employees
```

```
where name=<dtml-sqlvar name type="nb">
```

Dieser SQL-Befehl gibt die `name`-String-Variable zurück.

Bitte beachten Sie auch:

sqltest-Tag

tree: Stellt eine Baumstruktur dar

Das `tree`-Tag stellt eine dynamische Baumstruktur dar, indem es Zope-Objekte abfägt.

Syntax

`tree` Tag Syntax:

```
<dtml-tree [VariableName|expr="Variablenausdruck"]>
</dtml-tree>
```

Das `tree`-Tag ist ein Block-Tag. Es berechnet eine dynamische Baumstruktur in HTML. Die Wurzel dieser Baumstruktur wird über einen Variablennamen oder Ausdruck definiert, falls vorhanden, andernfalls wird das aktuelle Element benutzt. Der `tree`-Block wird für jeden Punkt in der Baumstruktur berechnet, wobei der aktuelle jeweils in den DTML-Namensraum eingefügt wird.

Der Tree-Status wird über HTTP-Cookies gesetzt, diese müssen also aktiviert sein, damit der Baum funktionieren kann. Auch können Sie nur einen Baum pro Seite haben.

Attribute

branches=string
: Findet Äste des Baumes, indem die genannte Methode aufgerufen wird. Die Defaultmethode ist `tpValues`, die von den meisten Zope-Objekten unterstützt wird.

branches_expr=string
: Findet Äste, indem der Ausdruck ausgewertet wird.

id=string
: Der Name einer Methode oder ID, um den Status des Baumes herauszufinden. Der Defaultwert ist hier `tpId`, was von den meisten Zope-Objekten unterstützt wird. Dieses Attribut ist nur für die fortgeschrittene Anwendung gedacht.

url=string
: Der Name einer Methode oder eines Attributes um die URLs von Elementen des Baumes zu bestimmen. Der Defaultwert ist hier `tpURL`, was von den meisten Zope-Objekten unterstützt wird. Dieses Attribut ist nur für die fortgeschrittene Anwendung gedacht.

leaves=string
: Der Name eines DTML-Dokumentes oder einer Methode, um Endknoten zu finden. Bitte beachten Sie: dieses Dokument sollte mit `<dtml-var standard_html_header>`

beginnen und mit `<dtml-var standard_html_footer>` enden, um eine saubere Darstellung im Baum zu gewährleisten.

header=string
: Der Name eines DTML-Dokuments oder einer Methode, das/die vor voll dargestellten Knoten angezeigt wird. Wird der Header nicht gefunden, wird er übergangen.

footer=string
: Der Name eines DTML Dokuments oder einer Methode, das/die nach voll dargestellten Knoten angezeigt wird. Wird der Footer nicht gefunden, wird er übergangen.

nowrap=boolean
: Ist dieser Wert auf true gesetzt, werden Knoten auf den verfügbaren Platz zugeschnitten, statt sie zu wrappen.

sort=string
: Sortiert die Äste anhand des Namensattributes.

reverse
: Dreht die Reihenfolge des Astes um.

assume_children=boolean
: Geht davon aus, dass ein Knoten weitere Verzweigungen hat. Dies ist nützlich, wenn das Suchen und Abfragen von Endknoten bzw. Verzweigungen zu aufwändig wäre. Als Ergebnis werden Plus-Boxen neben alle Knoten gezeichnet.

single=boolean
: Erlaubt die Vollansicht genau eines Astes. Wird ein neuer Ast in die Vollansicht gebracht, werden alle anderen wieder geschlossen.

skip_unauthorized
: Übergeht Knoten, die der Benutzer nicht betrachten darf, ohne eine Fehlermeldung auszugeben.

urlparam=string
: Ein Abfragestring, der in den Vollansicht- und Indexsicht-Links der Oberflächenelemente enthalten ist. Dieses Attribut ist nur für die fortgeschrittene Anwendung gedacht.

prefix=string
: Stellt Versionen der Tag-Variablen zur Verfügung, die mit dieser Vorsilbe statt "tree" beginnen und Unterstriche(_) statt Bindestriche (-) benutzen. Die Vorsilbe muss mit einem Buchstaben beginnen und darf nur alphanumerische Zeichen und Unterstriche (_) enthalten.

Tag-Variablen

tree-item-expanded
: Ist true, wenn der aktuelle Knoten in der Vollansicht ist.

tree-item-url
: Die URL des aktuellen Knotens.

tree-root-url
: Die URL des Wurzelknotens.

tree-level
: Die Tiefe des aktuellen Knotens. Top-Level-Knoten haben eine Tiefe von 0.

tree-colspan
: Die Nummer der Tiefenebene, in der der Baum berechnet wird. Diese Variable kann zusammen mit der `tree-level` Variable benutzt werden, um Zeilen und Spalten sowie Einstellungen in eine Baum-Tabelle einzufügen.

tree-state

Der Status des Baumes, ausgedrückt in einer Liste von IDs und Sub-Listen von IDs. Diese Variable ist nur für die fortgeschrittene Anwendung gedacht.

Tag Control-Variablen

Sie können das tree-Tag kontrollieren, indem Sie diese Variablen setzen.

expand_all
: Wird diese Variable auf true gesetzt, wird der komplette Baum in die Vollansicht gebracht.

collapse_all
: Wird diese Variable auf true gesetzt, wird der komplette Baum in die Index-Sicht gebracht.

Beispiele

Stellt einen Baum dar, der seine Wurzel im aktuellen Objetk hat:

```
<dtml-tree>
  <dtml-var title_or_id>
</dtml-tree>
```

Stellt einen Baum dar, der seine Wurzel in einem anderen Objekt hat, in dem die Methode custom branches benutzt wird:

```
<dtml-tree expr="folder.object" branches="objectValues">
  Node id : <dtml-var getId>
</dtml-tree>
```

try: Exceptionbehandlung

Das `try`-Tag ermöglicht die Exceptionbehandlung in DTML, es spiegelt Pythons `try/except`- und `try/finally`-Konstrukte wider.

Syntax

Das `try`-Tag hat zwei verschiedene Syntaxformen, nämlich `try/except/else` und `try/finally`.

`try/except/else`-Syntax:

```
<dtml-try>
<dtml-except [ExceptionName] [ExceptionName]...>
...
[<dtml-else>]
</dtml-try>
```

Das `try`-Tag umschliesst einen Block, in dem Exceptions abgefangen und behandelt werden können. Es kann ein oder mehrere `except`-Tags geben, die Null oder mehr Exceptions behandeln. Spezifiziert ein `except`-Tag keine Exception, behandelt es alle.

Wird eine Exception ausgelöst, springt das Programm zum ersten `except`-Tag, das die Exception behandelt. Gibt es kein `except`-Tag, um die Exception zu behandeln, wird sie wie üblich ausgelöst.

Wird keine Exception ausgelöst und es gibt ein `else`-Tag, so wird dieses nach dem `try`-Tag ausgeführt.

Sowohl `except`- als auch `else`-Tag sind optional.

`try/finally`-Syntax:

```
<dtml-try>
<dtml-finally>
</dtml-try>
```

Das `finally`-Tag kann nicht im selben `try`-Block benutzt werden wie die `except`- und `else`-Tags. Ist ein `finally`-Tag vorhanden, wird sein Block in jedem Fall ausgeführt, egal, ob im `try`-Block eine Exception ausgelöst wird oder nicht.

Attribute

except
: Null oder mehr Exceptionnamen. Werden keine Exceptions aufgelistet, behandelt das except-Tag alle Exceptions.

Tag-Variablen

Im `except`-Block werden folgende Variablen definiert.

error_type
: Der Typ der Exception.

error_value
: Der Wert der Exception.

error_tb
: Das Traceback (die Zurückverfolgung).

Beispiele

Einen mathematischen Fehler abfangen:

```
<dtml-try>
<dtml-var expr="1/0">
<dtml-except ZeroDivisionError>
You tried to divide by zero.
```

```
</dtml-try>
```

Informationen über die behandelte Exception zurückgeben:

```
<dtml-try>
<dtml-call dangerousMethod>
<dtml-except>
An error occurred.
Error type: <dtml-var error_type>
Error value: <dtml-var error_value>
</dtml-try>
```

Die Benutzung von finally, um sicherzustellen, dass "aufgerämt wird", egal, ob ein Fehler aufgetreten ist oder nicht:

```
<dtml-call acquireLock>
<dtml-try>
<dtml-call someMethod>
<dtml-finally>
<dtml-call releaseLock>
</dtml-try>
```

Bitte beachten Sie auch:

raise-Tag

Python Tutorial: Errors and Exceptions

Python Built-in Exceptions

unless: Testet eine Bedingung

Das `unless`-Tag stellt einen Shortcut zur Verfügung, um negative Bedingungen zu testen. Für ausführliches Testen von Bedingungen benutzen Sie das `if`-Tag.

Syntax

`unless` Tag Syntax:

```
<dtml-unless BedingungsVariable|expr="Bedingungsausdruck">
</dtml-unless>
```

Das `unless`-Tag ist ein Block-Tag. Gelangt der Wert der Bedingungsvariable oder des Ausdrucks zu false, wird der beinhaltete Block ausgeführt. Wie beim `if`-Tag, werden Variablen, die nicht erreichbar sind, als false angesehen.

Beispiele

Eine Variable testen:

```
<dtml-unless testMode>
  <dtml-call dangerousOperation>
</dtml-unless>
```

Der Block wird ausgeführt, wenn `testMode` nicht existiert oder aber false ist.

Bitte beachten Sie auch:

if-Tag

var: Fügt eine Variable ein

Mithilfe des `var`-Tags können Sie eine Variable in den DTML-Output einfügen.

Syntax

`var` Tag Syntax:

```
<dtml-var Variable|expr="Ausdruck">
```

Das `var`-Tag besteht nur aus einem Element. Es findet eine Variable, indem es den DTML-Namensraum durchsucht, der normalerweise aus aktuellem Objekt, dem Behälter des aktuellen Objektes und schliesslich der Web-Anfrage besteht. Wird die Variable gefunden, wird sie in den DTML-Output integriert. Wird sie nicht gefunden, löst Zope einen Fehler aus.

`var` Tag Instanz-Syntax:

```
&dtml-variableName;
```

Die Instanz-Syntax ist ein Shortcut, der die Variable einfügt und als HTML zitiert. Dies ist nützlich, wenn Sie Variablen in HTML-Tags einfügen.

`var`-Tag Instanz-Syntax mit Attributen:

```
&dtml.attribute1[.attribute2]...-variableName;
```

Bis zu einem gewissen Grad können Sie Attribute mit der Instanz-Syntax bestimmen. Sie können Null oder mehr Attribute, die von Perioden begrenzt werden, einfügen. Sie können keine Argumente für Attribute zur Verfügung stellen, die die Instanz-Syntax benutzen. Wenn

Sie null oder mehr Attribute zur Verfügung stellen, wird die Variable nicht automatisch in HTML zitiert. So können Sie HTML-Zitate vermeiden: `&dtml.-variableName;`.

Attribute

html_quote
: Wandelt Characters, die in HTML eine besondere Bedeutung haben, in HTML-Character-Instanzen um.

missing=string
: Spezifiziert einen Defaultwert für den Fall, das Zope die Variable nicht finden kann.

fmt=string
: Formatiert eine Variable. Zope bietet einige eingebaute Formate einschliesslich C-style Format Strings an. Für weitere Informationen über C-style Format Strings beachten Sie bitte auch: Python Library Reference Hat der Format String kein eingebautes Format, wird angenommen, dass es sich um eine Methode des Objektes handelt, und wird aufgerufen.
  whole-dollars
  Formatiert die Variable zu Dollar-Zeichen.
  dollars-and-cents
  Formatiert die Variable zu Dollar- und Cent-Zeichen.
  collection-length
  Die Länge der Variable, wobei angenommen wird, dass es sich um eine Sequenz handelt.
  structured-text
  Formatiert die Variable zu Structured Text. Für weitere Informationen über Structured Text beachten Sie bitte auch: Structured Text How-To auf der Zope.org Webseite.

null=string
: Ein Defaultwert, der benutzt wird, wenn die Variable None ist.

lower
: Wandelt Grossbuchstaben in Kleinbuchstaben um.

upper
: Wandelt Kleinbuchstaben in Grossbuchstaben um.

capitalize
: Schreibt den ersten Buchstaben des Wortes gross.

spacify
: Wandelt Unterstriche in Leerzeichen um.

thousands_commas
: Fügt alle drei Stellen links vom Dezimalpunkt ein Komma in eine Zahl ein, z.B. wird aus `12000` anschliessend `12,000`.

url
: Fügt die URL des Objektes ein, indem seine `absolute_url`-Methode aufgerufen wird.

url_quote
: Wandelt Buchstaben, die eine besondere Bedeutung in URLs haben, in HTML-Character-Instanzen um.

url_quote_plus
: URL-Anführungszeichen wie `url_quote` aber auch Leerzeichen werden in Pluszeichen umgewandelt.

sql_quote
: Wandelt einzelne Anführungszeichen in normale Anführungszeichen um. Die wird benötigt, um Werte richtig in SQL-Strings einzufügen.

newline_to_br

Wandelt Zeilenendenzeichen (einschliesslich des Zeilenumbruchs) in HTML-Break-Tags um.

size=arg
: Schneidet die Variable an der gegebenen Länge ab. (Bitte beachten Sie: ist in der zweiten Hälfte der abgeschnittenen Variable ein Leerzeichen vorhanden, wird der String in dieser Hälfte nochmals zerschnitten).

etc=arg
: Definiert einen String, der an das Ende eines anderen Strings, der zerschnitten wurde, angehängt wird (indem das `size`-Attribut, das hier weiter oben in der Liste steht, angewendet wird). Defaultmässig ist dies . . .

Beispiele

Eine einfache Variable in ein Dokument einfügen:

```
<dtml-var standard_html_header>
```

truncate:

```
<dtml-var colors size=10 etc=", etc.">
```

wird folgenden Output ergeben, wenn *colors* der String 'red yellow green' ist:

```
red yellow, etc.
```

C-style String-Formatierung:

```
<dtml-var expr="23432.2323" fmt="%.2f">
```

berechnet:

```
23432.23
```

Die Variable *link*, innerhalb eines HTML `A`-Tags mit der Instanz-Syntax:

```
<a href="&dtml-link;">Link</a>
```

Einen Link mit Hilfe der Instanz-Syntax mit Attributen in das Dokument `doc` einfügen:

```
<a href="&dtml.url-doc;"><dtml-var doc fmt="title_or_id"></a>
```

Dies erzeugt einen HTML-Link zu einem Objekt, indem seine URL und sein Titel benutzt werden. Dieses Beispiel ruft die `absolute_url`-Methode des Objekets für die URL (es wird das `url`-Attribut benutzt) und seine `title_or_id`-Methode für den Titel auf.

with: Kontrolliert die DTML-Variablen-Suche

Das `with` Tag fügt ein Objekt in den DTML-Namensraum ein. Variablen werden zuerst im eingefügten Objekt gesucht.

Syntax

`with` Tag Syntax:

```
<dtml-with Variable|expr="Ausdruck">
</dtml-with>
```

Das `with`-Tag ist ein Block-Tag. Es fügt den genannten Variablenausdruck für die Dauer des `with`-Blocks in den DTML-Namensraum ein. Hierdurch werden Namen zuerst im eingefügten Objekt gesucht.

Attribute

only
: Begrenzt den DTML-Namensraum so, dass nur der im `with`-Tag definierte benutzt wird.

mapping
: Zeigt an, dass die Variable oder der Ausdruck ein Oberflächenobjekt ist. Dies stellt sicher, dass Variablen korrekt im Objerflächenobjekt gesucht werden.

Beispiele

Eine Variable im REQUEST suchen:

```
<dtml-with REQUEST only>
  <dtml-if id>
    <dtml-var id>
  <dtml-else>
    'id' was not in the request.
  </dtml-if>
</dtml-with>
```

Die erste Verzweigung in den DTML-Namensraum einfügen:

```
<dtml-with expr="objectValues()[0]">
  First child's id: <dtml-var id>
</dtml-with>
```

Bitte beachten Sie auch:

let tag

Zopebuch: Inhaltsverzeichnis

Diese Referenz beschreibt die Schnittstellen der wichtigsten Zope-Objekte. Die Referenz ist vor allem nützlich, wenn Sie DTML-, Perl- und Python-Skripte schreiben, die Zope-Objekte erstellen und verändern.

Modul `AccessControl`

AccessControl: Sicherheitsfunktionen und -Klassen

Die Funktionen und Klassen dieses Moduls können in Python-basierenden Skripten und Seitenvorlagen verwendet werden.

Klasse `SecurityManager`

Ein Sicherheitsmanager (SecurityManager) stellt Methoden bereit, mit denen Zugriffe kontrolliert und ausführbare Inhalte sowie Richtlinien verwaltet werden können.

`calledByExecutable(self)`

Gibt einen Boolean zurück, der anzeigt, ob dieser Kontext von einer ausführbaren Datei aufgerufen wurde.

Berechtigung:
    Steht immer zur Verfügung.

`validate(accessed=None, container=None, name=None, value=None, roles=None)`

Validiert den Zugriff.

Argumente:

accessed
    Das Objekt, auf das zugegriffen wurde
container
    Das Objekt, in dem der Wert gefunden wurde
name
    Der Name, mit dem auf den Wert zugegriffen wurde
value
    Der Wert, der durch den Zugriff erhalten wurde.
roles
    Die Rollen des Objektes falls bereits bekannt.

Die Argumente können als Schlüsselargumente übergeben werden. Einige der Argumente können vernachlässigt werden, jedoch kann die Sicherheitsrichtlinie in einigen Fällen den Zugriff verweigern, wenn Argumente vernachlässigt werden. Es ist daher am besten, alle Werte, die bekannt sind, anzugeben

Berechtigung:
    Steht immer zur Verfügung.

`checkPermission(self, permission, object)`

Überprüft, ob der Sicherheitskontexts die gegebene Erlaubnis im Zusammenhang mit dem gegebenen Objekt erlaubt.

Berechtigung:
    Steht immer zur Verfügung.

`getUser(self)`

Gibt den aktuellen authentifizierten Benutzer zurück. Bitte beachten Sie hierzu auch die `AuthenticatedUser`-Klasse.

Berechtigung:
    Steht immer zur Verfügung.

`validateValue(self, value, roles=None)`

Ausreichend für gänige Fälle der Wert-Validierung.

Berechtigung:
    Steht immer zur Verfügung.

`def getSecurityManager()`

Gibt den Security-Manager zurück. Bitte beachten Sie hierzu auch die `SecurityManager`-Klass.

Modul `AuthenticatedUser`

Klasse `AuthenticatedUser`

Diese Schnittstelle muss von Objekten unterstützt werden, die durch Benutzer-Validierung zurückgegeben und für die Zugriffskontrolle benutzt werden.

`getUserName()`

Gibt den Namen eines Benutzers zurück.

Berechtigung:
    Steht immer zur Verfügung.

`getId()`

Gibt die ID des Benutzers zurück. Die ID kann von Python benutzt werden, um Benutzer aus der UserDatabase auszulesen.

Berechtigung:
Steht immer zur Verfügung.

`has_role(roles, object=None)`

Gibt true zurück, wenn der Benutzer mindestens eine Rolle aus einer Liste von Rollen hat, optinal im Kontext eines Objekts.

Berechtigung:
Steht immer zur Verfügung.

`getRoles()`

Gibt eine Liste mit den Rollen des Benutzers zurück.

Berechtigung:
Steht immer zur Verfügung.

`has_permission(permission, object)`

Gibt true zurück, wenn der User Rechte am Objekt hat.

Berechtigung:
Steht immer zur Verfügung.

`getRolesInContext(object)`

Gibt die Liste der Rollen zurück, die dem Benutzer zugeschrieben sind, inklusive lokaler Rollen, die ihm im Kontext eines Objekts zugeschrieben sind.

Berechtigung:
Steht immer zur Verfügung.

`getDomains()`

Gibt die Liste der Domain-Einschränkungen für einen Benutzer zurück.

Berechtigung:
Steht immer zur Verfügung.

Modul `DTMLDocument`

Klasse `DTMLDocument(ObjectManagerItem, PropertyManager)`

Ein DTML-Dokument ist ein Zope-Objekt, das DTML beinhaltet und ausführt. Es ist nützlich, um Webseiten darzustellen.

`manage_edit(data, title)`

Ändert das DTML-Dokument, indem es seinen Inhalt mit `data` ersetzt und seinen Titel ändert.

Das data-Argument kann eine Datei oder ein String sein.

Berechtigung:
`DTML-Dokumente verändern`

`document_src()`

Gibt den ungerenderten Quelltext des DTML-Dokuments zurück.

Berechtigung:
`Management-Screens betrachten`

`__call__(client=None, REQUEST={}, RESPONSE=None, **kw)`

Wird ein DTML-Dokument aufgerufen, so wird sein DTML-Inhalt interpretiert. Die Methode gibt das Ergebnis der Interpretation zurück, das jedes Objekt sein kann.

Um seine Aufgabe zu erfüllen, muss das DTML-Dokument oft verschiedene Namen in Objekten auflösen. Beispielsweise: wird der Code '<dtml-var spam>' ausgeführt, so versucht die DTML-Maschine, den Namen `spam` aufzulösen.

Um Namen aufzulösen, muss dem Dokument ein Namensraum zugänglich sein, in dem es die Namen suchen kann. Die kann über verschiedene Möglichkeiten erreicht werden:

- Indem ein `client`-Objekt übergeben wird -- Wird das Argument `client` übergeben, werden Namen als Attribute des Arguments gesucht.
- Indem eine `REQUEST`-Funktion übergeben wird -- Wird das Argument `REQUEST` übergeben, so werden Namen als Elemente des Arguments gesucht. Ist das Objekt keine Funktion, wird ein TypeError ausgegeben, sobald eine Namenssuche gestartet wird.
- Indem Schlüsselwort-Argument übergeben werden -- Namen und ihre Werte können dem Dokument als Schlüsselwort-Argumente übergeben werden.

Der Namensraum, der einem DTML-Dokument gegeben wird ist das Gemisch dieser drei Methoden. Sie können viele von ihnen übergeben oder aber auch gar keine. Namen werden zuerst in den Schlüsselwort-Argumenten gesucht, dann im Client, und schliesslich in der Funktion.

Ein DTML-Dokument gibt sich selber stillschweigend als Client-Argument als Zusatz zum spezifizierten Client weiter, wodurch Namen auch im DTML-Dokument selber gesucht werden.

Einen Namensraum an ein DTML-Dokument weitergeben wird oft als Versorgung des Dokuments mit einem *Kontext* bezeichnet.

DTML-Dokumente können auf drei Arten aufgerufen werden.

Aus DTML

Ein DTML-Dokument kann aus einem anderen DTML-Dokument oder einer anderen DTML-Methode heraus aufgerufen werden:

```
<dtml-var standard_html_header>
  <dtml-var aDTMLDocument>
<dtml-var standard_html_footer>
```

In diesem Beispiel wird das Dokument `aDTMLDocument` anhand seines Namens von einem anderen DTML-Objekt aus aufgerufen. Die Aufrufsmethode gibt den Wert `this` als Client-Argument und den aktuellen DTML-Namensraum als REQUEST-Argument weiter. Die Ausführung des Codes oben bewirkt das gleiche wie folgende Verwendung in einem DTML-Python-Ausdruck:

```
<dtml-var standard_html_header>
  <dtml-var "aDTMLDocument(_.None, _)">
<dtml-var standard_html_footer>
```

Aus Python

Produkte, externe Methoden und Skripte können ein DTML-Dokument auf gleichem Wege aufrufen wie ein DTML-Dokument aus einem Python-Ausdruck in DTML, was folgendes Beispiel verdeutlichen soll.

Anhand des Publishers (Veröffentlicher)

Wird die URL eines DTML-Dokumentes von Zope abgerufen, wird das DTML-Dokument anhand des Veröffentlichers aufgerufen. Das REQUEST-Objekt wird als zweites Argument an das Dokument übergeben.

Berechtigung:
`View`

`get_size()`

Gibt die Grösse des ungerenderten Quelltextes des DTML-Dokumentes in Bytes zurück.

Berechtigung:
`View`

ObjectManager Constructor

`manage_addDocument(id, title)`

Fügt ein DTML-Dokument zum aktuellen Objekt-Manager hinzu.

Modul `DTMLMethod`

Klasse `DTMLMethod(ObjectManagerItem)`

Eine DTML-Methode ist ein Zope-Objekt, das DTML beinhaltet und ausführt. Es kann als Vorlage dienen, um andere Objekte anzuzeigen. Es kann auch kleine Inhalte haben, die in andere DTML-Dokumente oder -Methoden eingefügt werden.

Die ID der DTML-Methode ist über die `document_id`-Variable erhältlich und der Titel über die `document_title`-Variable.

`manage_edit(data, title)`

Verändert die DTML-Methode, indem ihr Inhalt durch `data` ersetzt und der Titel geändert wird.

Das data-Argument kann ein File oder ein String sein.

Berechtigung:
`DTML-Methoden verändern`

`document_src()`

Gibt den ungerenderten Quelltext der DTML-Methode zurück.

Berechtigung:
`Management-Screens betrachten`

`__call__(client=None, REQUEST={}, **kw)`

Eine DTML-Methode aufzurufen, veranlasst die Methode dazu, enthaltene DTML zu interpretieren. Die Methode gibt das Ergebnis der Interpretation als Objekt zurück, es gibt hierbei keine Einschränkungen, welches Objekt.

Um ihre Aufgabe auszuführen, müssen DTML-Methoden oft verschiedene Namen in Objekten auflösen. Beispielsweise versucht die DTML-Maschine, wenn Code '<dtml-var spam>' ausgeführt wird, den Namen `spam` aufzulösen.

Um Namen aufzulösen, muss der Methode ein Namensraum übergeben werden, in dem sie sie suchen kann. Dies kann auf verschiedene Art und Weise erreicht werden:

- Indem ein `client`-Objekt weitergegeben wird -- Wird das Argument `client` weitergegeben, werden Namen als Attribute des Arguments gesucht.
- Indem eine `REQUEST`-Funktion weitergeben wird -- Wird das `REQUEST`-Argument weitergegeben, werden Namen als Elemente des Arguments gesucht. Ist das Objekt keine Funktion, wird ein TypeError ausgegeben, sobald eine Namenssuche gestartet wird.
- Indem Schlüsselwort-Argumente weitergegeben werden -- Namen und ihre Werte können als Schlüsselwort-Argumente an die Methode weitergegeben werden.

Der Namensraum, der einer DTML-Methode gegeben wird, ist die Mischung dieser drei Methoden. Sie können eine beliebige Anzahl von ihnen oder aber auch garkeine weitergeben.

Namen werden zuerst anhand der Schlüsselwort-Argumente gesucht, dann im Client und schliesslich in der Funktion.

Anders als DTML-Dokumente suchen DTML-Methoden Namen nicht in ihrem eigenen Instanz-Wörterbuch.

Einen Namensraum an eine DTML-Methode weiterzugeben wird oft auch als Versorgung einer Methode mit einem *Kontext* bezeichnet.

DTML-Methoden können auf drei verschiedene Arten aufgerufen werden:

Aus DTML

Eine DTML-Methode kann aus einer anderen DTML-Methode oder einem DTML-Dokument heraus aufgerufen werden:

```
<dtml-var standard_html_header>
  <dtml-var aDTMLMethod>
<dtml-var standard_html_footer>
```

In diesem Beispiel wird die Methode `aDTMLMethod` anhand ihres Namens von einem anderen DTML-Objekt heraus aufgerufen. Die Aufrufsmethode gibt den Wert `this` als Client-Argument und den aktuellen DTML-Namensraum als REQUEST-Argument weiter. Der Code oben bewirkt in der Ausführung das selbe wie folgender DTML-Ausdruck in Python:

```
<dtml-var standard_html_header>
  <dtml-var "aDTMLMethod(_.None, _)">
<dtml-var standard_html_footer>
```

Aus Python

Produkte, externe Methoden und Skripte können eine DTML-Methode auf dem selben Weg aufrufen wie eine DTML-Methode aus einem Python-Ausdruck in DTML heraus aufgerufen wird, wie folgendes Beispiel zeigt.

Anhand des Veröffentlichers

Wird die URL einer DTML-Methode von Zope abgerufen, wird die DTML-Methode anhand ihres Veröffentlichers aufgerufen. Das REQUEST-Objekt wird als zweites Argument an die Methode weitergegeben.

Berechtigung:
`View`

`get_size()`

Gibt die Grösse des ungerenderten Quelltextes der DTML-Methode in Bytes zurück.

Berechtigung:

View

ObjectManager Constructor

`manage_addDTMLMethod(id, title)`

Fügt eine DTML-Methode zum aktuellen Objekt-Manager hinzu.

Modul `DateTime`

Klasse `DateTime`

Das DateTime-Objekt stellt eine Schnittstelle zur Verfügung, mit der mit Daten und Zeiten in verschiedenen Formaten gearbeitet werden kann. DateTime stellt weiterhin Methoden f&uum;r Kalender-Arbeiten, Datums- und Zeit-Arithmetik und -Formatierung zur Verfügung.

DateTime-Objekte repräsentieren Zeitinstanzen und stellen Schnittstellen f˙r die Kontrolle der Darstellung zur Verfügung, ohne den absoluten Wert des Objekts zu beeinflussen.

DateTime-Objekte können aus einer grossen Auswahl an String- und numerischen Daten erstellt werden oder aus anderen DateTime-Objekten berechnet werden. DateTime unterstützt die Fähigkeit, Darstellungen in viele geläfige Zeitzonen zu konvertieren, sowie die Fähigkeit, ein DateTime-Objekt im Kontext der gegebenen Zeitzone zu erstellen.

DateTime-Objekte stellen teilweise numerisches Verhalten zur Verfügung:

- Zwei DateTime-Objekte können subtrahiert werden, um die Zeit in Tagen zwischen beiden Daten zu erhalten.
- Ein DateTime-Objekt und eine positive oder negative Zahl können addiert werden um ein neues DateTime-Objekt zu erhalten, das die gegebene Zahl an Tagen später als das ursprüngliche DateTime-Objekte darstellt.
- Eine positive oder negative Zahl kann von einem DateTime-Objekt abezogen werden, um ein neues DateTime-Objekt zu erhalten, das die angegebene Anzahl an Tagen früher darstellt als das usrprüngliche DateTime-Objekt.

DateTime-Objekte können in integer-, long- oder float-Zahlen von Tagen ab dem 1. Januar 1901 umgewandelt werden, indem die Standard int-, long- und float-Funktionen genutzt werden (Kompatibilitätshinweis: int, long und float geben die Zahl der Tage seit 1901 in GMT statt in Ortszeit zurück). DateTime-Objekte stellen auch Zugang zu ihrem Wert in Form eines float-Formats bereit, das mit dem Python Zeitmodul verwendet werden kann, sofern der gegebene Wert des Objektes im Rahmen des epochen-basierenden Zeitmoduls liegt.

Ein DateTime-Objekt sollte als unveränderlich angesehen werden; alle Umwandlungen und numerischen Operationen geben ein neues DateTime-Objekt zurück statt das gegebene zu verändern

Ein DateTime-Objekt erhält seinen Wert immer als absolute UTC-Zeit und wird im Kontext einer bestimmten Zeitzone dargestellt, die auf den Argumenten basiert, die benutzt wurden,

um das Objekt zu erstellen. Die Methode eines DateTime-Objekts gibt Werte zurück, die auf dem Zeitzonen-Kontext basieren.

Bitte beachten Sie, dass in jedem Fall die lokale Rechnerzeit für die Darstellung benutzt wird, falls keine Zeitzone angegeben wird.

DateTime-Objekte können unter Benutzung von null bis sieben Argumenten erstellt werden.

- Wird die Funktion ohne irgendwelche Argumente aufgerufen, wird die aktuelle Zeit/Datum-Kombination zurückgegeben, die in der Zeitzone des Rechners dargestellt wird.
- Wird die Funktion mit einem einzelnen String-Argument, welches den Namen einer Zeitzone darstellt, aufgerufen, wird ein Objekt zurückgegeben, das die aktuelle Zeit in der angegebenen Zeitzone darstellt.
- Wird die Funktion mit einem einzelnen String-Argument aufgerufen, das eine gültige Datum/Zeit-Kombination darstellt, wird ein Objekt, das diese Kombination darstellt, zurückgegeben.

  Als grundsätzliche Regel ist festzuhalten, dass jede Datum/Zeit-Darstellung die in Nordamerika erkannt werden kann und eindeutig ist, akzeptiert wird. (Der Grund für dies Qualifikation ist, dass in Nordamerika ein Datum wie z.B. 2/1/1994 als 1. Februar 1994 interpretiert wird, während in einigen Teilen der Welt das selbe Datum als 2. Januar 1994 gelesen wird.) Ein Datum/Zeit-String besteht aus zwei Komponenten, einer Datums-Komponenten und einer optionalen Zeit-Komponenten, die von einem oder mehreren Leerzeichen getrennt werden. Wird die Zeit-Komponente vernachlässigt, wird 12:00 Uhr (AM) angenommen. Jeder anerkannte Zeitzonen-Name, der als letztes Element des Datum/Zeit-Strings angegeben wird, wird für die Berechnung des Datum/Zeit-Werts verwendet. (Wenn Sie ein DateTime mit dem String `Mar 9, 1997 1:45pm US/Pacific` erstellen, wird der Wert im Grunde genommen der selbe sein wenn Sie time.time() am angegebenen Datum und zur angegebenen Zeit auf einem Rechner in dieser Zeitzone ausführen:

```
            e=DateTime("US/Eastern")
#gibt die aktuellen Werte von Datum/Zeit in US/Eastern dargestellt
zurück.
            x=DateTime("1997/3/9 1:45pm")
#gibt die angegebene Zeit in lokaler Rechnerzeit dargestellt zurück.
            y=DateTime("Mar 9, 1997 13:45:00")
#y ist identisch mit x
```

  Die Datumskomponente besteht aus Jahres-, Tages- und Monatswerten. Der Jahreswert muss ein ein-, zwei- oder vierstelliger Integer sein. Wird ein ein- oder zweistelliger Wert verwendet, wird angenommen, dass das Jahr im 20. Jahrhundert ist. Der Monat kan ein Integer von 1 bis 12 sein, ein Monatsname oder eine Monatsabkürzung, wobei der Abkürzung optional ein Punkt folgen kann. Der Tag muss ein Integer von 1 bis zur Anzahl der Tage des Monats sein. Die Jahres-, Monats- und Tageswerte können durch Punkte, Bindestriche, Schrägstriche oder Leerzeichen getrennt sein. Weitere Leerzeichen sind zur Abgrenzung erlaubt. Jahres-, Monats- und Tageswerte können in beliebiger Reihenfolge angegeben werden, so lange es möglich

ist, die Komponenten zu unterscheiden. Sind alle drei Komponenten Zahlen kleiner als 13, wird eine Monats-Tages-Jahres-Anordnung angenommen.

Die Zeitkomponente besteht aus Stunden-, Minuten- und Sekundenwert, die durch Kommata getrennt werden. Der Stundenwert muss ein Integer von 0 bis inklusive 23 sein. Der Minutenwert muss ein Integer zwischen 0 und einschliesslich 59 sein. Der Sekundenwert kann ein Intergerwert zwischen 0 un einschliesslich 59.999 sein. Der zweite Wert oder sowohl Minuten- als auch Sekundenwert können ausgelassen werden. Die Zeit kann von einem am oder pm in Gross- oder Kleinbuchstaben gefolgt sein, hier wird eine 12-Stunden-Uhr angenommen.

- Wird die DateTime-Funktion mit einem einzelnen numerischen Argument aufgerufen, wird angenommen, dass die Zahl ein Gleitkommawert ist, wie der, der von time.time() zurückgegeben wird.

  Ein DateTime-Objekt, das den GMT-Wert des time.time()-Floats in lokaler Rechnerzeit darstellt wird zurückgegeben.

- Wird die Funktion mit zwei numerischen Argumenten aufgerufen, wird das erste als Jahr und das zweite als Ursprung in Tagen vom Ursprung des Jahres her angenommen. Es wird der Context der lokalen Rechnerzeit genutzt. Der Datums-Zeit-Wert, der zurückgegeben wird ist die gegebene Zahl an Tagen vom Beginn des Jahres an, dargestellt in der Zeitzone des lokalen Rechners. Der Ursprung kann positiv oder negativ sein. Zweistellige Jahreszahlen werden automatisch zum 20. Jahrhundert gezählt.
- Wird die Funktion mit zwei Argumenten aufgerufen, das erste hierbei ein Float, der die Zahl der Sekunden seit der Epoche in GMT (wie das Resultat von time.time()) darstellt, der zweite ein String, der die erkannte Zeitzone nennt, wird ein DateTime-Wert zurückgegeben, der in der gegebenen Zeitzone dargestellt wird:

```
import time
t=time.time()

now_east=DateTime(t,'US/Eastern')
# Zeit t, dargestellt als US/Eastern

now_west=DateTime(t,'US/Pacific')
# Zeit t, dargestellt als US/Pacific

# now_east == now_west
# Nur die Darstellungen sind unterschiedlich.
```

- Wird die Funktion mit drei oder mehr numerischen Argumenten aufgerfuen, wird das erste als Jahr, dargestellt als Integer, angenommen, das zweite als Integer-Monat und das dritte als Integer-Tag. Ist die Kombination der Werte nicht gültig, wird ein DateTimeError ausgegeben. Zweistellige Jahres werden automatisch im 20. Jahrhundert angesiedelt. Das vierte, fünfte und sechste Argument spezifizieren eine Zeit in Stunden, Minuten und Sekunden; Stunden und Minuten sollten positive Integers sein, der Sekundenwert muss eine positive Gleitkommazahl sein, bei all diesen Werten gilt der Defaultwert null. Ein optionaler String kann angegeben werden, um die Zeitzone zu spezifizieren (der Effekt ist hier der gleiche wie bei einem Ergebnis von time.time() auf diesem Rechner in der angegebenen Zeitzone).

Kann ein String-Argument, dass dem DateTime-Konstruktor übergeben wurde, nicht geparst werden, wird ein DateTime.SyntaxError ausgegeben. Ungültige Datums-, Zeit- oder Zeitzonenangaben verursachen einen DateTime.DateTimeError.

Die Modulfunktion Timezones() gibt eine Liste der Zeitzonen zurück, die vom DateTime-Modul erkannt werden. Bei der Erkennung von Zeitzonen-Namen muss nicht auf die Gross- bzw. Kleinschreibung geachtet werden.

strftime(format)

Gibt einen Datums-Zeit-String zurück, der über `format` formatiert wurde.

Bitte beachten Sie auch Pythons time.strftime-Funktion.

dow()

Gibt den Tag der Woche als Integer zurück, wobei Sonntag den Wert 0 hat.

Berechtigung:
Steht immer zur Verfügung.

aCommon()

Gibt einen String zurück, der den Wert des Objektes im Format: Mar 1, 1997 1:45 pm darstellt

Berechtigung:
Steht immer zur Verfügung.

h_12()

Gibt die 12-Stunden-Uhr-Darstellung der Stunde zurück.

Berechtigung:
Steht immer zur Verfügung.

Mon_()

Kompatibilität: siehe pMonth

Berechtigung:
Steht immer zur Verfügung.

HTML4()

Gibt das Objekt im Format, das in der HTML4.0-Spezifikation (ein ISO8601-Standard) definiert ist zurück.

Bitte beachten Sie auch: HTML 4.0 Specification

Daten sind Ausgaben im Format : YYYY-MM-DDTHH:MM:SSZ T und Z sind Literale. Die Zeit wird in UTC dargestellt.

Berechtigung:
Steht immer zur Verfügung.

greaterThanEqualTo(t)

Vergleicht ein DateTime-Objekt mit einem anderen DateTime-Objekt ODER einer Gleitkommazahl, wie die, die vom Python time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt einen Zeitwert darstellt, der grösser oder gleich ist als das angegebene DateTime-Objekt oder die time-Modul-Zeit. Wurde revidiert, um genauere Ergebnisse durch Vergleich von langen Integer-Millisekunden zu geben.

Berechtigung:
Steht immer zur Verfügung.

dayOfYear()

Gibt den Tag des Jahres im Kontext der Zeitzonendarstellung des Objektes zurück.

Berechtigung:
Steht immer zur Verfügung.

lessThan(t)

Vergleicht ein DateTime-Objekt mit einem anderen DateTime-Objekt ODEr einer Gleitkommazahl wie jener, die vom Python time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die kleiner ist als das angegebene DateTime-Objekt oder die time-Modul-Zeit. Wurde revidiert, um durch den Vergleich von Integer-Millisekunden genauere Ergebnisse zu erzielen.

Berechtigung:
Steht immer zur Verfügung.

AMPM()

Gibt den Zeit-String für ein Objekt für die nächste Sekunde zurück.

Berechtigung:
Steht immer zur Verfügung.

isCurrentHour()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die in die aktuelle Stunde fällt, auch hier wieder im Kontext der Zeitzonen-Darstellung des Objektes.

Berechtigung:
Steht immer zur Verfügung.

Month()

Gibt den vollen Monatsnamen zurück.

Berechtigung:
Steht immer zur Verfügung.

mm()

Gibt den Monat als zweistelligen String zurück.

Berechtigung:
Steht immer zur Verfügung.

ampm()

Gibt den passenden Zeit-Modifikator zurück (am oder pm).

Berechtigung:
Steht immer zur Verfügung.

hour()

Gibt die 24-Stunden Darstellung der Stunde zurück.

Berechtigung:
Steht immer zur Verfügung.

aCommonZ()

Gibt eine String-Darstellung des Wertes des Objektes im Format: Mar 1, 1997 1:45 pm US/Eastern zurück.

Berechtigung:
Steht immer zur Verfügung.

Day_()

Kompatibilität : siehe pDay

Berechtigung:
Steht immer zur Verfügung.

pCommon()

Gibt eine String-Darstellung des Wertes des Objektes im Format: Mar. 1, 1997 1:45 pm zurück.

Berechtigung:
Steht immer zur Verfügung.

minute()

Gibt die Minute zurück.

Berechtigung:
Steht immer zur Verfügung.

day()

Gibt den Tag als Integer zurück.

Berechtigung:
Steht immer zur Verfügung.

earliestTime()

Gibt ein neues DateTime-Objekt zurück, das die frühestmögliche Zeit (in ganzen Sekunden), die noch in den aktuellen Tag des Objektes fällt, wie immer im Kontext der Zeitzone des Objektes.

Berechtigung:
Steht immer zur Verfügung.

Date()

Gibt den Datums-String für das Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

Time()

Gibt den Zeit-Sring eines Objektes für die nächste Sekunde zurück.

Berechtigung:
Steht immer zur Verfügung.

isFuture()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die später ist als die Zeit des Aufrufes.

Permission
Always available

greaterThan(t)

Vergleicht das DateTime-Objekt mit einem anderen DateTime-Objekt ODER einer Gleitkommazahl wie jener, die vom Python-time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die gr&oum;sser ist als das spezifizierte DateTime-Objekt oder die time-Modul-Zeit. Wurde revidiert umd durch den Vergleich von Integer-Millisekunden genauere Ergebnisse zu erzielen.

Berechtigung:
Steht immer zur Verfügung.

TimeMinutes()

Gibt den Zeit-String f¨r ein Objekt zurück, ohne die Sekunden anzuzeigen.

Berechtigung:
Steht immer zur Verfügung.

yy()

Gibt das Kalenderjahr als zweistelligen String zurück.

Berechtigung:
Steht immer zur Verfügung.

isCurrentDay()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die in den aktuellen Tag fällt, wie immer im Kontext der Zeitzonen-Darstellung des Objekts.

Berechtigung:
Steht immer zur Verfügung.

dd()

Gibt den Tag als zweistelligen String zurück.

Berechtigung:
Steht immer zur Verfügung.

rfc822()

Gibt das Datum im RFB 822-Format zurück.

Berechtigung:
Steht immer zur Verfügung.

isLeapYear()

Gibt true zurück, wenn das aktuelle Jahr (im Kontext der Zeitzone des Objekts) ein Schaltjahr ist.

Berechtigung:
Steht immer zur Verfügung.

fCommon()

Gibt eine String-Darstellung des Wertes des Objekts im Format: March 1, 1997 1:45 pm zurück.

Berechtigung:
Steht immer zur Verfügung.

isPast()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, die früher ist als die Zeit des Aufrufs.

Berechtigung:
Steht immer zur Verfügung.

fCommonZ()

Gibt einen String zurück, der den Wert des Objektes im Format: March 1, 1997 1:45 pm US/Eastern darstellt.

Berechtigung:
Steht immer zur Verfügung.

timeTime()

Gibt Datum/Zeit als Gleitkommatzahl in UTC zurück, wobei das Format des Python time-Moduls verwendet wird. Bitte beachten Sie, dass es möglich ist, Datum/Zeit-Werte mit DateTime zu erstellen, die im time-Modul keine sinnvolle Bedeutung haben.

Berechtigung:
Steht immer zur Verfügung.

toZone(z)

Gibt ein DateTime-Objekt mit den Werten als aktuelles Objekt zurück und wird in der angegebenen Zeitzone dargestellt.

Berechtigung:
Steht immer zur Verfügung.

lessThanEqualTo(t)

Vergleicht dieses DateTime-Objekt mit einem anderen DateTime-Objekt ODER einer Gleitkommazahl wie der, die vom Python time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die kleiner oder gleich dem angegebenen DateTime-Objekt bzw. der time-Modul-Zeit ist. Wurde überarbeitet, um exaktere Ergebnisse durch den Vergleich von integer-Millisekunden zu gewährleisten.

Berechtigung:
Steht immer zur Verfügung.

Mon()

Kompatibilität: bitte beachten Sie aMonth

Berechtigung:
Steht immer zur Verfügung.

parts()

Gibt einen Tupel zurück, der das Kalenderjahr, Monat, Tag, Stunde, Minute, Sekunde und Zeitzone des Objektes enthält.

Berechtigung:
Steht immer zur Verfügung.

isCurrentYear()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die in das aktuelle Jahr fällt, wie immer im Kontext der Zeitzonen-Darstellung des Objektes.

Berechtigung:
Steht immer zur Verfügung.

PreciseAMPM()

Gibt den Zeit-String für das Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

AMPMMinutes()

Gibt den Zeit-String für ein Objekt zurück, ohne die Sekunden anzuzeigen.

Berechtigung:
Steht immer zur Verfügung.

equalTo(t)

Vergleicht dieses DateTime-Objekt mit einem anderen DateTime-Objekt ODER einer Gleitkommazahl wie der, die vom Python time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die gleich dem angegebenen DateTime-Objekt oder der time-Modul-Zeit ist. Wurde überarbeitet um exaktere Ergebnisse durch den Vergleich von Millisekunden zu gewähren.

Berechtigung:
Steht immer zur Verfügung.

pDay()

Gibt den abekürzten (mit Punkt) Namen des Tages der Woche zurück.

Berechtigung:
Steht immer zur Verfügung.

notEqualTo(t)

Vergleicht dieses DateTime-Objekt mit einem anderen DateTime-Objekt ODER einer Gleitkommazahl wie der, die vom Python time-Modul zurückgegeben wird. Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die NICHT gleich dem angegebenen

DateTime-Objekt oder der time-Modul-Zeit ist. Wurde überarbeitet um exaktere Ergebnisse durch den Vergleich von Millisekunden zu gewähren.

Berechtigung:
Steht immer zur Verfügung.

h_24()

Gibt die 24-Stunden-Darstellung der Stunde zurück.

Berechtigung:
Steht immer zur Verfügung.

pCommonZ()

Gibt eine String-Darstellung der Werte des Objektes im Format: Mar. 1, 1997 1:45 pm US/Eastern zurück.

Berechtigung:
Steht immer zur Verfügung.

isCurrentMonth()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die in den aktuellen Monat fällt, wie immer im Kontext der Zeitzonen-Darstellung des Objekts.

Berechtigung:
Steht immer zur Verfügung.

DayOfWeek()

Kompatibilität: siehe Day

Berechtigung:
Steht immer zur Verfügung.

latestTime()

Gibt ein neues DateTime-Objekt zurück, das die spätestmögliche Zeit (in ganzen Sekunden) im Zeitzonenkontext des Objektes anzeigt, die noch in den Tag des Objektes fällt.

Berechtigung:
Steht immer zur Verfügung.

dow_1()

Gibt den Integerwert des Wochentages zurück, wobei Sonntag den Wert 1 hat.

Berechtigung:
Steht immer zur Verfügung.

timezone()

Gibt die Zeitzone zurück, in der das Objekt dargestellt wird.

Berechtigung:
Steht immer zur Verfügung.

year()

Gibt das Kalenderjahr des Objekts zurück.

Berechtigung:
Steht immer zur Verfügung.

PreciseTime()

Gibt den Zeit-String für das Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

ISO()

Gibt das Objekt im ISO-Standard-Format zurück.

Daten werden im Format: YYYY-MM-DD HH:MM:SS ausgegeben.

Berechtigung:
Steht immer zur Verfügung.

millis()

Gibt die Millisekunden seit der Epoche in GMT zurück.

Berechtigung:
Steht immer zur Verfügung.

second()

Gibt die Sekunde zurück.

Berechtigung:
Steht immer zur Verfügung.

month()

Gibt den Monat des Objekts als Integer zurück.

Berechtigung:
Steht immer zur Verfügung.

pMonth()

Gibt den abgekürzten (mit Punkt) Monatsnamen zurück.

Berechtigung:
Steht immer zur Verfügung.

aMonth()

Gibt den abgekürzten Monatsnamen zurück.

Berechtigung:
Steht immer zur Verfügung.

isCurrentMinute()

Gibt true zurück, wenn das Objekt ein Datum/eine Zeit darstellt, das/die in die aktuelle Minute f&aumllt, wie immer im Kontext der Zeitzonen-Darstellung des Objekts.

Berechtigung:
Steht immer zur Verfügung.

Day()

Gibt den vollen Namen des Wochentages zurück.

Berechtigung:
Steht immer zur Verfügung.

aDay()

Gibt den abgekürzten Namen des Wochentages zurück.

Berechtigung:
Steht immer zur Verfügung.

Modul **ExternalMethod**

Klasse **ExternalMethod**

Im Web aufrufbare Funktionen die externe Python-Funktionen enthalten.

Die Funktion wird in einer externen Datei definiert. Diese Datei wird wie ein Modul behandelt, ist jedoch keins. Sie wird nicht direkt importiert, sondern wird gelesen und ausgewertet. Die Datei muss sich im `Extensions`-Unterverzeichnis der Zope-Installation oder in einem `Extensions`-Unterverzeichnis eines Produktverzeichnisses befinden.

Durch die Art, wie ExternalMethods geladen werden, ist es *noch* nicht möglich, Python-Module zu importieren, die sich im `Extensions`-Verzeichnis befinden. Es ist möglich, Module zu importieren, die im `lib/python`-Verzeichnis der Zope-Installation gefunden werden, ebenso Packages, die sich im `lib/python`-Verzeichnis befinden.

manage_edit(title, module, function, REQUEST=None)

Verändern Sie die externe Methode.

Bitte beachten Sie auch die Beschreibung von manage_addExternalMethod, um noch mehr über die Argumente `module` und `function` zu erfahren.

Bitte beachten Sie, dass der Aufruf von `manage_edit` dazu führt, dass das Modul effektiv neu geladen wird. Dies ist nützlich, während Debugs durchgeführt werden, um die Effekte von Veränderung zu sehen, kann jedoch zu Funktions-Poblemen bei freigegebenen globalen Daten führen.

`__call__(*args, **kw)`

Ruft die externe Methode auf.

Eine externe Methode aufzurufen kommt in etwa dem Aufruf der originalen Funktion in Python gleich. Positionale und Schlüssel-Parameter können wie üblich übergeben werden. Bitte beachten Sie jedoch, dass, anders als im Fall einer normalen Python-Methode, das "self"-Argument explizit weitergegeben werden muss. Eine Ausnahme zu dieser Regel wird gemacht wenn:

- die gelieferte Nummer von Argumenten um eins kleiner ist als die benötigte Anzahl an Argumenten und
- der Name des ersten Argumtents der Funktion `self` ist.

In diesem Fall wird der URL-Ursprung des Objektes als erstes Argument übergeben.

ObjectManager Constructor

`manage_addExternalMethod(id, title, module, function)`

Eine externe Methode zu einem `ObjectManager` hinzufügen.

Zusätzlich zu den standardmässigen Argumenten zur Objekt-Erzeugung, nämlich `id` und title, sind die folgenden Argumente definiert:

function
: Der Name der Python-Funktion. Dies kann eine gewöhnliche Python-Funktion oder eine gebundene Methode sein.

module
: Der Name der Datei, die die Funktionsdefinition enthält.

Das Modul befindet sich normalerweise im `Extensions`-Verzeichnis, es kann jedoch die Vorsilbe `product.` haben, die anzeigt, dass es in einem Produktverzeichnis zu finden sein sollte.

Beispiel: ist das Modul `ACMEWidgets.foo`, wird ein Versuch gestartet, die Datei `lib/python/Products/ACMEWidgets/Extensions/foo.py` zu benutzen. Schlägt dies fehl, wird die Datei `Extensions/ACMEWidgets.foo.py` benutzt.

Modul `File`

Klasse `File(ObjectManagerItem, PropertyManager)`

Ein File ist ein Zope-Objekt, dass Dateiinhalt enthält. Ein File-Objekt kann benutzt werden, um Dateiinformationen mit Zope rauf- bzw. herunterzuladen.

Ein File-Objekt in Zope zu benutzen ist leicht. Die gebrächliste Verwendung ist die Darstellung der Inhalte einer Datei in einer Webseite. Dies wird einfach dadurch erreicht, dass das Objekt von DTML referenziert wird:

```
<dtml-var standard_html_header>
  <dtml-var FileObject>
<dtml-var standard_html_footer>
```

Ein komplexeres Beispiel ist die Darstellung der Datei zum Download durch den Benutzer. Das nächste Beispiel zeigt einen Link zu jedem File-Objekt in einem Ordner zum Download durch den Benutzer:

```
<dtml-var standard_html_header>
<ul>
  <dtml-in "ObjectValues('File')">
    <li><a href="<dtml-var absolute_url>"><dtml-var
    id></a></li>
  </dtml-in>
</ul>
<dtml-var standard_html_footer>
```

In diesem Beispiel werden die `absolute_url`- und `id`-Methode benutzt, um eine Liste von HTML-Hyperlinks zu allen File-Objekten im aktuellen Objekt-Manager zu erstellen.

Bitte beachten sie auch ObjectManager für Details zur `objectValues`-Methode.

`getContentType()`

Gibt den Inhaltstyp der Datei zurück.

Berechtigung:
`View`

`update_data(data, content_type=None, size=None)`

Aktualisiert den Inhalt der Datei mit `data`.

Das `data`-Argument muss ein String sein. Wird `content_type` nicht angegeben, wird kein Content-Type definiert. Ist size nicht angegeben, wird die Grösse der Datei aus `data` errechnet.

Berechtigung:
Steht nur in Python zur Verfügung.

`getSize()`

Gibt die Dateigrösse in Bytes zurück.

Berechtigung:
`View`

ObjectManager Constructor

`manage_addFile(id, file="", title="", precondition="", content_type="")`

Fügt ein neues File-Objekt hinzu.

Erstellt ein neues File-Objekt `id` mit dem Inhalt von `file`.

Modul `Folder`

Klasse `Folder(ObjectManagerItem, ObjectManager, PropertyManager)`

Ein Folder ist ein generisches Container-Objekt in Zope.

Folders sind die gebrächlichsten Unterklassen des Objekt-Managers in Zope.

ObjectManager Constructor

`manage_addFolder(id, title)`

Fügt einen Folder zum aktuellen Objekt-Manager hinzu.

Berechtigung:
`Add Folders`

Modul `Image`

Klasse `Image(File)`

Ein Image ist ein Zope-Objekt, das Bildinhalte enthält. Ein Image-Objekt kann benutzt werden, um Bildinformationen mit Zope rauf- bzw. herunterzuladen.

Image-Objekte haben zwei Eigenschaften die ihre Dimesion definieren, nämlich `height` und `width`. Sie werden berechnet wenn das Bild hochgeladen wird. Bei Image-Typen, die Zope nicht versteht, könnten diese Eigenschaften undefiniert bleiben.

Ein Image-Objekt in Zope zu benutzen ist leicht. Die gebräuchlichste Anwendung ist die Darstellung die Inhalte eines Image-Objektes in einer Webseite. Dies wird durch die einfache Referenzierung des Objektes von DTML erreicht:

```
<dtml-var standard_html_header>
  <dtml-var ImageObject>
<dtml-var standard_html_footer>
```

Die generiert ein HTML IMG-Tag, das die URL zum Bild referenziert. Dies ist das gleiche wie:

```
<dtml-var standard_html_header>
  <dtml-with ImageObject>
    <img src="<dtml-var absolute_url>">
  </dtml-with>
<dtml-var standard_html_footer>
```

Sie können die Bilddarstellung noch genauer kontrollieren, indem sie die `tag`-Methode benutzen. Beispielsweise:

```
<dtml-var "ImageObject.tag(border='5', align='left')">
```

`tag(height=None, width=None, alt=None, scale=0, xscale=0, yscale=0, **args)`

Diese Methode gibt einen String zurück, der ein HTML IMG-Tag-Referenz zum Bild enthält.

Optional können die `height`-, `width`-, `alt`-, `scale`-, `xscale`- und `yscale`-Argumente angegeben werden, die in Attribute des HTML IMG-Tags umgewandelt werden. Bitte beachten Sie, dass `height` und `width` defaultmässig angegeben werden und `alt` aus der `title_or_id`-Methode stammt.

Schlüsselwort-Argumente können angegeben werden, um andere oder zukünftige Attribute des IMG-Tags zu unterstützen. Die einzige Ausnahme hierbei ist das "HTML Cascading Style Sheet"-Tag `class`. Da das Wort `class` in Python ein reserviertes Wort ist, muss es durch das Schlüsselwort-Argument `css_class` ersetzt werden. Dies wird im gerenderten `img`-Tag wieder in das HTML IMG-Tag Attribut `class` umgewandelt werden.

Berechtigung:
`View`

ObjectManager Constructor

`manage_addImage(id, file, title="", precondition="", content_type="")`

Fügt ein neues Image-Objekt hinzu.

Erstellt ein neues Image-Objekt `id` mit den Inhalten von `file`.

Modul `MailHost`

Klasse `MailHost`

MailHost-Objekte arbeiten als Adapter zu Simple Mail Transfer Protcol (SMTP)-Servern. MailHosts werden von DTML `sendmail`-Tags verwendet, um den richtigen Host, an den die Mail geschickt werden soll, zu finden.

send(messageText, mto=None, mfrom=None, subject=None, encode=None)

Sendet eine e-mail.
Die Argumente sind:

messageText
: Der Körper der e-mail.

mto
: Ein String bzw. eine Liste von Empfänger(n) der e-mail.

mfrom
: Die Absenderadresse.

subject
: Das Thema der mail.

encode
: Die durch rfc822 definierte Verschlüsselung der Nachricht. Der Defaultwert `None` bedeutet, dass keine Verschlüsselung benutzt wird. Gültige Werte umfassen `base64`, `quoted-printable` und `uuencode`.

ObjectManager Constructor

manage_addMailHost(id, title="", smtp_host=None, localhost=localhost, smtp_port=25, timeout=1.0)

Fügt ein mailhost-Objekt zum Objekt-Manager hinzu.

Modul `ObjectManager`

Klasse `ObjectManager`

Ein ObjectManager beinhaltet andere Zope-Objekte. Die beinhalteten Objekte heissen Object Manager Items.

Um ein Objekt innerhalb eines ObjectManagers zu erstellen benutzen Sie `manage_addProduct`:

```
self.manage_addProduct['OFSP'].manage_addFolder(id, title)
```

In DTML wäre dies:

```
<dtml-call "manage_addProduct['OFSP'].manage_addFolder(id,
title)">
```

Diese Beispiele erstellen einen neuen Folder innerhalb des aktuellen ObjectManagers.

`manage_addProduct` ist eine Zuordnung die Zugriff auf Methoden zur Produkt-Konstruktion bereitstellt. Sie hat die Produkt-ID als Index.

Konstruktor-Methoden werden während der Produktinitialisierung registriert und sollten in den API-Dokumentationen für jedes Produkt, das hinzugefügt werden kann, dokumentiert sein.

objectItems(type=None)

Diese Methode gibt eine Sequenz von (ID, Objekt)-Tupeln zurück.

Wie objectValues und objectIds, so wird auch hier ein Argument akzeptiert, entweder ein String oder eine Liste, um die Resultate auf Objekte eines gegebenen Meta-Typs oder eines Sets von Meta-Typen zu beschr&aum;nken.

Jedes erste Element eines Tupels ist die ID eines Objektes, das im Objekt-Manager beinhaltet ist, jedes zweite ist das Objekt selber.

Beispiel:

```
<dtml-in objectItems>
 id: <dtml-var sequence-key>,
 type: <dtml-var meta_type>
<dtml-else>
  Es gibt keine Unterobjekte.          </dtml-in>
```

Berechtigung:
Access contents information

superValues(type)

Diese Methode gibt eine Liste von Objekten des gegebenen Meta-Typs (der gegebenen Meta-Typen), die im Objekt-Manager und allen darüberliegenden enthalten sind, zurück.

Das type-Argument spezifiziert den Meta-Typ (die Meta-Typen). Es kann ein String sein, der nur einen Meta-Typ beschreibt oder aber eine Liste von String um mehrere zu beschreiben.

Berechtigung:
Nur in Python verfügbar.

objectValues(type=None)

Diese Methode gibt eine Sequenz von beinhalteten Objekten zurück.

Wie objectItems und objectIds, akzeptiert sie ein Agrument, entweder einen String oder eine Liste um die Ergebnisse auf Objekte eines gegebenen Meta-Typs bzw. eines Sets von Meta-Typen zu beschränken.

Beispiel:

```
<dtml-in expr="objectValues('Folder')">
  <dtml-var icon>
  Dies ist das Icon für den: <dtml-var id> Folder<br>.
<dtml-else>
```

```
  Es gibt keine Folders.
</dtml-in>
```

Die Ergebnisse wurden auf Folders beschränkt da ein Meta-Typ an die `objectValues`-Methode übergeben wurde.

Berechtigung:
`Access contents information`

`objectIds(type=None)`

Diese Methode gibt eine Liste der IDs der beinhalteten Objekte zurück.

Optional kann ein Argument übergeben werden, das definiert, auf welche(n) Meta-Typ(en) die Ergebnisse beschränkt werden sollen. Dieses Argument kann ein String sein, der einen Meta-Typ beschreibt oder aber eine Liste von Strings um mehrere zu beschreiben.

Beispiel:

```
<dtml-in objectIds>
  <dtml-var sequence-item>
<dtml-else>
  Es gibt keine Unterobjekte.
</dtml-in>
```

Dieser DTML-Code wird alle IDs der Objekte anzeigen, die im aktuellen Objekt-Manager enthalten sind.

Berechtigung:
`Access contents information`

Modul **`ObjectManagerItem`**

Klasse **`ObjectManagerItem`**

Ein Zope-Objekt kann in einem Objekt-Manager enthalten sein. Fast alle Zope-Objekte, die über das Web verwaltet werden können sind Object Manager Items.

ObjectMangerItems haben diese Instanz-Attribute:

`title`
: Der Titel des Objekts.

  Dies ist eine optionale einzeilige String-Beschreibung des Objekts.

`meta_type`
: Ein Kurzname für den Typ des Objekts.

  Dies ist der Name der in den Listen für hinzuzufügende Produkte für das Objekt angezeigt wird und gebraucht wird, wenn Objekte nach dem Typ gefiltert werden

Dieses Attribut wird von der Objekt-Klasse übermittelt und sollte nicht eigenhändig verändert werden.

`REQUEST`
Die aktuelle Web-Anfrage.

Dieses Objekt wird benötigt und sollte nicht verändert werden.

`title_or_id()`

Wird der Titel nicht leergelassen, wird er zurückgegeben, andernfalls jedoch die ID.

Berechtigung:
Steht immer zur Verfügung.

`getPhysicalRoot()`

Gibt das Top-Level Zope-Anwendungs-Objekt zurück.

Berechtigung:
Nur in Python verfügbar.

`manage_workspace()`

Dies ist die Web-Methode, die aufgerufen wird, wenn ein Benutzter ein Element in einer Ansicht des Objekt-Managers auswählt oder in der Zope Management-Navigations-Ansicht.

Berechtigung:
`View management screens`

`getPhysicalPath()`

Gibt den Pfad eines Objekts von der Wurzel an zurück, wobei virtuelle Hosts ignoriert werden.

Berechtigung:
Steht immer zur Verfügung.

`unrestrictedTraverse(path, default=None)`

Gibt das Objekt zurück, indem der gegebene Pfad von dem Objekt aus durchquert wird, von dem die Methode aufgerufen wurde. Diese Methode beginnt mit "unrestricted", da (fast) keine Sicherheits-Überprüfungen durchgeführt werden.

Wird ein Objekt nicht gefunden, wird das `default`-Argument zurückgegeben.

Berechtigung:
Steht nur in Python zur Verfügung.

`getId()`

Gibt die ID des Objektes zurück.

Die `ID` ist der eindeutige Name eines Objektes innerhalb seines übergeordneten Objekt-Managers. Dies sollte ein String sein, der Buchstaben, Zahlen, Unterstriche, Bindestriche, Kommata und Leerzeichen enthalten kann.

Diese Methode ersetzt den direkten Zugriff auf das `id`-Attribut.

Berechtigung:
Steht immer zur Verfügung.

`absolute_url(relative=None)`

Gibt die absolute URL zum Objekt zurück.

Wird das relativ-Argument mit true angegeben, ist die zurückgegebene URL relativ zum Objekt. Bitte beachten Sie: wenn virtuelle Hosts verwendet werden, ist der zurückgegeben Pfad ein logischer, statt eines physischen Pfades.

Berechtigung:
Steht immer zur Verfügung.

`this()`

Gibt das Objekt zurückl

Dies stellt sich in zwei Situationen als sehr nützlich heraus. Zunächst stellt dies einen Weg bereit, um Objekte in DTML zu referenzieren.

Die zweite Verwendung ist komplexer. Es wird ein Weg bereitgestellt, ein Objekt zu erhalten, ohne den vollen Kontext in dem es erhalten wurde, zu bekommen. Dies ist beispielsweise in Fällen nützlich, in denen Sie in einer Methode eines Unterobjektes (das kein Element ist) eines Elementes sind und in denen sie das Element aus dem Kontext des Unterobjektes herausbekommen müssen.

Berechtigung:
Steht immer zur Verfügung.

`restrictedTraverse(path, default=None)`

Gibt das erhaltene Objekt zurück, indem der gegeben Pfad umgedreht wird, und zwar vom Objekt aus, von dem aus die Methode aufgerufen wurde, wobei Sicherheitsüberprüfungen durchgeführt werden.

Wird ein Objekt nicht gefunden, wird das `default`-Argument zurückgegeben.

Berechtigung:
Steht immer zur Verfügung.

`title_and_id()`

Ist der Titel nicht leer, ist das Resultat der Titel, gefolgt von der ID in runden Klammern. Andernfalls wird die ID zurückgegeben.

Berechtigung:
Steht immer zur Verfügung.

Modul `PropertyManager`

Klasse `PropertyManager`

Ein PropertyManager-Objekt enthält eine Sammlung von geschriebenen Attributen, die Properties (Eigenschaften) genannt werden. Properties können über das Web oder via DTML verwaltet werden.

Zusätzlich dazu, dass sie einen Typ haben, können Properties beschreibbar oder nur lesbar sein und Default-Werte haben.

`propertyItems()`

Gibt eine Liste von (ID, Property)-Tupeln zurück.

Berechtigung:
`Access contents information`

`propertyValues()`

Gibt eine Liste von Property-Werten zurück.

Berechtigung:
`Access contents information`

`propertyMap()`

Gibt ein Tupel von Zuordnungen zurück, die Meta-Daten der Properties beschreiben. Die Meta-Daten umfassen `id`, `type`, und `mode`.

Berechtigung:
`Access contents information`

`propertyIds()`

Gibt eine Liste von Property-IDs zurück.

Berechtigung:
`Access contents information`

`getPropertyType(id)`

Sucht den Typ der Property-`ID`. Gibt none zurück, wenn die Property nicht existiert.

Berechtigung:
`Access contents information`

getProperty(id, d=None)

Gibt den Wert der Property-ID zurück. Wird die Property nicht gefunden, wird das optionale zweite Argument oder None zurückgegeben.

Berechtigung:
Access contents information

hasProperty(id)

Gibt true zurück, wenn der PropertyManager die Property-ID enthält. Andernfalls wird false zurückgegeben.

Berechtigung:
Access contents information

Modul PropertySheet

Klasse PropertySheet

Ein PropertySheet ist eine Abstraktion, um mit einem Set von ähnlichen Properties zu arbeiten und Properties zu verwalten. Grundsätzlich verhält sich das PropertySheet wie ein Container für ein Set von verwandten Properties und Meta-Daten, die diese Properties beschreiben. Ein PropertySheet kann eine Web-Schnittstelle bereitstellen, um die Properties zu verwalten, dies ist jedoch nicht zwingend notwendig.

xml_namespace()

Gibt einen Namensraum-String zurück, der als XML-Namensraum für dieses Property-Set verwendet werden kann. Dies kann ein leerer String sein, wenn es keinen Default-Namensraum f&uum;r ein gegebenes PropertySheet gibt (besonders dann, wenn es sich um ein PropertySheet handelt, dass in ZClass-Definitionen hinzugefügt wurde).

Berechtigung:
Steht nur in Python zur Verfügung.

propertyItems()

Gibt eine Liste von (ID, Property)-Tupeln zurück.

Berechtigung:
Access contents information

propertyValues()

Gibt eine Liste der aktuellen Property-Werte zurück.

Berechtigung:
Access contents information

getPropertyType(id)

Gibt den Typ der Property-ID zurück. Gibt None zurück, wenn die Property nicht existiert.

Berechtigung:
Steht nur in Python zur Verfügung.

propertyInfo()

Gibt eine Zuordnung zurück, die Meta-Daten von Properties enthält.

Berechtigung:
Steht nur in Python zur Verfügung.

getProperty(id, d=None)

Sucht die Property-ID, wobei das optionale zweite Argument zurückgegeben wird oder aber None, wenn keine Property gefunden wurde.

Berechtigung:
Steht nur in Python zur Verfügung.

manage_delProperties(ids=None, REQUEST=None)

Löscht eine oder mehrere Properties mit den gegebenen IDs. Das IDs-Argument sollte eine Sequenz (Tupel oder Liste) sein, die die IDs der zu löschenden Properties enthält. Ist das IDs-Argument leer wird keine Aktion ausgeführt. Existiert eine der Properties, die in IDs genannt werden nicht, wird ein Fehler ausgegeben.

Einige Objekte haben "besondere" Properties, die von Produktautoren definiert werden und nicht entfernt werden können. Wird eine dieser Properties in IDs genannt, wird eine HTML-Fehlermeldung zurückgegeben.

Wird an das REQUEST-Argument kein Wert weitergegeben, gibt die Methode None zurück. Wird ein Wert für REQUEST angegeben (wie es der Fall ist, wenn über das Web gearbeitet wird), wird das Management-Formular für das Objekt gerendert und zurückgegeben.

Diese Methode kann über das Web, per DTML oder aus Python-Code aufgerufen werden.

Berechtigung:
Manage Properties

manage_changeProperties(REQUEST=None, **kw)

Verändert existierende Objekt-Properties, indem entweder ein Zuordnungs-Objekt als REQUEST übergeben wird, das Name:Wert-Paare enthält oder indem Name=Schlüsselwort-Argumente übergeben werden.

Einige Objekte haben "besondere" Properties, die von Produktautoren definiert werden und nicht verändert werden können. Wenn Sie versuchen, eine dieser Properties mit dieser Methode zu verändern, wird ein Fehler ausgegeben.

Bitte beachten Sie, dass keine Rechtschreibprüfung oder Konvertierung geschieht, wenn diese Methode aufgerufen wird, also liegt es in der Verantwortlichkeit des Benutzers,

sicherzustellen, dass die aktualisierten Werte vom richtigen Typ sind. *Dies wird sich vielleicht demnächst ändern*.

Wird für `REQUEST` ein Wert angegeben (wie es der Fall ist wenn über das Web gearbeitet wird), gibt die Methode einen HTML-Dialog zurück. Wird kein REQUEST weitergegeben, gibt die Methode nach erfolgreicher Ausführung `None` zurück.

Diese Methode kann über das Web, aus DTML oder Python-Code aufgerufen werden.

Berechtigung:
`Manage Properties`

`manage_addProperty(id, value, type, REQUEST=None)`

Fügt eine neue Property mit der gegebenen `id`, mit `value` und `type` hinzü.

Dies sind Property-Types:

`boolean`
: 1 oder 0.

`date`
: Ein `DateTime`-Wert, beispielsweise `12/31/1999 15:42:52 PST`.

`float`
: Eine Dezimalzahl, beispielsweise `12.4`.

`int`
: Ein integer, beispielsweise `12`.

`lines`
: Eine Liste von Strings, ein String pro Zeile.

`long`
: Ein long, beispielsweise `12232322322323232323423`.

`string`
: Ein String aus Charactern, beispielsweise `This is a string`.

`text`
: Ein mehrzeiliger String, beispielsweise ein Absatz.

`tokens`
: Eine Liste von Strings, die durch Leerzeichen getrennt werden, beispielsweise `one two three`.

`selection`
: Ein String, der mit einem Pop-Up-Menü ausgewählt wird.

`multiple selection`
: Eine Liste von Strings, die aus einer Selektions-Liste ausgewählt werden.

Diese Methode benutzt den eingegebenen `type` um den Versuch einer Konvertierung der `value`-Argument zu den genannten Typen zu starten. Kann der angegebene `value`-Wert nicht konvertiert werden, wird ein ValueError ausgegeben.

Die für `selection` und `multiple selection` angegebenen Werte können Attribute oder Methodennamen sein. Die Attribute oder Methoden müssen Sequenz-Werte zurückgeben.

Wird der gegebene `type` nicht erkannt, werden die Werte für `value` und `type` einfach blind in das Objekt gepackt.

Wird kein Wert für `REQUEST` weitergegeben, gibt die Methode `None` zurück. Wird ein Wert für `REQUEST` weitergegeben (wie es der Fall ist wenn über das Web aufgerufen wird), wird das PropertyManagement-Formular f&uum;r das Objekt gerendert und ausgegeben.

Diese Methode kann über das Web, aus DTML oder Python-Code heraus aufgerufen werden.

Berechtigung:
`Manage Properties`

`propertyMap()`

Gibt einen Tupel von Zuordnungen zurück, der Meta-Daten für Properties enthält.

Berechtigung:
Steht nur in Python zur Verfügung.

`propertyIds()`

Gibt eine Liste von Property-IDs zurück.

Berechtigung:
`Access contents information`

`hasProperty(id)`

Gibt true zurück, wenn `self` eine Property mit der gegebenen `id` hat, andernfalls false.

Berechtigung:
`Access contents information`

Modul **`PropertySheets`**

Klasse **`PropertySheets`**

Ein PropertyShee ist eine Abstraktion um mit einem Set von verwandten Properties zu arbeiten und Properties zu verwalten. Grundsätzlich verhält es sich wie ein Container für ein Set von verwandten Properties und Meta-Daten, die diese Properties beschreiben. Auf PropertySheet-Objekte wird über ein PropertySheet-Objekt zugegriffen, das als Sammlung von PropertySheet-Instanzen fungiert.

Objekte, die PropertySheets unterstützen (Objekte, die die PropertyManager-Schnittstelle oder ZClass-Objekte unterst&uumltzen) haben ein `propertysheets`-Attribut (eine PropertySheets-Instanz), welche die Sammlung der PropertySheet-Objekte ist. Das PropertySheet-Objekt enthüllt eine Schnittstelle, die der Python-Zuordnung stark ähnelt, so dass auf individuelle PropertySheet-Objekte über Dicitionary-ähnliche Schlüsselsuche zugegriffen werden kann.

`get(name, default=None)`

Gibt das PropertySheet zurück, welches durch `name` oder den Wert identifiziert wird, der in `default` angegeben ist, falls das genannte PropertySheet nicht gefunden wird.

Berechtigung:
Steht nur in Python zur Verfügung.

`values()`

Gibt eine Sequenz aller PropertySheet-Objekte in der Sammlung zurück.

Berechtigung:
Steht nur in Python zur Verfügung.

`items()`

Gibt eine Sequenz zurück, die einen `(ID, Objekt)`-Tupel für jedes PropertySheet-Objekt in der Sammlung enthält.

Berechtigung:
Steht nur in Python zur Verfügung.

Modul `PythonScript`

Klasse `PythonScript(Script)`

Python-Skripte enthalten Python-Code der ausgeführt wird, wenn Sie das Skript mit den folgenden Möglichkeiten aufrufen:

- Das Skript wird über das Web aufgerufen, indem man einem Web-Browser auf seine Adresse zugreift.
- Das Skript wird aus einem anderen Skript-Objekt heraus aufgerufen.
- Das Skript wird aus einem Methoden-Objekt aufgerufen, wie z.B. einer DTML-Methode.

Python-Skripts können ein "sicheres" Subset der Python-Sprache enthalten. Python-Skripts müssen sicher sein, dass sie unter Umständen von vielen verschiedenen Benutzer durch ein unsicheres Medium wie das Web bearbeitet werden können. Die folgenden Sicherheitsaspekte erfüllen die Anforderungen für sichere Python-Skripte:

- Da viele Benutzer Zope verwenden können, muss ein Python-Skript sicherstellen, dass es einem Benutzer nicht erlaubt, etwas zu tun, das er nicht tun darf, wie z.B. Objekte löschen, die der Benutzer mit seinen Rechten nicht löschen dürfte. Aufgrund dieser Anforderung führen Python-Skripte viele Sicherheitsüberprüfungen aus, während sie ausgeführt werden.
- Da Python-Skripte über das unsichere Medium "Web" bearbeitet werden können, ist es ihnen nicht gestattet, auf das Dateisystem des Zope-Servers zuzugreifen. Normale Python-Komponenten wie `open` sind darum nicht erlaubt.
- Da viele standardmässige Python-Module die oberen zwei Sicherheitsbestimmungen verletzen, darf nur ein kleiner Teil der Python-Module mit dem "import"-Statement in Python-Skripte eingefügt werden, es sei denn, sie wurden von Zopes Sicherheitsmethode validiert. Gegenwärtig sind die folgenden standardmässigen Python-Module validiert worden:

    - string
    - math
    - whrandom und random
    - Products.PythonScripts.standard
- Da es ihnen erlaubt, beliebigen Python-Code auszuführen, ist das "exec"-Statement von Python in Python-Methoden nicht erlaubt.
- Da sie Sicherheitsverletzungen darstellen oder verursachen können, sind einige standardmässige Python-Funktionen nicht erlaubt. Die folgenden standardmässigen Python-Funktionen sind nicht erlaubt:
    - open
    - input
    - raw_input
    - eval
    - execfile
    - compile
    - type
    - coerce
    - intern
    - dir
    - globals
    - locals
    - vars
    - buffer
    - reduce
- Andere standardmässige Funktionen sind von Natur aus eingeschränkt. Es handelt sich hierbei um die folgenden:

  range
  Aufgrund möglicher "Memory Denial of Service"-Attacken ist range standardmässig so beschränkt, dass nur ranges erstellt werden können, die kürzer als 10.000 Elemente sind.
  filter, map, tuple, list
  Aus oben genannten Gründen verarbeiten standardmässige Funktionen, die aus Sequenzen Listen erstellen, keine Strings.
  getattr, setattr, delattr
  Da diese Funktionen Python-Code dazu befähigen könnten, das Sicherheitssystem von Zope zu umgehen, werden sie mit allgemeinen, durch das Sicherheitssystem eingeschränkte Versionen ersetzt.

- Um mit den Python-Ausdrücken, die in der DTML benutzt werden, konsistent zu sein, werden die standardmässigen Funktionen mit einer kleinen Zahl an Funktionen und Klassen erweitert:
    - test
    - namespace
    - render
    - same_type
    - DateTime
- Da die "print"-Statements nich normal mit Zope operieren können, wurde ihr Effekt geändert. Statt Text an stdout zu senden, hängt sich "print" an eine interne Variable. Der spezielle standardmässige Name "printed" berechnet die Verkettung des gesamten bisherigen Textes während der Ausführung des Skripts.

document_src(REQUEST=None, RESPONSE=None)

Gibt den Text der `read`-Methode, zurück, wobei der Inhaltstyp `text/plain` auf RESPONSE gesetzt wird.

ZPythonScript_edit(params, body)

Verändert die Parameter und den Inhalt des Skripts. Diese Methode akzeptiert zwei Argumente:

params
: Der neue Wert des Python Skript-Parameters. Muss eine durch Kommata getrennte Liste von Werten in gültiger Signatur-Syntax von Python sein. Enthält dieser Parameter keinen gültigen Signatur-String, wird ein SyntaxError ausgegeben.

body
: Der neue Wert des Inhaltes des Python-Skripts. Muss gültige Python-Syntax enthalten. Ist dies nicht der Fall, wird ein SyntaxError ausgegeben.

ZPythonScript_setTitle(title)

Ändert den Titel des Skripts. Diese Methode akzeptiert ein Argument, nämlich `title`, was der Wert für den neuen Titel des Skripts ist und ein String sein muss.

ZPythonScriptHTML_upload(REQUEST, file="")

Gibt einen Text mit der `write`-Methode an eine Datei weiter.

write(text)

Verändert das Skript, indem das Text-Argument in Teile geparst wird. Führende Zeilen, die mit `##` beginnen werden ausgeschnitten und wenn Sie von der Form `##name=value` sind, werden sie verwendet, um Meta-Daten wie etwa den Titel und Parameter zu setzen. Der Rest des Textes wird als neuer Inhalt des Python-Skripts gesetzt.

ZScriptHTML_tryParams()

Gibt eine Liste der benötigten Parameter zurück, mit denen das Skript getestet wird.

read()

Gibt den Inhalt des Python-Skripts zurück, inklusive eines besonderen vorangestellten Kommentarblocks. Dieser Block enthält Meta-Daten in der Form von Kommentarzeilen, wie sie von der `write`-Methode erwartet werden..

ZPythonScriptHTML_editAction(REQUEST, title, params, body)

Verändert die Hauptparameter des Skripts. Diese Methode akzeptiert die folgenden Argumente:

REQUEST
: Der aktuelle REQUEST.

title
: Der neue Wert des Titels des Python-Skripts. Dies muss ein String sein.

params
: Der neue Wert der Parameter des Python-Skripts. Muss eine durch Kommata getrennte Liste von Werten in gültiger Signatur-Syntax von Python sein. Ist kein gültiger Signatur-String enthalten, wird ein SyntaxError ausgegeben.

body
: Der neue Wert des Inhalts des Python-Skripts. Muss gültige Python-Syntax enthalten. Ist dies nicht der Fall, wird ein SyntaxError ausgegeben.

ObjectManager Constructor

`manage_addPythonScript(id, REQUEST=None)`

Fügt ein Python-Skript zu einem Ordner hinzu.

Modul `Request`

Klasse `Request`

Das Request-Objekt kapselt alle Informationen bezüglich der aktuellen Anfrage in Zope ein. Dies umfasst die Input-Header, die Formulardaten, die Serverdaten und Cookies.

Das Request-Objekt ist ein Zuordnungs-Objekt das eine Sammlung von Variable-Wert-Zuordnungen enthält. Zusätzlich werden Variablen in fünf Kategorien unterteilt:

- Umgebungsvariablen

  Diese Variablen umfassen Input-Header, Serverdaten und andere anfrage-spezifische Daten. Die Variablennamen sind spezifiziert und zwar inCGI-Spezifikation

- Formulardaten

  Diese Daten werden entweder von einem URL-verschlüsselten Abfrage-String oder -Inhalt extrahiert, sofern vorhanden.

- Cookies

  Es handelt sich hier um die Cookie-Daten, sofern vorhanden.

- Lazy Data ("Faule Daten")

  Dies sind Aufrufe, die aufgeschoben werden, bis ausdrücklich referenziert wird, wann sie aufgelöst (aufgerufen) und ihre Ergebnisse als "andere" Daten, v.a. normale Abfrage-Daten gespeichert werden.

  Hierdurch sind sie "faule" Datenelemente. Ein Beispiel sind die SESSION-Objekte.

"Faule" Daten in einem Aufruf können nur von der Python-Methode method set_lazy(name,callable) im REQUEST-Objekt aufgerufen werden. Diese Methode kann nicht aus DTML oder über das Web aufgerufen werden.

- Andere

  Daten die von einem Anwendungsobjekt gesetzt werden können.

Das Anfrage-Objekt kann als Zuordnungsobjekt verwendet werden, in diesem Fall werden Werte in folgender Reihenfolge gesucht: Umgebungsvariablen, andere Variablen, Formulardaten und schliesslich Cookies.

Diese besonderen Variablen werden im REQUEST gesetzt:

`PARENTS`
: Eine Liste von Objekten, die umgedreht wurde um die veröffentlichten Objekte zu erhalten. So wäre `PARENTS[0]` der Vorfahre des veröffentlichten Objekts.

`REQUEST`
: Das Request-Objekt.

`RESPONSE`
: Das Response-Objekt.

`PUBLISHED`
: Das endgültige Objekt, welches durch die Umkehrung der URLs veröffentlicht wurde.

`URL`
: Die URL eines Requests ohne den Abfrage-String.

URLn
: `URL0` ist das Selbe wie `URL`. `URL1` ist das Selbe wie `URL0`, wenn der letzte Pfad entfernt wurde. `URL2` ist das Selbe wie `URL1` wenn das letzte Element entfernt wurde, usw...

  Beispiel: ist URL='http://localhost/foo/bar', dann ist URL1='http://localhost/foo' und URL2='http://localhost'.

URLPATHn
: `URLPATH0` ist der Teil des Pfades von `URL`, `URLPATH1` ist der Teil des Pfades von `URL1`, und so weiter.

  Beispiel: Ist URL='http://localhost/foo/bar', dann ist URLPATH1='/foo' and URLPATH2='/'.

BASEn
: `BASE0` ist die URL bis zum aber ohne das Zope-Anwendungsobjekt. `BASE1` ist die URL des Zope-Anwendungsobjekts. `BASE2` ist die URL des Zope-Anwendungsobjekts mit einem zusätzlichen Pfad-Element, das in den Pfad des veröffentlichten Elements integriert wurde. Und so weiter.

  Beispiel: ist URL='http://localhost/Zope.cgi/foo/bar', dann ist BASE0='http://localhost', BASE1='http://localhost/Zope.cgi' und BASE2='http://localhost/Zope.cgi/foo'.

BASEPATHn

BASEPATH0 ist der Teil des Pfades von BASE0, BASEPATH1 ist der Teil des Pfades von BASE1 und so weiter. BASEPATH1 ist der nach aussen hin sichtbare Pfad zum Zope-Wurzelordner, was gleich ist mit SCRIPT_NAME in CGI, aber virtuelle Hosts erkennt.

Beispiel: ist URL='http://localhost/Zope.cgi/foo/bar', dann ist BASEPATH0='/', BASEPATH1='/Zope.cgi', und BASEPATH2='/Zope.cgi/foo'.

get_header(name, default=None)

Gibt den Namen des HTTP-Headers zurück, oder aber ein optionales Default-Argument oder None wenn der Header nicht gefunden wird. Bitte beachten Sie, dass sowohl die Namen von ursprünglichen als auch von CGI-Headern ohne HTTP_ am Anfang erkannt werden, beispielsweise sollten Content-Type, CONTENT_TYPE und HTTP_CONTENT_TYPE alle den Content-Type-Header zurückgeben, falls vorhanden.

Berechtigung:
Steht immer zur Verfügung.

items()

Gibt eine Sequenz von (Schlüssel, Wert)-Tupeln für alle Schlüssel im REQUEST-Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

keys()

Gibt eine sortierte Sequenz aller Schlüssel im REQUEST-Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

setVirtualRoot(path, hard=0)

Verändert URL, URLn, URLPATHn, BASEn, BASEPATHn und absolute_url(), so dass das aktuelle Objekt den Pfad path hat. Ist hard true, dann wird PARENTS geleert.

Stellt Unterstützung f&uum;r virtuelle Hosts zur Verfügung. Ist für den Aufruf von traversalen Veröffentlichungspunkten gedacht.

Berechtigung:
Steht immer zur Verfügung.

values()

Gibt eine Sequenz von Werten für alle Schlüssel im Request-Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

set(name, value)

Erstellt einen neuen Namen im Request-Objekt und weist ihm einen Wert zu. Dieser Name und der dazugehörige Wert werden in der `"Andere"`-Kategorie gespeichert.

Berechtigung:
Steht immer zur Verfügung.

has_key(key)

Gibt true zurück, wenn das Request-Objekt Schlüssel enthält, ansonsten false.

Berechtigung:
Steht immer zur Verfügung.

setServerURL(protocol=None, hostname=None, port=None)

Gibt die spezifizierten Elemente von `SERVER_URL` zurück, was auch Einfluss auf `URL`, `URLn`, `BASEn` und `absolute_url()` hat.

Stellt Unterstützung für virtuelle Hosts zur Verfügung.

Berechtigung:
Steht immer zur Verfügung.

Modul `Response`

Klasse `Response`

Das Response-Objekt stellt die Antwort auf eine Zope-Anfrage dar.

setHeader(name, value)

Setzt den Namen des HTTP-Return-Headers auf den Wert "value", wobei das vorige Werte-Set für den Header gelöscht wird, falls eines existiert. Ist das Literal-Flag true, wird der Fall des Header-Namens gesichert, andernfalls wird bei der Ausgabe eine Wort-Grossschreibung am Header-Namen ausgeführt.

Berechtigung:
Steht immer zur Verfügung.

setCookie(name, value, **kw)

Setzt ein HTTP-Cookie im Browser.

Die Antwort beinhaltet einen HTTP-Header, der auf cookie-verarbeitenden Browsern ein Cookie mit dem Schlüssel "name" und dem Wert "value" setzt. Hierdurch werden alle früher gesetzten Werte für das Cookie im Response-Objekt gelöscht.

Berechtigung:
Steht immer zur Verfügung.

addHeader(name, value)

Setzt einen neuen HTTP-Return-Header mit dem angegebenen Wert, während alle früher gesetzten Header mit dem selben Namen beibehalten werden.

Berechtigung:
Steht immer zur Verfügung.

appendHeader(name, value, delimiter=,)

Fügt einen Wert zu einem Cookie hinzu.

Setzt den Namen eines HTTP-Return-Headers auf den Wert "value", falls es schon einen früheren Wert gibt, wird der neue mit einem Komma angehängt.

Berechtigung:
Steht immer zur Verfügung.

write(data)

Gibt Daten als Stream zurück.

HTML-Daten können unter Verwendung einer Stream-orientierten Schnittstelle zurückgegeben werden. Dies erlaubt dem Browser, einseitige Ergebnisse während der Berechnung einer Antwort zum Fortfahren darzustellen.

Das veröffentlichte Objekt sollte zuerst jegliche Ausgabe-Header oder Cookies im Response-Objekt setzen.

Bitte beachten Sie, dass veröffentlichte Objekte nach dem Beginn der Stream-orientierten Ausgabe keine Fehler produzieren dürfen.

Berechtigung:
Steht immer zur Verfügung.

setStatus(status, reason=None)

Setzt den HTTP-Statuscode des Response-Objekts, das Argument kann entweder ein integer oder einer der folgenden Strings sein:

OK, Created, Accepted, NoContent, MovedPermanently, MovedTemporarily, NotModified, BadRequest, Unauthorized, Forbidden, NotFound, InternalError, NotImplemented, BadGateway, ServiceUnavailable

dies wird in den korrekten integer-Wert umgerechnet.

Berechtigung:
Steht immer zur Verfügung.

setBase(base)

Setzt die Basis-URL für das zurückgegebene Dokument.

Berechtigung:
    Steht immer zur Verfügung.

`expireCookie(name, **kw)`

Verursacht, dass ein HTTP-Cookie aus dem Browser entfernt wird.

Das Response-Objekt beinhaltet einen HTTP-Header, der das Cookie anhand seines Namens auf dem Client entfernt, falls es existiert. Dies wird durch den Versand eines neuen Cookies mit einem Verfallsdatum, das schon abgelaufen ist, vervollständigt. Bitte beachten Sie, dass einige Clients einen angegebenen Pfad benötigen - dieser Pfad muss exakt der Pfad des ursprünglichen Cookies sein. Der Pfad kann als Schlüsselwort-Argument angegeben werden.

Berechtigung:
    Steht immer zur Verfügung.

`appendCookie(name, value)`

Gibt einen HTTP-Header zurück, der ein Cookie auf Browsern, die mit Cookies arbeiten, setzt, wobei die Schlüssel "name" und "value" verwendet werden. Wurde einer der Werte für das Cookie kürzlich im Response-Objekt gesetzt, wird der neue Wert mit einem Komma an den alten angehängt.

Berechtigung:
    Steht immer zur Verfügung.

`redirect(location, lock=0)`

Verursacht eine Umleitung ohne die Ausgabe eines Fehlers. Wird das "lock"-Argument mit dem Wert true übergeben, wird der HTTP-Umleitungs-Antwort-Code nicht verändert, sogar, wenn in der weiteren Verarbeitung der Anfrage ein Fehler auftritt (nachdem redirect() aufgerufen wurde).

Berechtigung:
    Steht immer zur Verfügung.

Modul `Script`

Klasse `Script`

Eine Skript-basierte und über das Web aufrufbare Schnittstelle.

`ZScriptHTML_tryAction(REQUEST, argvars)`

Hängt die Test-Parameter, die aus dem Dicitonary `argvars` übergeben wurden an. Dies ruft das aktuelle Skript mit den gegebenen Argumenten auf und gibt das Ergebnis zurück.

Modul `SessionInterfaces`

Session API

Bitte beachten Sie auch:

- Transient Object API

Klasse `SessionDataManagerErr`

Ein fehler, der während einer Operation des Session-Data-Managers aufgerufen wird, wie in der API-Dokumentation des Session-Data-Managers noch ausführlicher erklärt wird.

Diese Exception kann in Python-Skripten abgefangen werden. Ein erfolgreicher Import der Exception zur Benutzung in einem Python-Skript würde hierbei so aussehen:

```
from Products.Sessions import SessionDataManagerErr
```

Klasse `BrowserIdManagerInterface`

Die Schnittstelle für den Manager der Zope Browser-IDs.

Eine Zope Browser-ID-Manager ist für die Zuordnung von IDs zu Besuchern verantwortlich und auch dafür, Anfragen von Session-Data-Managern in Beziehung zu Browser-IDs zu betreuen.

`getBrowserId(self, create=1)`

Hat create den Wert 0, so wird die aktuelle Browser-ID zurückgegeben oder aber None, falls es keinen Browser gibt, der mit der aktuellen Anfrage verknüpft ist. Hat create den Wert 1, wird die aktuelle Browser-ID zurückgegeben oder aber eine neu erstellte, falls es keinen Browser gibt, der mit der aktuellen Anfrage verknüpft ist. Diese Methode ist nützlich im Zusammenhang mit getBrowserIdName, wenn sie die Kombination Browser-ID-Name/Browser-ID als versteckten Wert in ein POST-basiertes Formular einfügen wollen. Die Browser-ID ist undurchlässig, hat keine Bedeutung für den Betrieb und Länge, Typ sowie Zusammenstellung können verändert werder.

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr falls eine ungültige Browser-ID im REQUEST gefunden wird.

`isBrowserIdFromCookie(self)`

Gibt true zurück, wenn die Browser-ID von einem Cookie stammt.

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, falls es keine aktuelle Browser-Id gibt.

`isBrowserIdNew(self)`

Gibt true zurück, falls die Browser-ID `neu` ist. Eine Browser-ID ist `new`, wenn sie zum ersten mal erstellt wird und der Client sie also nicht innerhalb einer Anfrage an den Server zurückgeschickt hat.

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, falls es keine aktuelle Browser-ID gibt.

`encodeUrl(self, url)`

Verschlüsselt eine bereitgestellte URL mit der aktuellen ID des Anfragen-Browsers und gibt die Ergebnisse zurück. Beispielsweise gibt der Aufruf encodeUrl('http://foo.com/amethod') das Ergebnis `http://foo.com/amethod?_ZopeId=as9dfu0adfu0ad` zurück.

Benötigte Berchtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, falls es keine aktuelle Browser-ID gibt.

`flushBrowserIdCookie(self)`

Löscht das Cookie mit der Browser-ID vom Browser des Clients, wenn der Namensraum der `cookies`-Browser-ID verwendet wird.

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, wenn der `cookies`-Namensraum zur Zeit des Aufrufs kein Namensraum der Browser-IDs ist.

`getBrowserIdName(self)`

Gibt einen String mit dem Namen der Cookie/Formular-Variablen zurück, die vom aktuellen Browser-ID-Manager als zu suchender Name verwendet wird, um den Wert der Browser-ID zu erhalten. Beispiel: `_ZopeId`.

Benötigte Berechtigung: Access contents information

`isBrowserIdFromForm(self)`

Gibt true zurück, wenn die Browser-ID aus einer Formular-Variablen stammt (Abfrage-String oder -Post).

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, wenn es keine aktuelle Browser-ID gibt.

`hasBrowserId(self)`

Gibt true zurück, wenn es eine Browser-ID für diese Anfrage gibt.

Benötigte Berechtigung: Access contents information

`setBrowserIdCookieByForce(self, bid)`

Setzt das Cookie mit der Browser-ID in jedem Fall auf die Browser-ID `bid`. Dies ist nützlich, wenn sie Browser-ID-Cookies für den selben User auf mehreren Domains einrichten müssen (vielleicht auch nur temporär, indem Abfrage-Strings verwendet werden).

Benötigte Berechtigung: Access contents information

Gibt folgenden Fehler aus: BrowserIdManagerErr, wenn der `cookies`-Namensraum zur Zeit des Aufrufs kein Namensraum für Browser-IDs ist.

Klasse `BrowserIdManagerErr`

Der Fehler, der währen einiger Operationen des Browser-ID-Managers ausgegeben werden kann. Er wird noch ausführlicher in der API-Dokumentation des Browser-ID-Managers beschrieben.

Diese Exception kann in Python-Skripts abgefangen werden. Ein erfolgreicher Import der Exception in Python-Skripts sieht folgendermassen aus:

```
from Products.Sessions import BrowserIdManagerErr
```

Klasse `SessionDataManagerInterface`

Die Schnittstelle von Zopes Session-Data-Manager.

Ein Zope Session-Data-Manager ist verantwortlich für die Aufrechterhaltung von Session-Data-Objekten und für die Bearbeitung von Anfragen aus Anwendungscode an die Session-Data-Objekte. Auch kommuniziert er mit einem Browser-ID-Manager um Informationen über Browser-IDs bereitzustellen..

`getSessionDataByKey(self, key)`

Gibt ein Session-Data-Objekt zurück, das mit einem `Schlüssel` verknüpft wird. Gibt es kein solches Session-Data-Objekt, wird None zurückgegeben.

Benötigte Berechtigung: Access arbitrary user session data

`getSessionData(self, create=1)`

Gibt ein Session-Data-Objekt zurück, das mit der aktuellen Browser-ID verknüpft ist. Gibt es keine aktuelle Browser-ID und create hat den Wert true, wird ein neues Session-Data-Objekt zurückgegeben. Gibt es keine aktuelle Browser-ID und create hat den Wert false, wird None zurückgegeben.

Benötigte Berechtigung: Access session data

`getBrowserIdManager(self)`

Gibt den nächstgelegenstens erreichbaren Browser-ID-Manager zurück.

Ein SessionDataManagerErr tritt auf, wenn kein Browser-ID-Manager gefunden werden kann.

Benötigte Berechtigung: Access session data

`hasSessionData(self)`

Gibt true zurück, wenn ein Session-Data-Objekt, das mit der aktuellen Browser-ID verknüpft ist, im Session-Data-Container gefunden wird. Existiert kein Session-Data-Objekt, wird auch kein neues erstellt.

Benötigte Berechtigung: Access session data

Modul `TransienceInterfaces`

Transient Objects

Klasse `TransientObject`

Transient Objects ("vorübergehende Objekte") sind temporäre Objekte, die in einem temporären Objekt-Container aufbewahrt werden.

Die meiste Zeit über werden Sie ein temporäres Objekt wie ein Dictionary verwenden. Sie können die Schreibweise der Sub-Objekte von Python benutzen:

```
SESSION['foo']=1
foo=SESSION['foo']
del SESSION['foo']
```

Wenn Sie ein temporäres Objekt in Python-basierten Skripts oder DTML verwenden können Sie statt dessen die `get`-, `set`- und `delete`-Methoden verwenden.

Methoden von temporären Objekten werden nicht durch die Sicherheitsrichtlinine geschützt.

Es ist notwendig, veränderliche Sub-Objekte neu zuzuordnen, wenn Sie sie abwandeln. Beispiel:

```
l=SESSION['myList']
l.append('spam')
SESSION['myList']=l
```

Dies ist notwendig, um Ihre Veränderungen zu speichern. Bitte beachten Sie, dass dieser Sachverhalt sogar für veränderbare Sub-Elemente, die von der Persistenz erben, gilt. Persistenz-Klasse.

`delete(self, k)`

Call __delitem__ with key k.

Berechtigung:
Steht immer zur Verfügung.

setLastAccessed(self)

Setzt die letzte Zugriffszeit auf die aktuelle Zeit.

Berechtigung:
Steht immer zur Verfügung.

getCreated(self)

Gibt in der Form der integer-Sekunden seit der letzten Epoche an, wann das temporäre Objekt erstellt wurde.

Berechtigung:
Steht immer zur Verfügung.

values(self)

Gibt eine Sequenz von Wert-Elementen zurück.

Berechtigung:
Steht immer zur Verfügung.

has_key(self, k)

Gibt true zurück, wenn das Element, das vom Schlüssel k referenziert wird, existiert.

Berechtigung:
Steht immer zur Verfügung.

getLastAccessed(self)

Gibt die Zeit zurück, zu der zum letzten Mal auf das temporäre Objekt zugegriffen worden ist. Es wird wieder die Form der Integer-Sekunden seit der letzten Epoche verwendet.

Berechtigung:
Steht immer zur Verfügung.

getId(self)

Gibt eine aussagekräftige und eindeutige ID für das Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

update(self, d)

Verbindet das Dictionary d mit dem dem Dictionary aus "self".

Berechtigung:
Steht immer zur Verfügung.

clear(self)

Entfernt alle Schlüssel/Wert-Paare.

Berechtigung:
Steht immer zur Verfügung.

items(self)

Gibt eine Sequenz von (Schlüssel, Wert)-Elementen zurück.

Berechtigung:
Steht immer zur Verfügung.

keys(self)

Gibt eine Sequenz von Schlüssel-Elementen zurück.

Berechtigung:
Steht immer zur Verfügung.

get(self, k, default=marker)

Gibt Werte verbunden mit dem Schlüssel k zur&uumk;ck. Existiert k nicht und default hat nicht den Wert marker, wird default zurückgegeben, ansonsten wird ein KeyError ausgegeben.

Berechtigung:
Steht immer zur Verfügung.

set(self, k, v)

Ruft __setitem__ mit dem Schlüssel k und dem Wert v auf.

Berechtigung:
Steht immer zur Verfügung.

getContainerKey(self)

Gibt den Schlüssel zurück, unter dem das Objekt in seinem Container gelagert wird. getContainerKey gibt oftmals einen anderen Wert zurück als getId.

Berechtigung:
Steht immer zur Verfügung.

invalidate(self)

De-Validiert das temporäre Objekt, lässt es also verfallen.

Als weiterer Effekt wird die "before destruct"-Methode im temporären Objekt Container, die zu diesem Objekt gehört, aufgerufen.

Berechtigung:
Steht immer zur Verfügung.

Klasse `MaxTransientObjectsExceeded`

Eine vom Modul Products.Transience.Transience importierbare Exception, die auftritt, wenn ein Versuch gestartet wird, ein Element zu einem temporären Objekt-Container hinzuzufügen, der `voll` ist.

Diese Exception kann durch einen normalen Import in Python-Skripts abgefangen werden. Ein erfolgreicher Import der Exception kann folgendermassen erreicht werden:

```
from Products.Transience import MaxTransientObjectsExceeded
```

Klasse `TransientObjectContainer`

TransientObjektContainers enthalten temporäre Objekte, meist sind dies Session-Daten.

Sie werden selten einen temporären Objekt-Container ausarbeiten müssen. Meistens werden Sie mit den temporären Objekten selber arbeiten, die sie normalerweise über `REQUEST.SESSION` erhalten.

`new(self, k)`

Erstellt ein neues Sub-Objekt mit dem Schlüssel k. Das Objekt hat den Typ, der von diesem Container unterstützt wird und wird anschliessend zurückgegeben.

Existiert ein Objekt mit dem Schlüssel k bereits in diesem Container, wird ein KeyError aufgerufen.

"k" muss ein String sein, da sonst ein KeyError ausgelöst wird.

Ist der Container `voll`, wird ein MaxTransientObjectsExceeded-Fehler ausgelöst.

Berechtigung:
`Create Transient Objects`

`setDelNotificationTarget(self, f)`

Setzt die `before destruction`-Funktion auf den Wert `f`.

Kann `f` nicht aufgerufen werden und ist kein String, wird sein Wert als Zope-Pfad zu einer ausführbaren Funktion behandelt.

`before destruction`-Funktionen benötigen ein einzelnes Argument: `item`, was das zu löschende Element beschreibt.

Berechtigung:
`Manage Transient Object Container`

`getTimeoutMinutes(self)`

Gibt die Zahl der Minuten zurück, die für die Inaktivität von Objekten genehmigt ist, bevor diese verfallen.

Berechtigung:
`View management screens`

`has_key(self, k)`

Gibt true zurück, wenn der Container einen Wert enthält, der mit dem Schlüssel k verknüpft ist, ansonsten wird false zurückgegeben.

Berechtigung:
`Access Transient Objects`

`setAddNotificationTarget(self, f)`

Bewirkt, dass die `after add`-Funktion auf den Wert `f` gesetzt wird.

Ist `f` nicht ausführbar, jedoch ein String, wird sein Wert als Zope-Pfad zu einer ausführbaren Funktion behandelt.

`after add`-Funktionen benötigen ein einzelnes Argument: `item`, was das Objekt ist, das in den Container eingefügt werden soll.

Berechtigung:
`Manage Transient Object Container`

`getId(self)`

Gibt eine aussagekräftige und eindeutige ID für das Objekt zurück.

Berechtigung:
Steht immer zur Verfügung.

`setTimeoutMinutes(self, timeout_mins)`

Ändert die Anzahl an Minuten, die ein Sub-Objekt inaktiv sein darf, bevor es verfällt.

Berechtigung:
`Manage Transient Object Container`

`new_or_existing(self, k)`

Wenn bereits ein Objekt mit dem Schlüssel k im Container existiert, wird es zurückgegeben.

Andernfalls wird ein neues Sub-Objekt vom in diesem Container unterstützten Typ mit dem Schlüssel k erstellt und zurückgegeben.

"k" muss ein String sein, ansonsten wird ein TypeError ausgegeben.

Ist der Container `voll`, wird eine MaxTransientObjectsExceeded-Exception ausgegeben.

Berechtigung:
`Create Transient Objects`

`get(self, k, default=None)`

Gibt den Wert zurück, der mit dem Schlüssel k verknüpft ist. Existiert kein solcher Wert, wird der Default-Wert zurückgegeben.

Berechtigung:
`Access Transient Objects`

`getAddNotificationTarget(self)`

Gibt die aktuelle `after add`-Funktion oder aber None zurück.

Berechtigung:
`View management screens`

`getDelNotificationTarget(self)`

Gibt die aktuelle `before destruction`-Funktion oder aber None zurück.

Berechtigung:
`View management screens`

Modul `UserFolder`

Klasse `UserFolder`

UserFolder-Objekte sind Container für Benutzer-Objekte. Programmierer können mit Sammlungen von Benutzer-Objekten arbeiten, indem sie die API verwenden, die von den UserFolder-Implementierungen bereitgestellt wird.

`userFolderEditUser(name, password, roles, domains, **kw)`

Eine API-Methode, mit der die Attribute von Benutzer-Objekten verändert werden können. Bitte beachten Sie, dass nicht alle UserFolder-Implementierungen die Änderung dieser Attribute erlauben. Implementierungen, die sie nicht unterstützen werden einen Fehler für diese Methode ausgeben.

Berechtigung:
Manage users

`userFolderDelUsers(names)`

Eine API-Methode, mit der ein oder mehrere Benutzer-Objekte entfernt werden können. Bitte beachten Sie, dass nicht alle UserFolder-Implementierungen das Löschen von Benutzer-Objekten erlauben. Implementierungen, die dies nicht unterstützen geben einen Fehler für diese Methode aus.

Berechtigung:
Manage users

userFolderAddUser(name, password, roles, domains, **kw)

Eine API-Methode, mit der neue Benutzer-Objekte angelegt werden können. Bitte beachten Sie, dass nicht alle UserFolder-Implementierungen das Erstellen von Benutzer-Objekten erlauben. Implementierungen, die dies nicht unterstützen, geben einen Fehler für diese Methode aus.

Berechtigung:
Manage users

getUsers()

Gibt eine Sequenz aller Benutzer-Objekte zurück, die sich im Benutzer-Ordner befinden.

Berechtigung:
Manage users

getUserNames()

Gibt eine Sequenz von Namen der Benutzer zurück, die sich im Benutzer-Ordner befinden.

Berechtigung:
Manage users

getUser(name)

Gibt das durch den Namen spezifizierte Benutzer-Objekt zurück. Gibt es keinen Benutzer namens `name` im Benutzer-Ordner, wird None zurückgegeben.

Berechtigung:
Manage users

Modul `Vocabulary`

Klasse `Vocabulary`

Ein Vocabulary verwaltet Wort- und Sprachregeln für die Katalogisierung von Text. Text-Katalogisierung wird vom ZCatalog und anderen Produkten von Drittanbietern durchgeführt.

words()

Gibt eine Wortliste zurück.

insert(word)

Fügt ein Wort in das Vocabulary ein.

query(pattern)

Frägt das Vocabulary nach Worten ab, die dem Suchmuster entsprechen.

ObjectManager Constructor

manage_addVocabulary(id, title, globbing=None, REQUEST=None)

Fügt ein Vocabulary-Objekt zum Objekt-Manager hinzu.

Modul `ZCatalog`

Klasse `ZCatalog`

ZCatalog-Objekt

Ein ZCatalog enthält beliebige Indexe wie etwa Referenzen auf Zope-Objekte. ZCatalogs können entweder `Field`-Werte des Objekts indexieren oder aber `Text`-Werte oder `KeyWord`-Werte:

ZCatalogs haben drei Arten von Indexen:

Text
: Text-Indexe indexieren Textinhalte. Der Index kann verwendet werden, um nach Objekten zu suchen, die bestimmte Worte enthalten.

Field
: Field-Indexe indexieren atomare Werte. Der Index kann zur Suche nach Objekten verwendet werden, die bestimmte Eigenschaften haben.

Keyword
: Keyword-Indexe indexieren Wert-Sequenzen. Der Index kann verwendet werden, um nach Objekten zu suchen, die einem oder mehreren Suchausdrücken entsprechen.

Der ZCatalog kann eine Table von zusätzlichen Daten aufrechterhalten, die katalogisierte Objekte beschreiben. Diese Informationen können auf Such-Ergebnisseiten verwendet werden, um Informationen über die Suchergebnisse anzuzeigen.

Das Schema der Tabelle der Meta-Daten wird verwendet, um das Schema für die ZCatalog-Ergebnisobjekte zu erstellen. Die Objekte haben die gleichen Attribute wie die Spalten der Tabelle der Meta-Daten.

ZCatalogs speichern keine Referenzen zum Objekt selber sonder zu einem eindeutigen Bezeichner der definiert, wie man das Objekt bekommt. In Zope ist dieser eindeutige Bezeichner der relative Pfad des Objekts zum ZCatalog (da zwei Zope-Objekte nie die gleiche URL haben können, ist dies der beste eindeutige Bezeichner).

schema()

Gibt eine Sequenz von Namen zurück, die mit den Spalten in der Meta-Daten-Tabelle korrespondieren.

__call__(REQUEST=None, **kw)

Durchsucht den Katalog auf die selbe Art wie `searchResults`.

uncatalog_object(uid)

Entfernt das Objekt mit dem eindeutigen Bezeichner `uid` (nur!) aus dem Katalog.

`getobject(rid, REQUEST=None)`

Gibt ein katalogisiertes Objekt zurück, wenn eine `data_record_id_` angegeben wurde.

`indexes()`

Gibt eine Sequenz von Namen zurück, die mit den Indexen korrespondieren.

`getpath(rid)`

Gibt den Pfad zum katalogisierten Objekt zurück, wenn eine `data_record_id_` angegeben wurde.

`index_objects()`

Gibt eine Sequenz akuteller Index-Objekte zurück.

`searchResults(REQUEST=None, **kw)`

Durchsucht den Katalog. Such-Ausdrücke können im REQUEST oder als Schlüsselwort-Argumente übergeben werden.

Such-Anfragen bestehen aus einer Zuordnung von Indexnamen zu Such-Parametern. Sie können entweder eine Zuordnung an searchResults als Variable `REQUEST` übergeben oder Sie können Indexnamen und Such-Parameter als Schlüsselwort-Argumente der Methode verwenden, in anderen Worten:

```
searchResults(title='Elvis Exposed',
              author='The Great Elvonso')
```

ist das Selbe wie:

```
searchResults({'title' : 'Elvis Exposed',
               'author : 'The Great Elvonso'})
```

In diesen Beispielen sind `title` und `author` Indexe. Diese Anfrage gibt alle Objekte zurück, die den Titel *Elvis Exposed* haben UND vom Autor *The Great Elvonso* sind. Ausdrücke, die als Schlüssel und Werte in einem searchResults()-Aufruf übergeben werden werden ohne Bedingung mit AND verknüpft. Um zwei Sucherergebnisse mit OR zu verknüpfen rufen Sie die searchResults()-Methode zweimal auf und verknüpfen die Ergebnisse wie hier:

```
results = ( searchResults(title='Elvis Exposed') +
            searchResults(author='The Great Elvonso') )
```

Dies gibt alle Objekte zurück, die den Titel ODER den Autor haben.

Es gibt einige spezielle Index-Namen, die Sie übergeben können, um das Verhalten der Such-Abfrage zu verändern:

sort_on
: Dieser Parameter legt fest, nach welchem Index die Resultate sortiert werden sollen.

sort_order
: Sie können festlegen, ob die defaultmässige Sortierreihenfolge `reverse` (rückwärts) oder `descending` (absteigend) sein soll. Wurde hier nichts verändert, wird immer ascending (aufsteigend) sortiert.

Es gibt einige Regeln, die sie beim Benutzen dieser Methode beachten sollten:

- eine leere Suchzuordnung (oder ein simulierter REQUEST) gibt alle Elemente des Katalogs zurück.
- Ergebnisse aus einer Abfrage, die nur Feld/Schlüsselwort-Indexe einbezieht, z.B. {'id':'foo'} und keinen `sort_on` benutzen werden unsortiert zurückgegeben.
- Ergebnisse einer komplexen Abfrage, die einen Feld/Schlüsselwort-Index *und* einen Text-Index enthält, z.B. {'id':'foo','PrincipiaSearchSource':'bar'} und keinen `sort_on` beschreiben werden unsortiert zurückgegeben.
- Ergebnisse einer einfachen Text-Index-Abfrage, z.B. {'PrincipiaSearchSource':'foo'} werden in absteigender Reihenfolge, nach `score` sortiert, zurückgegeben. Ein Text-Index kann nicht als `sort_on`-Parameter verwendet werden, falls Sie dies doch versuchen, tritt ein Fehler auf.

Abhängig vom Typ des Indexes, den sie abfragen, können Sie fortgeschrittene Such-Parameter angeben, die das Suchfeld eingrenzen oder Platzhalter erzeugen. Diese Eigenschaften werden im Zope-Buch beschrieben.

`uniqueValuesFor(name)`

Gibt die eindeutigen Werte für einen angegebenen FieldIndex namens `name` zurück.

`catalog_object(obj, uid)`

Katalogisiert die Objekte `obj` mit dem eindeutigen Bezeichner `uid`.

ObjectManager Constructor

`manage_addZCatalog(id, title, vocab_id=None)`

Fügt ein ZCatalog-Objekt hinzu.

`vocab_id` ist der Name eines Vocabulary-Objektes, das dieser Katalog verwenden soll. Der Wert None wird hier verursachen, dass der Katalog sein eigenes privates Vocabulary erstellt.

Modul `ZSQLMethod`

Klasse `ZSQLMethod`

ZSQL-Methoden abstrahieren SQL-Code in Zope.

SQL-Methoden verhalten sich wie Methoden der Ordner, in denen auf sie zugegriffen wird. Insbesondere können sie von anderen Methoden verwendet werden, beispielsweise Dokumenten, externen Methoden, oder sogar anderen SQL-Methoden.

Datenbank-Methoden unterstützen die Searchable-Objekt-Schnittstelle. Assistenten für Such-Schnittstellen können verwendet werden, um Benutzer-Schnittstellen für sie zu erstellen. Sie können in joins und unions verwendet werden. Sie stellen Meta-Daten über ihre Input-Parameter und Ergebnis-Daten bereit.

Für weitere Informationen beachten Sie bitte die Spezifikationen der Searchable-Objekt-Schnittstelle.

Datenbank-Methoden unterstützen das Traversal von URLs, um auf Methoden individueller Record-Ojbkete zuzugreifen und sie aufzurufen. Zum Beispiel: angenommen, Sie haben eine `employees`-Datenbank-Methode die ein einzelnes Argument `employee_id` aufnimmt. Weiterhin haben die employees eine `service_record`-Methode (definiert in einer Record-Klasse oder aus einem Ordner erhalten). Auf die `service_record`-Methode könnten mit einer URL wie:

```
employees/employee_id/1234/service_record
```

zurgreifen.

Suchergebnisse werden als Record-Objekte zurückgegeben. Das Schema der Record-Objekte entspricht dem Schema der in der Suche abgefragten Tabelle.

`manage_edit(title, connection_id, arguments, template)`

Verändert die Einstellungen der Datenbank-Methoden.

Das `connection_id`-Argument ist die ID einer Datenbank-Verbindung, die sich im aktuellen Ordner oder in einem Ordner über dem aktuellen Ordner befindet. Die Datenbank sollte SQL verstehen.

Das `arguments`-Argument ist ein String, der eine Spezifizierung eines Arguments enthält, wie sie auch im Formular zur Erstellung von SQL-Methoden angegeben würde.

Das `template`-Argument ist ein String, der die Quelle für das SQL-Template enthält.

`__call__(REQUEST=None, **kw)`

Ruft die ZSQL-Methode auf.

Die Argumente zu dieser Methode sollten als Schlüsselwort-Argumente übergeben werden oder aber in einem einzelnen Zuordnungs-Objekt. Werden keine Argumente angegeben und die Methode über das Web aufgerufen, wird die Methode versuchen, das Web-REQUEST-Objekt zu erhalten und als Zuordnungs-Argument zu verwenden.

Die zurückgegebenen Werte sind eine Sequenz von Record-Objekten.

ObjectManager Constructor

manage_addZSQLMethod(id, title, connection_id, arguments, template)

Fügt eine SQL-Methode zum Objekt-Manager hinzu.

Das `connection_id`-Argument ist die ID einer Datenbank-Verbindung, die sich im aktuellen Ordner oder in einem Ordner über dem aktuellen Ordner befindet. Die Datenbank sollte SQL verstehen.

Das `arguments`-Argument ist ein String, der eine Spezifizierung eines Argumentes enthält, wie sie auch im Formular zuer Erstellung einer SQL-Methode angegeben werden würde.

Das `template`-Argument ist ein String, der die Quelle für das SQL-Template enthält.

Modul `ZTUtils`

ZTUtils: Page Template Utilities (Dienstprogramme für Seitenvorlagen)

Die Klassen in diesem Modul sind von Seitenvorlagen aus verwendbar.

Klasse `Batch`

Batch - eine Sektion einer grossen Sequenz.

Sie können Batches verwenden um grosse Sequenzen (wie etwa Suchergebnisse) auf verschiedene Seiten aufzuteilen.

Batches stellen Seitenvorlagen bereit, die vergleichbare Funktionen haben wie die, die in `<dtml-in>` standardmässig enthalten sind.

Sie können auf Elemente eines Batches einfach so zugreifen, wie sie das bei Elementen einer Liste tun, z.B:

```
>>> b=Batch(range(100), 10)
>>> b[5]
4
>>> b[10]
IndexError: list index out of range
```

Batches haben diese öfftentlichen Attribute:

start
: Die Nummer des ersten Elements (von 1 weg gezählt).

first
: Der erste Element-Index (von 0 weg gezählt). Bitte beachten Sie, dass dies das selbe ist wie start - 1.

end
: Die Nummer des letzten Elementes (von 1 weg gezählt).

orphan

Die gewünschte Minimal-Grösse des Batches. Dies kontrolliert, wie Sequenzen in Batches aufgeteilt werden. Käme ein kleineres Batch als die Grösse von orphan vor, würde keine Aufteilung ausgeführt und ein Batch, das grösser ist als die Batch-Grösse käme heraus.

overlap
: Die Zahl an Elementen, die sich zwischen zwei Batches überschneiden.

length
: Die tatsächliche Länge des Batches. Bitte beachten Sie, dass dies ein anderes Ergebnis als size ergeben kann, je nachdem, welchen Wert orphan hat.

size
: Die gewünschte Grösse. Bitte beachten Sie, dass dies ein anderes Ergebnis als die aktuelle Länge des Batches ergeben kann, je nachdem, welchen Wert orphan hat.

previous
: Das vorige Batch oder None, falls dies das erste Batch ist.

next
: Das nächste Batch oder None, falls dies das letzte Batch ist.

`__init__(self, sequence, size, start=0, end=0, orphan=0, overlap=0)`

Erstellt ein neues Batch mit der gegebenen Sequenz und der gewünschten Batch-Grösse.

sequence
: Die komplette Sequenz.

size
: Die gewünschte Batch-Grösse.

start
: Der Index des Anfangs des Batches (von 0 weg gezählt).

end
: Der Index des Endes des Batches (von 0 weg gezählt).

orphan
: Die gewünschte Minimal-Grösse des Batches. Die kontrolliert, wie Sequenzen in Batches aufgeteilt werden. Kommt ein Batch vor, das kleiner ist als die Grösse von orphan, wird keine Aufteilung durchgeführt und ein Batch, das grösser ist als die Batch-Grösse wird zurückgegeben.

overlap
: Die Anzahl an Elementen, die sich zwischen Batches überschneiden.

Modul `math`

math: Python `math`-Modul

Das `math`-Modul stellt trigonomische und andere mathematische Funktionen bereit. Es ist ein standardmässiges Python-Modul.

Da Zope 2.4 wird Python 2.2 benötigt, lesen Sie hierzu bitte die Dokumentation von Python 2.1.

Bitte beachten Sie auch die

Python `math` module-Dokumentation auf Python.org.

module `random`

random: Python `random`-Modul

Das `random`-Modul stellt Funktionen für Pseudo-Zufallszahlen bereit. Mit ihnen können Sie Zufallszahlen erstellen und zufällige Elemente aus Sequenzen auswählen. Dieses Modul ist standardmässig in Python integriert.

Da Zope 2.4 wird Python 2.2 benötigt, lesen Sie hierzu bitte die Dokumentation von Python 2.1.

Bitte beachten Sie auch die

Python `random` module-Dokumentation auf Python.org

module `sequence`

sequence: Modul zum Sortieren von Sequenzen

Dieses Modul stellt eine `sort`-Funktion bereit, die mit DTML, Seitenvorlagen und Python-basierten Skripten verwendet werden kann.

```
def sort(seq, sort)
```

Sortiert die Sequenz *seq* von Objekten anhand des optionalen Sortierschemas *sort*. *sort* ist eine Sequenz von `(key, func, direction)`-Tupeln, die die Sortierreihenfolge beschreiben.

key
: Das Attribut des Objektes, nach dem sortiert wird.

func
: Definiert die Vergleichsfunktion (optional). Erlaubte Werte:
"cmp"
Standardmässige Vergleichsfunktion von Python
"nocase"
Vergleich unterscheidet nicht nach Gross- und Kleinschreibung
"strcoll" or "locale"
String-Vergleich unter Einbeziehung der Lokalen
"strcoll_nocase" or "locale_nocase"
String-Vergleich unter Einbeziehung der Lokalen ohne Beachtung von Gross- und Kleinschreibung
other
Eine spezifizierte, vom Benutzer definierte Vergleichsfunktion, die 1, 0 oder -1 zurückgeben sollte.
direction
definiert die Sortierreihenfolge für den Schlüssel (optional). (Erlaubte Werte: "asc", "desc")

DTML-Beispiele

Sortiert das child-Objekt (unter Benutzung der `objectValues`-Methode) anhand der ID (unter Benutzung der `getId`-Methode), wobei Gross- und Kleinschreibung ignoriert werden:

```
<dtml-in expr="_.sequence.sort(objectValues(),
                               (('getId', 'nocase'),))">
  <dtml-var getId> <br>
</dtml-in>
```

Sortiert die child-Objekte anhand des Titels (ohne Gross- und Kleinschreibung) und des Datums (vom neuesten zum ältesten):

```
<dtml-in expr="_.sequence.sort(objectValues(),
                               (('title', 'nocase'),
                                ('bobobase_modification_time',
                                 'cmp', 'desc')
                               ))">
  <dtml-var title> <dtml-var bobobase_modification_time> <br>
</dtml-in>
```

Beispiele für Seitenvorlagen

Sie können die `sequence.sort`-Funktion in Python-Ausdrücken verwenden, um Objekte zu sortieren. Hier ein Beispiel, dass das DTML-Beispiel von oben darstellt:

```
<table tal:define="objects here/objectValues;
                   sort_on python:(('title', 'nocase', 'asc'),

('bobobase_modification_time', 'cmp', 'desc'));
                   sorted_objects
python:sequence.sort(objects, sort_on)">
  <tr tal:repeat="item sorted_objects">
    <td tal:content="item/title">title</td>
    <td tal:content="item/bobobase_modification_time">
      modification date</td>
  </tr>
</table>
```

Dieses Beispiel läuft über eine sortierte Liste von Objekten und zeichnet für jedes Objekt eine Zeile in einer Tabelle. Die Objekte werden anhand des Titels und des Änderungsdatums sortiert.

Bitte beachten Sie auch:

Python cmp function

Modul `standard`

1. PythonScripts.standard: Dienstanwendungsfunktionen und -klassen

   Die Funktionen und Klassen dieses Moduls können in Python-basierten Skripten, DTML und Seitenvorlagen verwendet werden.

`def structured_text(s)`

Konvertiert einen String vom structured-text-Format zu HTML.

Bitte beachten Sie auch:

Structured-Text Rules

`def html_quote(s)`

Konvertiert Zeichen, in HTML eine besondere Bedeutung haben, zu HTML-Instanzen.

Bitte beachten Sie auch:

Python `cgi` module_in_cgi_module.html `escape`-Funktion.

`def url_quote_plus(s)`

Wie url_quote, aber hier werden auch Leerzeichen durch + ersetzt. Dies wird für die Erstellung von Abfrage-String in einigen Fällen benötigt.

Bitte beachten Sie auch

Python `urllib` module `url_quote_plus`-Funktion.

`def dollars_and_cents(number)`

Stellt einen numerischen Wert mit einem Dollar-Symbol und zwei Nachkommastellen dar.

`def sql_quote(s)`

Konvertiert einfach Anführungszeichen zu doppelten. Dies wird benütigt, um Werte richtig in SQL-Strings einzufügen.

`def whole_dollars(number)`

Stellt einen numerischen Wert mit einem Dollar-Zeichen dar.

`def url_quote(s)`

Konvertiert Zeichen, die in URLs eine besondere Bedeutung haben, in HTML Character-Instanzen um, wobei Dezimalwerte bnutzt werden.

Bitte beachten Sie auch

Python `urllib` module `url_quote`-Funktion.

class `DTML`

DTML - temporäre, durch Sicherheitsrichtlininen eingeschränkte DTML-Objekte

`__init__(source, **kw)`

Erstellt ein DTML-Objekt inklusive Quelltext und Schlüssel-Variablen. Der Quelltext definiert den DTML-Quellinhalt. Die optionalen Schlüsselwort-Argumente definieren Variablen.

`call(client=None, REQUEST={}, **kw)`

Rendert die DTML.

Um ihre Aufgaben zu erfüllen, muss die DTML oftmals verschiedene Namen in Objekte auflösen. Beispiel: wenn der Code <dtml-var ausgeführt wird, versucht die DTML, den Namen `spam` aufzulösen.

Um Namen aufzulösen, müssen Sie einen Namensraum an die DTML übergeben. Dies kann auf verschiedene Arten erreicht werden:

- Indem ein `client`-Objekt übergeben wird - Wird das Argument `client` übergeben, werden Namen als Attribute der Argumente gesucht.
- Indem eine `REQUEST`-Zuordnung übergeben wird - Wird das Argument `REQUEST` übergeben, werden Namen als Elemente eines Arguments gesucht. Ist das Argument keine Zuordnung, wird ein TypeError ausgegeben, sobald ein Suchvorgang gestartet wird.
- Indem Schlüsselwort-Argumente übergeben werden -- Namen und ihre Werte können als Schlüsselwort-Argumente an die Methode übergeben werden.

Der Namensraum, der an ein DTML-Objekt übergeben wird ist die Zusammenstellung dieser drei Methoden. Sie können jede Anzahl von ihnen oder aber keine der drei Methoden übergeben. Namen werden zuerst im Schlüsselwort-Argument gesucht, dann im Client und schliesslich in der Zuordnung.

`def thousand_commas(number)`

Fügt alle drei Stellen links vom Dezimalpunkt ein Komma in Werte ein, die Nummern enthalten. Beispiel: Der Wert "12000 widgets" wird zu "12,000 widgets".

`def newline_to_br(s)`

Konvertiert Zeilenumbrüche zu break-Tags.

Modul `string`

string: Python `string`-Modul

Das `string`-Modul stellt String-Manipulationen, -Konvertierungen und Suchfunktionen bereit. Es ist standardm&auuml;ssig in Python integriert.

Da Zope 2.4 wird Python 2.2 benötigt, lesen Sie hierzu bitte die Dokumentation von Python 2.1.

Bitte beachten Sie auch die

Python `string` module-Dokumentation auf Python.org

Zopebuch: Inhaltsverzeichnis

Anhang C: Zope-Page-Templates-Referenz

Zope-Seitenvorlagen (Zope Page Templates) sind ein Werkzeug zur HTML/XML-Erzeugung. Dieser Anhang ist eine Referenz für die Zope-Seitenvorlagen-Standards: die Tag Attribute Language (TAL), die TAL Expression Syntax (TALES) und die Macro Expansion TAL (METAL).

TAL-Überblick

Der Standard *Template Attribute Language* (TAL) ist eine Attributsprache für das Erzeugen dynamischer Vorlagen. Er ermöglicht es, die Elemente eines Dokuments zu ersetzen, sie zu wiederholen, oder sie wegzulassen.

Die Anweisungen von TAL sind XML-Attribute aus dem TAL-Namensraum. Diese Attribute können auf ein XML- oder HTML-Document angewendet werden, um daraus eine Vorlage zu machen.

Eine **TAL-Anweisung** hat einen Namen (den Attributnamen) und einen Körper (den Attributwert). Zum Beispiel könnte eine `content`-Anweisung so aussehen: `tal:content="string:Hallo"`. Das Element in dem eine Anweisung definiert wird ist dessen **Anweisungselement**. Die meisten TAL-Anweisungen benötigen Ausdrücke, aber die Syntax und Semantik dieser Ausdrücke sind nicht Teil von TAL. TALES wird für diesen Zweck empfohlen.

TAL-Namensraum

Der URI des TAL-Namensraums und das empfohlene Namensraumkürzel sind zur Zeit definiert als:

```
xmlns:tal="http://xml.zope.org/namespaces/tal"
```

Dies ist kein URL, sondern lediglich ein eindeutiger Bezeichner. Erwarten Sie nicht, dass ein Browser ihn erfolgreich auflösen kann.

Zope benötigt keine Deklaration eines XML-Namensraums beim Erstellen von Vorlagen mit dem Inhaltstyp `text/html`. Jedoch benötigt es die Deklaration eines XML-Namensraums für alle anderen Inhaltstypen.

TAL-Anweisungen

Dies sind die TAL-Anweisungen:

- tal:attributes - dynamisch Elementattribute ändern.
- tal:define - Variablen definieren.
- tal:condition - Bedingungen überprüfen.
- tal:content - den Inhalt eines Elements ersetzen.
- tal:omit-tag - ein Element entfernen, nicht jedoch dessen Inhalt.
- tal:on-error - Fehler behandeln.
- tal:repeat - ein Element wiederholen.
- tal:replace - den Inhalt eines Elements ersetzen und das Element entfernen, nicht jedoch dessen Inhalt.

In Anweisungen verwendete Ausdrücke können Werte beliebigen Typs zurückgeben, auch wenn die meisten Anweisungen nur Zeichenketten akzeptieren oder die Werte in eine Zeichenketten-Darstellung überführen. Die Ausdruckssprache muss einen Wert namens nothing definieren, der keine Zeichenkette ist. Dieser Wert ist vor allem nützlich für das Löschen von Elementen und Attributen.

Anweisungsreihenfolge

Wenn nur ein TAL-Anweisung pro Element vorhanden ist, ist die Reihenfolgein der sie ausgeführt werden einfach. Beginnend mit dem Wurzelelement werden die Anweisungen jedes Elements ausgeführt. Danach werden alle seine Kinderknoten besucht um das selbe zu tun.

In gleichen Elementen kann jede Kombination von Anweisungen auftreten, außer dass die Anweisungen `content` und `replace` nicht zusammen auftreten dürfen.

Falls ein Element mehrere Anweisungen besitzt werden sie in dieser Reihenfolge ausgeführt:

1. `define`
2. `condition`
3. `repeat`
4. `content` oder `replace`
5. `attributes`
6. `omit-tag`

Da die Anweisung `on-error` nur bei Auftreten eines Fehlers aufgerufen wird, ist sie in der Liste nicht aufgeführt.

Die Überlegung, die hinter dieser Reihenfolge steht ist folgende: Man möchte oft Variablen festlegen, die in anderen Anweisungen verwendet werden, also kommt `define` als erstes. Gleich danach muss entschieden werden, ob dieses Element überhaupt enthalten sein soll, also kommt `condition` als nächstes. Da die Bedingung von Variablen abhängen kann, die man gerade gesetzt hat kommt es nach `define`. Es ist nützlich verschiedene Teile eines Elements bei jedem Schleifendurchlauf durch andere Werte ersetzen zu können, also kommt `repeat` als nächstes. Es macht keinen Sinn Attribute zu ersetzen und sie danach weguwerfen, also kommt zuletzt `attributes`. Die übrigen Anweisungen fallen zusammen, weil jedes von ihnen das Anweisungselement ersetzen oder verändern.

Siehe auch

TALES-Überblick

METAL-Überblick

tal:attributes

tal:define

tal:condition

tal:content

tal:omit-tag

tal:on-error

tal:repeat

tal:replace

attributes: Elementattribute ersetzen

Syntax

`tal:attributes`-Syntax:

```
        argument             ::= attribut_anweisung [';'
attribut_anweisung]*
        attribut_anweisung   ::= attribut_name ausdruck
        attribut_name        ::= [namensraum ':'] Name
        namensraum           ::= Name
```

Hinweis: Wenn Sie einen Strichpunkt (;) in einem `ausdruck` eingeben wollen, muss er mit einem Strichpunkt als Steuerzeichen versehen werden, was zu einer Verdopplung führt (;;).

Beschreibung

Die Anweisung `tal:attributes` ersetzt den Wert eines Attributs (oder erzeugt ein Attribut) mit einem dynamischen Wert. Sie können ein Attribut mit einem Namensraum - zum Beispiel `html:table` - näher bestimmen, wenn Sie ein XML-Dokument mit mehreren Namensräumen erzeugen. Der Wert jedes Ausdrucks wird gegebenenfalls in eine Zeichenkette umgewandelt.

Wenn der mit einer Attributzuweisung verknüpfte Ausdruck *nothing* ergibt, dann wird das Attribut aus dem Audrucks-Element gelöscht. Wenn der Ausdruck *default* ergibt, dann bleibt das Attribut unverändert. Alle Attributzuweisungen sind unabhängig voneinander. So können Attribute in derselben Anweisung zugewiesen werdne, in der andere Attribute gelöscht werden oder andere belassen werden.

Wenn man `tal:attributes` auf ein Element mit einer aktiven `tal:replace`-Anweisung anwendet, wird die `tal:attributes`-Anweisung ignoriert.

Wenn man `tal:attributes` auf ein Element mit einer `tal:repeat`-Anweisung anwendet, wird die Ersetzung bei jeder Wiederholung des Elements durchgeführt und der Ersetzungsausdruck wird für jede Wiederholung neu berechnet.

Beispiele

Einen Verweis ersetzen:

```
<a href="/beispiel/verweis.html"
 tal:attributes="href here/unter/absolute_url">
```

Zwei Attribute ersetzen:

```
<textarea rows="80" cols="20"
 tal:attributes="rows request/rows;cols request/cols">
```

condition: ein Element bedingt einfügen oder entfernen

Syntax

`tal:condition`-Syntax:

```
argument ::= ausdruck
```

Beschreibung

Die Anweisung `tal:condition` fügt das Anweisungselement nur in die Vorlage ein, wenn die Bedingung erfüllt ist, anderenfalls wird es weggelassen. Wenn ihr Ausdruck den Wert *true* (wahr) ergibt, wird das Element normal weiterverarbeitet, anderenfalls wird das Anweisungselement sofort aus der Vorlage entfernt. Für diese Fälle ist der Wert *nothing* unwahr und *default* hat den selben Effekt, als würde man einen wahren Wert zurückgeben.

Hinweis: Zope betrachtet fehlende Variablen, None, Null, leere Zeichenketten und leere Folgen als unwahr. Alle anderen Werte sind wahr.

Beispiele

Eine Variable überprüfen, bevor sie eingfügt wird (das erste Beispiel prüft auf Existenz und Wahrheit, während das zweite nur auf Existenz prüft):

```
<p tal:condition="request/nachricht | nothing"
 tal:content="request/message">Hierher kommt die Nachricht</p>
```

```
<p tal:condition="exists:request/nachricht"
 tal:content="request/nachricht">Hierher kommt die Nachricht</p>
```

Auf wechselseitige Bedingungen prüfen:

```
<div tal:repeat="item python:range(10)">
<p tal:condition="repeat/item/even">gerade</p>
<p tal:condition="repeat/item/odd">ungerade</p>
</div>
```

content: den Inhalt eines Elements ersetzen

Syntax

`tal:content`-Syntax:

```
argument ::= (['text'] | 'structure') ausdruck
```

Beschreibung

Um nicht ein komplettes Element ersetzen zu müssen, kann man Text oder Strukturen anstatt der Unterelemente mit der Anweisung `tal:content` einfügen. Das Anweisungsargument entspricht genau dem von `tal:replace` und wird in auf die selbe Weise interpretiert. Wenn der Ausdruck *nothing* ergibt, bekommt das Anweisungselement keine Unterelemente. Wenn der Ausdruck *default* ergibt, bleibt der Inhalt des Elements unverändert..

Hinweis: Das Standard-Ersetzungsverhalten ist `text`.

Beispiele

Den Benutzernamen einfügen:

```
<p tal:content="user/getUserName">Fred Farkas</p>
```

HTML/XML einfügen:

```
<p tal:content="structure here/getStory">Hierer kommt
<b>aus</b>gezeichneter Inhalt.</p>
```

Siehe auch

`tal:replace`

define: Variablen definieren

Syntax

`tal:define`-Syntax:

```
argument        ::= define_bereich [';' define_bereich]*
define_bereich  ::= (['local'] | 'global') define_var
define_var      ::= variablenname ausdruck
variablenname   ::= Name
```

Hinweis: Wenn Sie einen Strichpunkt (;) in einem `ausdruck` eingeben wollen, muss er mit einem Strichpunkt als Steuerzeichen versehen werden, was zu einer Verdopplung führt (;;).

Beschreibung

Die Anweisung `tal:define` definiert Variablen. Man kann zwei verschiedene Arten von TAL-Variablen definieren: lokale und globale. Wenn man eine lokale Variable in einerm Anweisungselement definiert, kann man diese Variable lediglich in diesem Element und seinen Unterelementen verwenden. Wenn man eine lokale Variable in einem Unterlement neudefiniert, versteckt die neue Definition die Definition des umschließenden Elements innerhalb des inneren Elements. Wenn man eine globale Variable definiert, kann man sie in jedem Element verwenden, dass nach dem definierenden Element verarbeitet wird. Wenn man eine globale Variable neudefiniert, ersetzt man ihre Definition für den Rest der Vorlage.

Hinweis: Standardmäßig werden lokale Variablen verwendet

Wenn der mit einer Variable verbundene Ausdruck *nothing* ergibt, hat die Variable den Wert *nothing* und darf als solcher in weiteren Ausdrücken verwendet werden. Wenn der Ausdruck *default* ergibt, hat die Variable ebenso den Wert *default* und darf ebenso als solcher in weiteren Ausdrücken verwendet werden.

Beispiele

Eine globale Variable definieren:

```
tal:define="global fimenname string:Zope Corp, Inc."
```

Zwei Variablen definieren, wobei die zweite von der ersten abhängt:

```
tal:define="meintitel template/title; tlaenge
python:len(meintitel)"
```

omit-tag: ein Element entfernen, nicht jedoch dessen Inhalt

Syntax

`tal:omit-tag`-Syntax:

```
argument ::= [ ausdruck ]
```

Beschreibung

Die Anweisung `tal:omit-tag` lääst den Inhalt eines Tags unberührt, während es die umgebendenden Start- und End-Tags weglässt.

Wenn sein Ausdruck *false* ergibt, wird mit der normalen Verarbeitung des Tags fortgefahren, ohne dass das Tag weggelassen wird. Ergibt sein Ausdruck einen wahren Wert (*true*) oder es gibt keinen Ausruck, wird das Anweisungs-Tag durch seinen Inhalt ersetzt.

Zope behandelt leere Zeichenketten, leere Sequenzen, eine Null, None, *nothing* und *default* als unwahr (*false*). Alle anderen Werte werden als wahr (*true*) erachtet.

Beispiele

Weglassen eines Tags ohne Bedingung:

```
<div tal:omit-tag="" comment="Dieses Tag wird entfernt">
  <i>... aber dieser Text wird bleiben.</i>
</div>
```

Weglassen eines Tags mit Bedingung:

```
<b tal:omit-tag="not:fett">Ich darf fettgedruckt sein.</b>
```

Das obige Beispiel wird das Tag `b` weglassen, wenn die Variable `bold` false ist.

Zehn Absatz-Tags (`p`) ohne das umschließende Tag erzeugen:

```
<span tal:repeat="n python:range(10)"
      tal:omit-tag="">
  <p tal:content="n">1</p>
</span>
```

on-error: Fehler behandeln

Syntax

`tal:on-error`-Syntax:

```
argument ::= (['text'] | 'structure') ausdruck
```

Beschreibung

Die Anweisung `tal:on-error` ermöglicht Ihrer Vorlage Fehlerbehandlung. Wenn eine TAL-Anweisung einen Fehler erzeugt, sucht der TAL-Interpreter nach einer `tal:on-error`-Anweisung im selben Element, dann im umschließenden Element und so weiter. Das erste `tal:on-error` das gefunden wird, wird aufgerufen. Es wird wie eine `tal:content`-Anweisung behandelt.

Eine lokale Variable, `error`, wird gesetzt. Diese Variable hat folgende Attribute:

`type`
: der Ausnahmetyp

`value`
: die Ausnahmeinstanz

`traceback`
: das Traceback-Objekt

Die einfachste Art einer `tal:on-error`-Anweisung hat einen Fehlertext oder *nothing* als Ausdruck. Eine komplexere Behandlungsroutine könnte ein Skript aufrufen, das den Fehler untersucht und entweder einen Fehlertext ausgibt oder eine Ausnahme wirft, um den Fehler nach außen weiterzugeben.

Beispiele

Einfache Fehlermeldung:

```
<b tal:on-error="string: Benutzername ist nicht definiert!"
 tal:content="here/getUsername">Ishmael</b>
```

Elemente bei Fehlern entfernen:

```
<b tal:on-error="nothing"
   tal:content="here/getUsername">Ishmael</b>
```

Ein Skript zur Fehlerbehandlung aufrufen:

```
<div tal:on-error="structure here/fehlerSkript">
  ...
</div>
```

So könnte ein Fehlerbehandlungsskript aussehen:

```
## Script (Python) "fehlerBehandlung"
##bind namespace=_
##
fehler=_['error']
if fehler.type==ZeroDivisionError:
    return "<p>Durch 0 kann nicht geteilt werden.</p>"
else:
    return """<p>Ein Fehler ist aufgetreten.</p>
              <p>Fehlertyp: %s</p>
              <p>Fehlerwert: %s</p>""" % (error.type,
                                          error.value)
```

Siehe auch

Python Tutorial: Errors and Exceptions

Python Built-in Exceptions

repeat: ein Element wiederholen

Syntax

`tal:repeat`-Syntax:

```
argument      ::= variablenname ausdruck
variablenname ::= Name
```

Beschreibung

Die Anweisung `tal:repeat` repliziert einen Unterbaum Ihres Dokuments für jedes Element in einer Sequenz. Der Ausdruck sollte eine Sequenz ergeben. Wenn die Sequenz leer ist, wird das Anweisungselement gelöscht, wenn nicht wird es für jeden Wert in der Sequenz wiederholt. Wenn der Ausdruck *default* ist, wird das Element unverändert belassen und es werden keine neuen Variablen definiert.

`variablenname` wird verwendet, um eine lokale Variable und eine Wiederholungsvariable zu definieren. Bei jeder Wiederholung wird die lokale Variable auf das aktuelle Sequenzelement gesetzt und die Wiederholungsvariable wird auf ein Iterationsobjekt gesetzt.

Wiederholungsvariablen

Wiederholungsvariablen verwendet man, um auf Informationen über den aktuellen Wiederholungsdurchgang (wie den Wiederholdungsindex) zuzugreifen. Die Wiederholungsvariable hat den selben Namen wie die lokale Variable, ist aber nur über die feste Variable `repeat` ansprechbar.

Die folgenden Informationen sind über die Wiederholungsvariable verfügbar:

- *index* - Anzahl der Wiederholungen, beginnend bei Null.

- *number* - Anzahl der Wiederholungen, beginnend bei Eins.
- *even* - ist true bei Wiederholungen mit geradem Index (0, 2, 4, ...).
- *odd* - ist true bei Wiederholungen mit ungeradem Index (1, 3, 5, ...).
- *start* - ist true beim Anfangsdurchlauf (Index 0).
- *end* - ist true beim End-, also beim letzten Durchlauf.
- *first* - ist true beim ersten Element einer Gruppe - siehe Anmwerkung unten
- *last* - ist true beim letzten Element einer Gruppe - siehe Anmwerkung unten
- *length* - Länge der Sequenz, was gleichzeitig auch der Gesamtzahl der Wiederholungen entspricht.
- *letter* - Anzahl der Wiederholungen als Kleinbuchstabe: "a" - "z", "aa" - "az", "ba" - "bz", ..., "za" - "zz", "aaa" - "aaz", usw.
- *Letter* - Genau wie *letter*, nur mit Großbuchstaben.
- *roman* - Anzahl der Wiederholungen kleine römische Zahlen: "i", "ii", "iii", "iv", "v", etc.
- *Roman* - Genau wie *roman*, nur mit Großbuchstaben.

Sie können auf den den Inhalt der Wiederholungsvariable mit Pfadausdrücken oder Python-Ausrücken zugreifen. In Pfadausrücken schreiben Sie einen dreiteiligen Pfad bestehend aus dem Namen `repeat`, dem Namen der Anweisungsvariablen und dem Namen der Information die sie abfragen wollen, zum Beispiel `repeat/item/start`. In Python-Ausdrücken benutzen Sie die normale Dicitionary-Schreibweise um die Wiederholungsvariable zu bekommen, dann Attributzugriff um die Information zu erhalten, zum Beispiel "python:repeat['item'].start".

Beachten Sie, dass `first` und `last` für die Verwendung in sortierten Listen gedacht sind. Sie versuchen die Sequenz in Gruppen von Element mit gleichem Wert einzuteilen. Wenn Sie einen Pfad verwenden, wird der Wert, der sich durch Verfolgen dieses Pfads ergibt, für die Gruppierung verwendet. Andernfalls wird der Wert des Elements verwendet. Sie können den Pfad bereitstellen, indem Sie ihn als Parameter übergeben, wie bei "python:repeat['item'].first('color')" oder indem Sie ihn an den Pfad der Wiederholungsvariable anhängen, wie bei "repeat/item/first/color".

Beispiele

Über eine Sequenz von Zeichenketten iterieren:

<p tal:repeat= "txt python:'one', <code>two</code>, 'three'"> </p>

Einfügen einer Reihe von Tabellenzeilen und verwenden der Wiederholungsvariablen um die Zeilen zu nummerieren:

```
<table>
  <tr tal:repeat="item here/einkaufswagen">
      <td tal:content="repeat/item/number">1</td>
      <td tal:content="item/beschreibung">Ding</td>
      <td tal:content="item/preis">1,50 €</td>
  </tr>
</table>
```

Verschachtelte Wiederholungen:

```
<table border="1">
  <tr tal:repeat="zeile python:range(10)">
    <td tal:repeat="spalte python:range(10)">
      <span tal:define="x repeat/zeile/number;
                        y repeat/spalte/number;
                        z python:x*y"
            tal:replace="string:$x * $y = $z">1 * 1 = 1</span>
    </td>
  </tr>
</table>
```

Objekte einfügen. Teilt Objekte anhand des Metatyps in Gruppen ein, indem es eine Linie dazwischen setzt:

```
<div tal:repeat="objekt objekte">
  <h2 tal:condition="repeat/objekt/first/meta_type"
    tal:content="objekt/meta_type">Meta-Typ</h2>
  <p tal:content="objekt/getId">Objekt-ID</p>
  <hr tal:condition="repeat/objekt/last/meta_type" />
</div>
```

Beachten Sie, dass die Objekte im obigen Beispiel bereits nach dem Metatyp sortiert sein sollten.

replace: ein Element ersetzen

Syntax

`tal:replace`-Syntax:

```
argument ::= (['text'] | 'structure') ausdruck
```

Beschreibung

Die Anweisung `tal:replace` ersetzt ein Element mit dynamischem Inhalt. Sie ersetzt das Anweisungselement entweder mit Text oder einer Struktur (Code, bei dem die Steuerzeichen nicht ersetzt wurden). Der Körper der Anweisung ist ein Ausdruck mit optionalem Typ-Präfix. Der Wert des Ausrucks wird in eine Zeichenkette mit ersetzten Steuerzeichen konvertiert, wenn man dem Ausdruck `text` voranstellt oder das Präfix weglässt und er wird unverändert eingefügt, wenn man `structure` voranstellt. Beim Ersetzen von Steuerzeichen wird "&" in "&", "<" in "<", und ">" in ">" umgewandelt.

Wenn der Wert *nothing* ist, wird das Element einfach entfernt. Wenn der Wert *default* ist, bleibt das Element unverändert.

Beispiele

Zwei Arten den Titel einer Vorlage einzufügen:

```
<span tal:replace="template/title">Titel</span>
<span tal:replace="text template/title">Titel</span>
```

HTML/XML einfügen:

```
<div tal:replace="structure tabelle" />
```

Nichts einfügen:

```
<div tal:replace="nothing">Dieses Element is ein Kommentar.</div>
```

Siehe auch

`tal:content`

TALES-Überblick

Der Standard *Template Attribute Language Expression Syntax* (TALES) beschreibt Ausdrücke die TAL und METAL Daten liefern. TALES ist *eine* mögliche Ausdruckssyntax für diese Sprachen, sie sind jedoch nicht an diese Definition gebunden. Ebenso könnte TALES in Situationen verwendet werden, in denen es nicht mit TAL oder METAL in Berührung kommt.

TALES-Ausdrücke werden weiter unten ohne jegliches Trennsymbol oder Anführungszeichen-Code aus höhergelegenen Sprachebenen beschrieben. Hier ist die grundlegende Definition der TALES-Syntax:

```
Ausdruck   ::= [typpraefix ':'] Zeichenkette
typpraefix ::= Name
```

Hier sind einige einfache Beispiele:

```
a/b/c
path:a/b/c
nothing
path:nothing
python: 1 + 2
string:Hallo ${user/getUserName}
```

Das optionale *Typpräfix* legt die Semantik und Syntax der nachfolgenden *Ausdruckszeichenkette* fest. Eine TALES-Implementation kann eine beliebige Zahl von Asudruckstypen definieren, mit jedweder Syntax. Sie legt auch fest, welcher Ausruckstyp beim Weglassen des Präfixes verwendet wird.

Wenn man kein Präfix angibt, nimmt Zope an, dass es sich um einen *path*-Ausdruck handelt.

TALES-Ausdruckstypen

Dies sind die von Zope unterstützten TALES-Ausdruckstypen:

- path-Ausdrücke - finden einen Wert anhand seines Pfads.
- exists-Ausdrücke - überprüfen, ob ein Pfad gültig ist.
- nocall-Ausdrücke - finden ein Objekt anhand seines Pfads.
- not-Ausdrücke - negieren einen Ausdruck
- string-Ausdrücke - formatieren eine Zeichenkette
- python-Ausdrücke - führen einen Python-Ausdruck aus

Feststehende Namen

Dies sind die Namen die TALES-Ausdrücken in Zope ständig zur Verfügung stehen:

- *nothing* - spezieller Wert, der einen *Nicht-Wert* darstellt (z. B. void, None, NULL).
- *default* - spezieller Wert, der bestimmt, dass vorhandener Text nicht ersetzt werden soll. Für Einzelheiten darüber, wie *default* von den jweiligen TAL-Anweisungen interpretiert wird, sehen Sie auch in deren der Dokumentation nach.
- *options* - Die *Schlüsselwort*-Argumente die der Vorlage übergeben wurden. Diese sind üblicherweise eher verfügbar, wenn eine Vorlage von Methoden und Skript aufgerufen wird, als wenn der Aufruf vom Web kam.
- *repeat* - die `repeat`-Variablen; Siehe tal:repeat-Dokumentation.
- *attrs* - Ein Dictionary, das die anfänglichen Werte der Attribute des aktuellen Anweisungs-Tags enthält.
- *CONTEXTS* - die Liste der feststehenden Namen (diese Liste). Dies kann verwendet werden, um auf eine feststehende Variable zuzugreifen, die durch eine lokale oder globale Variable gleichen Namens verdeckt wurde.
- *root* - Das oberste Objekt im System: der Zope-Wurzelordner.
- *here* - das Objekt auf das die Vorlage gerade angewandt wird.
- *container* - Der Ordner in dem sich die Vorlage befindet.
- *template* - Die Vorlage selbst.
- *request* - das Objekt der Veröffentlichungsanfrage.
- *user* - das Objekt des autentifizierten Benutzers.
- *modules* - eine Sammlung durch die Python-Module und -Pakete angesprochen werden können. Es können nur Module angesprochen werden, die vom Zope-Sicherheitssystem (der "Zope security policy") freigegeben wurden.

Beachten Sie, dass die Namen `root`, `here`, `container`, `template`, `request`, `user` und `modules` optionale Namen sind, die von Zope unterstützt werden. Sie werden vom TALES-Standard jedoch nicht benötigt.

Siehe auch

TAL-Überblick

METAL-Überblick

exists-Ausdrücke

nocall-Ausdrücke

not-Ausdrücke

string-Ausdrücke

path-Ausdrücke

python-Ausdrücke

TALES-Exists-Ausdrücke

Syntax

Syntax des Ausdrucks Exists:

```
exists-ausdruck ::= 'exists:' path-ausdruck
```

Beschreibung

Exists-Ausdrücke überprüfen die Existenz von Pfaden. Ein Exists-Ausdruck gibt true zurück, wenn die auf ihn folgenden Path-Ausdrücke einen Wert liefern. Er ist false, wenn der Path-Ausdruck kein Objekt finden kann.

Beispiele

Überprüfung der Existenz einer Fomularvariable:

```
<p tal:condition="not:exists:request/form/nummer">
  Bitte geben Sie eine Zahl zwischen 0 und 5 ein
</p>
```

Beachten Sie, dass Sie in diesem Fall den Ausdruck `not:request/form/nummer` nicht verwenden können, da dieser Ausdruck true wäre, wenn die Varaible `nummer` existiert und Null ist.

TALES-Nocall-Ausdrücke

Syntax

Syntax des Ausdrucks Nocall:

```
nocall-ausdruck ::= 'nocall:' path-ausdruck
```

Beschreibung

Nocall-Ausdrücke verhindern die Interpretation der Ergebnisse eines Path-Ausdrucks.

Ein gewöhnlicher Path-Ausdruck versucht das Objekt, das er liefert zu interpretieren (rendern). Das bedeutet, dass, wenn das Objekt eine Funktion, ein Skript, eine Methode oder etwas anderes ausführbares ist, der Ausdruck zum Ergebnis des aufgerufenen Objekts wird. Dies will man normalerweise auch, aber nicht immer. Wenn man zum Beispiel ein DTML-Dokument in einer Variable speichern will, um dessen Eigenschaften ansprechen zu können, kann man keinen normalen Path-Ausdruck verwenden, weil es das Dokument in eine Zeichenkette umwandelt.

Beispiele

Verwendung von Nocall, um die Eigenschaften eines Dokument zu erhalten:

```
<span tal:define="dok nocall:here/einDok"
      tal:content="string:${dok/getId}: ${dok/title}">
Id: Titel</span>
```

Verwendung von Nocall bei einer Funktion:

```
<p tal:define="join nocall:modules/string/join">
```

Dieses Beipiel definiert eine Variable `join`, die mit der Funktion `string.join` verknüpft ist.

TALES-Not-Ausdrücke

Syntax

Syntax des Ausdrucks Not:

```
not-ausdruck ::= 'not:' ausdruck
```

Beschreibung

Not-Ausdrücke berechnen die Ausdruckszeichenkette (rekursiv) als kompletten Ausdruck und geben die boolesche Negation von dessen Wert zurück. Wenn der übergebene Ausdruck keinen booleschen Wert ergibt, gibt *not* eine Warnung und *zwingt* den Wert des Ausdrucks basierend auf den folgenden Regeln in einen booleschen Typ:

1. die Zahl 0 ist *false*
2. Zahlen > 0 sind *true*
3. eine leere Zeichenkette oder andere Folgen sind *false*
4. eine nicht-leere Zeichenkette oder andere Folgen sind *true*
5. ein *Nicht-Wert* (z. B. void, None, Nil, NULL, etc.) ist *false*
6. alle anderen Werte sind abhängig von der Implementation.

Wenn keine Ausdruckszeichenkette übergeben wird, sollte ein Fehler erzeugt werden.

Zope betrachtet alle Objekte, die nicht explizit oben als *false* aufgelistet wurden (einschließlich negativer Zahlen) als *true*.

Beispiele

Testen einer Folge:

```
<p tal:condition="not:here/objectIds">
  Es existieren keine enthaltenen Objekte.
</p>
```

TALES-Path-Ausdrücke

Syntax

Syntax des Path-Ausdrucks:

```
PathAusdr ::= Pfad [ '|' Pfad ]*
Pfad      ::= Variable [ '/' URL-Segment ]*
Variable  ::= Name
```

Beschreibung

Ein Path-Ausdruck besteht aus einem oder mehreren *Pfaden* getrennt durch vertikale Striche (|). Ein Pfad besteht aus einem oder mehreren nicht-leeren Zeichenketten getrennt durch Schrägstriche (/). Der erste String muss ein Variablenname sein (einer feststehenden oder benutzerdefinierten Variable) und die übrigen Zeichenketten, die *Pfadsegmente*, dürfen Buchstaben, Ziffern, Leerzeichen und die Satzzeichen Unterstrich, Bindestrich, Punkt, Komma und Tilde enthalten.

Zum Beispiel:

```
request/cookies/hafermehl
nothing
here/eine_datei 2001_02.html.tar.gz/foo
root/zu/zweig | default
request/name | string:Anonymer Feigling
```

Wenn ein Path-Ausdruck berechnet wird, versucht Zope den Pfad von links nach rechts zu durchlaufen, bis es am Ende angelangt ist oder keine Pfadsegmente mehr hat. Um einen Pfad zu durchlaufen, holt es sich zuerst das Objekt, das in der Variable gespeichert ist. Für jedes Pfadsegment läuft es vom aktuellen Objekt bis zum Unterobjekt, das durch das Pfadsegment angegeben wird. Unterobjekte werden über die Standard-Zope-Traversierungsregeln (mit getattr, getitem, oder Traversal-Hooks) ausfindig gemacht..

Ist ein Pfad einmal erfolgreich durchlaufen, ist das daraus entstandene Objekte der Wert des Ausdrucks. Wenn es ein aufrufbares Objekt ist, wie etwa eine Methode oder eine Vorlage, wird es aufgerufen.

Wenn ein Druchlaufschritt fehlschlägt, wird die Berechnung beim nächsten Pfad fortgesetzt. Wenn keine weiteren Pfade existieren, ist ein Fehler das Ergebnis.

Der Ausdruck in einer Reihe von Pfaden, getrennt durch vertikale Striche, kann ein beliebiger TALES-Ausdruck sein. Zum Beispiel 'request/name | string:Anonymer Feigling'. Dies ist vor allem nützlich, wenn man Standardwerte bereitstellen will, wie Zeichenketten oder Zahlen, die nicht als Path-Ausdrücke ausgedrückt werden können.

Wenn kein Pfad angegeben wird, ist das Ergebnis *nothing*.

Da jeder Pfad mit einem Variablennamen beginnen muss, benötigt man eine Reihe von Startvariablen, die man verwenden kann, um andere Objekte und Werte aufzufinden. Für eine Liste feststehender Varaiblen sehen Sie im TALES-Überlick nach. Da Variablennamen zuerst in den lokalen Variablen, dann in den globalen Variablen und dann erst in dieser Liste gesucht werden, verhalten sich diese Namen genau wie feststehende Variablen in Python: Sie sind immer verfügbar, aber sie können durch Deklaration globaler oder lokaler Variablen verdeckt werden. Man kann die feststehenden Namen immer explizit durch Voranstellen von *CONTEXTS* ansprechen (Zum Beispiel CONTEXTS/root, CONTEXT S/nothing, etc.).

Beispiele

Eine Cookie-Variable oder eine Eigenschaft einfügen:

```
        <span tal:replace="request/cookies/voreinstellung |
here/voreinstellung">
          Voreinstellung
        </span>
```

Den Benutzernamen einfügen:

```
        <p tal:content="user/getUserName">
          Benutzername
        </p>
```

TALES-Python-Ausdrücke

Syntax

Syntax des Python-Ausdrucks:

```
        Jeder gültige Ausdruck der Python-Sprache
```

Beschreibung

Python-Ausdrücke berchnen Python-Code in einer sicherheithalber eingeschränkten Umgebung. PYthon-Ausdrücke bieten die selben Möglichkeiten wie diejenigen, die in python-basierten Skripten und DTML-Variablenausdrücken zur Verfügung stehen.

Sicherheitsbeschränkungen

Python-Ausdrücke unterliegen den selben Sicherheitsbeschränkungen wie python-basierte Skripte. Diese Beschränkungen beinhalten:

Zugriffsgrenzen
: Pyhon-Ausdrücke unterliegen den Berechtigungs- und Rollen-Siucherheitsbeschränkungen von Zope. Darüberhinaus können Ausdrücke keine Objekte ansprechen, deren Name mit einem Unterstrich beginnt.

Schreibgrenzen
: Python-Ausdrücke können keine Attribute von Zope-Objekten ändern.

Trotz dieser Einschränkungen können böswillige Python-Ausdrücke Probleme verursachen. Für weitere Informationen sehen Sie im Zope-Buch nach.

Feststehende Funktionen

Python-Ausdrücke haben die selben feststehenden Funktionen wie python-basierte Skripte mit einige zusätzliche.

Diese Standard-Python-Ausdrücke sind verfügbar: `None`, `abs`, `apply`, `callable`, `chr`, `cmp`, `complex`, `delattr`, `divmod`, `filter`, `float`, `getattr`, `hash`, `hex`, `int`, `isinstance`, `issubclass`, `list`, `len`, `long`, `map`, `max`, `min`, `oct`, `ord`, `repr`, `round`, `setattr`, `str`, `tuple`.

Die Funktionen `range` und `pow` sind zwar verfügbar und funktionieren genau wie im Standard-Python, jedoch sind sie wurden sie so eingeschränkt, dass sie keine sehr großen Zahlen und Sequenzen erzeugen können. Diese Einschränkung hilft, sich vor Denial-Of-Service-Angriffe zu schützen.

Zusätzlich sind diese Hilfsfunktionen verfügbar: `DateTime`, `test` und `same_type`. Für weitere Informationen über diese Funktionen sehen Sie unter DTML-Funktionen nach.

Schließlich gibt es diese Funktionen in Python-Ausdrücken, jedoch nicht in python-basierten Skripten:

`path(Zeichenkette)`
: Berechnet einen TALES-Path-Ausdruck.

`string(Zeichenkette)`
: Berechnet einen TALES-String-Ausdruck.

`exists(Zeichenkette)`
: Berechnet einen TALES-Exists-Ausdruck.

`nocall(Zeichenkette)`
: Berechnet einen TALES-Nocall-Ausdruck.

Python-Module

Viele Python-Module sind standardmäßig verfügbar. Sie können weitere Module verfügbar machen. Man kan Module entweder über Path-Ausdrücke (zum Beipiel 'modules/string/join') oder in Python über das Zuordnungsobjekt `modules` (zum Beipiel 'modules["string"].join') ansprechen. Hier sind die Standardmodule:

`string`
: Das Standard-Python-Modul String. Hinweis: Die meisten der Funktionen im Modul sind auch als Methoden auf String-Objekte verfügbar.

`random`
: Das Standard-Python-Modul Random.

`math`
: Das Standard-Python-Modul Math.

`sequence`
: Ein Modul mit einer mächtigen Sortierfunktion. Für weitere Informationen sehen Sie unter "sequence" nach.

`Products.PythonScripts.standard`
: Verschiedene Funktionen zur HTML-Formattierung aus DTML. Für weitere Informationen sehen Sie unter "Products.PythonScripts.standard" nach.

`ZTUtils`
: Funktionen zur Stapelverarbeitung, ähnlich denen die `dtml-in` bietet. Für weitere Informationen sehen Sie unter "ZTUtils" nach.

`AccessControl`
: Sicherheits- und Zugriffsüberprüfungs-Funktionen. Für weitere Informationen sehen Sie unter "AccessControl" nach.

Beispiele

Verwendung eines Moduls (eine zufällige Auswahl aus einer Liste treffen):

```
<span tal:replace="python:modules['random'].choice(['eins',
                      'zwei', 'drei', 'vier', 'fünf'])">
  Eine zufällige Zahl zwischen eins und fünf
</span>
```

Zeichenkettenverarbeitung (den Benutzernamen in Großbuchstaben umwandeln):

```
<p tal:content="python:user.getUserName().capitalize()">
  Benutzername
</p>
```

Grundlegende Mathematik (eine Bildgröße in Megabytes umwandeln):

```
<p tal:content="python:image.getSize() / 1048576.0">
  12.2323
</p>
```

Zeichenkettenformattierung (eine Gleitkommazahl auf zwei Nachkommastellen formattieren):

```
<p tal:content="python:'%0.2f' % groesse">
  13.56
</p>
```

TALES-String-Ausdrücke

Syntax

Syntax des String-Ausdrucks:

```
string-ausdruck  ::= ( reiner_string | [ varsub ] )*
varsub           ::= ( '$' Pfad ) | ( '${' Pfad '}' )
reiner_string    ::= ( '$$' | kein_dollar )*
kein_dollar      ::= alle zeichen außer '$'
```

Beschreibung

String-Ausdrücke interpretieren die Ausdruckszeichenkette als Text. Wenn keine Ausdruckszeichenkette übergeben wird, ist die resultierende Zeichenkette *leer*. Die Zeichenkette kann Variablenersetzungen der Form `$Name` oder `${Pfad}` enthalten, wobei *Name* ein Variablenname ist und *Pfad* ein Path-Ausdruck. Der Wert des Path-Ausdrucks wird als Zeichenkette mit ersetzten Sonderzeichen in die Zeichenkette eingefügt. Um zu verhindern, dass ein `$` entsprechend interpretiert wird, muss es als `$$` geschrieben werden.

Beispiele

Grundlegende Formattierung einer Zeichenkette:

```
<span tal:replace="string:$dies und $das">
  Spam und Eier
</span>
```

Pfade verwenden:

```
<p tal:content="Gesamt: ${request/form/gesamt}">
  Gesamt: 12
</p>
```

Verwendung eines Dollar-Zeichens:

```
<p tal:content="Kosten: $$$cost">
  Kosten: $42.00
</p>
```

METAL-Überblick

Der Standard *Macro Expansion Template Attribute Language* (METAL) dient zur Vorverarbeitung von HTML/XML-Makros. Es kann zusammen mit oder unabhängig von TAL und TALES verwendet werden.

Makros bieten eine Möglichkeit, ein Stück zur Darstellung in einer Vorlage zu definieren und es mit anderen Vorlagen zu teilen, sodass sich Änderungen am Makro sofort auf alle Stellen auswirken die es verwenden. Darüberhinaus werden Makros immer als vollständig ausgeschriebener Code dargestellt, sogar im Quelltext einer Vorlage, sodass die Vorlage ihrer endgültigen Darstellung sehr nahe kommt.

METAL-Namensraum

Der METAL-Namensraum-URI and der empfohlene Alias sind derzeit folgendermaßen festgelegt:

```
xmlns:metal="http://xml.zope.org/namespaces/metal"
```

Wie auch der TAL-Namensraum-URI, ist dieser URI nicht an eine Webseite gebunden, er ist lediglich ein eindeutiger Bezeichner.

Zope benötigt keine Deklaration eines XML-Namensraums wenn Vorlagen mit dem Inhaltstyp `text/html` erzeugt werden. Für alle anderen Inhaltstypen wird jedoch eine die Deklaration eines XML-Namensraums benötigt.

METAL-Anweisungen

METAL definiert mehrere Anweisungen:

- metal:define-macro - Ein Makro definieren.
- metal:use-macro - Ein Makro verwendet.
- metal:define-slot - Einen anpassbaren Makropunkt definieren.
- metal:fill-slot - Ein Makro anpassen.

Obwohl METAL die Syntax nicht-terminaler Anweisungselemente nicht festlegt, sondern der Implementierung überlässt, wird in der TALES-Spezifikation eine kanonische Ausdruckssyntax für die Verwendung in METAL-Argumenten beschrieben.

Siehe auch

TAL-Überblick

TALES-Überblick

metal:define-macro

metal:use-macro

metal:define-slot

metal:fill-slot

define-macro: ein Makro definieren

Syntax

`metal:define-macro`-Syntax:

```
argument ::= Name
```

Beschreibung

Die Anweisung `metal:define-macro` definiert ein Makro. Der Name des Makros wird durch den Anweisungsausdruck festgelegt und ist definiert als das Element und dessen Unterbaum.

In Zope ist eine Makro-Definition als Unterobjekt des Objekts `macros` einer Vorlage verfügbar. Um zum Beispiel ein Makro namens `kopf` in einer Volage namens `haupt.html` anzusprechen, könnte man den Path-Ausdruck `haupt.html/macros/kopf` verwenden.

Beispiele

einfache Makro-Definition:

```
<p metal:define-macro="copyright">
  Copyright 2001, <em>Foobar</em> GmbH
</p>
```

Siehe auch

metal:use-macro

metal:define-slot

define-slot: einen anpassbaren Makropunkt definieren

Syntax

`metal:define-slot`-Syntax:

```
argument ::= Name
```

Beschreibung

Die Anweisung `metal:define-slot` definiert einen anpassbaren Makropunkt oder *Slot* (dt. Einschub). Wenn ein Makro verwendet wird, können dessen Slots ersetzt werden, um das Makro anzupassen. Alot-Definitionen stellen Standardinhalt für den Slot bereit. Man erhält den Standardinhalt, falls man sich entscheidet das Makro nicht anzupassen wenn man es benutzt.

Die Anweisung `metal:define-slot` muss innerhalb einer `metal:define-macro`-Anweisung verwendet werden.

Slot-Namen müssen innerhlab eines Makros eindeutig sein.

Beispiele

Einfaches Makro mit Slot:

```
<p metal:define-macro="hallo">
  Hallo <b metal:define-slot="name">Welt</b>
</p>
```

Dieses Beispiel definiert ein Makro mit einem Slot namens `name`. Wenn man dieses Makro verwendet kann man das `b`-Element anpassen indem man den Slot `name` füllt.

Siehe auch

metal:fill-slot

fill-slot: ein Makro anpassen

Syntax

`metal:fill-slot`-Syntax:

```
argument ::= Name
```

Beschreibung

Die Anweisung `metal:fill-slot` passt ein Makro an, indem sie einen *Slot* im Makro mit dem Anweisungselement (und dessen Inhalt) ersetzt.

Die Anweisung `metal:fill-slot` muss innerhalb einer `metal:define-macro`-Anweisung verwendet werden.

Slot-Namen müssen innerhalb eines Makros eindeutig sein.

Wenn der angegebene Slot im Makro nicht exisitert, wird der Slot-Inhalt ohne Fehlermeldung ignoriert.

Beispiele

Gegeben sei dieses Makro:

```
<p metal:define-macro="hallo">
  Hallo <b metal:define-slot="name">Welt</b>
</p>
```

Man kann den Slot `name` folgendermaßen füllen:

```
<p metal:use-macro="container/haupt.html/macros/hallo">
  Hallo <b metal:fill-slot="name">Kevin Bacon</b>
</p>
```

Siehe auch

metal:define-slot

use-macro: ein Makro verwenden

Syntax

`metal:use-macro`-Syntax:

```
argument ::= ausdruck
```

Beschreibung

Die Anweisung `metal:use-macro` ersetzt das Anweisungselement mit einem Makro. Der Anweisungsausdruck beschriebt eine Makrodefinition.

In Zope wird der Ausdruck im Allgemeinen ein Path-Ausdruck sein, der auf ein Makro verweist, das in einer anderen Vorlage definiert wurde. Für weitere Informationen sehen Sie unter "metal:define-macro" nach.

Der Erweiterungsvorgang bei einem Makros ist das Aufsetzen eines Unterbaums von einem anderen Dokument (oder von anderswo im aktuellen Dokument) an die Stelle des Anweisungselements und dem gleichzeitigen Ersetzen des vorhandenen Unterbaums. Teile des ursprünglichen Unterbaums können erhalten bleiben, aufgesetzt auf den neuen Unterbaum, wenn das Makro *Slots* besitzt. Für weitere Informationen sehen Sie unter "metal:define-slot" nach. Wenn der Makro-Körper wiederum Makros verwendet, werden diese zuerst erweitert.

Wenn ein Makro erweitert wird sein `metal:define-macro`-Attribut mit dem `metal:use-macro` des Anweisungselements ersetzt. Dies macht das Wurzelelement des erweiterten Malros zu einem gültigen `use-macro`-Anweisungselement.

Beispiele

Grundlegende Verwendung eines Makros:

```
<p metal:use-macro="container/andere.html/macros/kopf">
  kopf-Makro definiert in der Vorlage andere.html
</p>
```

Diese Beispiel bezieht sich auf das `kopf`-Makro, das in der Vorlage `andere.html` definiert wurde, die sich im selben Ordner wie die aktuelle Vorlage befindet. Wenn das Marko erweitert wird, wird das Element `p` un dessen Inhalt druch das Makro ersetzt. Hinweis: das `metal:use-macro`-Attribut bleibt beim Ersetzungselement erhalten.

Siehe auch

metal:define-macro

metal:fill-slot

Zopebuch: Inhaltsverzeichnis

Anhang D: Zope-Ressourcen

Zum Verfassungs-Zeitpunkt gibt es eine Vielzahl von Informationsquellen über Zope im Internet, jedoch sehr wenige gedruckte. Wir haben einige der wichtigsten Links gesammelt, die Sie verwenden können, um mehr über Zope erfahren.

Zope-Web-Sites

Zope.org ist die offizielle Zope-Web-Site. Es gibt Downloads, Dokumentationen, Neuigkeiten und viele Ressourcen der Zope-Gemeinschaft.

ZopeZen ist eine Site der Zope-Gemeinschaft, die Neuigkeiten bringt und ein Zope-Job-Board hat. Die Site wird von dem bekannten Zope-Gemeinschafts-Mitglied Andy McKay betrieben.

Zope Newbies ist ein Weblog, das Zope-Neuigkeiten und themenverwandte Information bietet. Zope Newbies ist eine der ältesten und besten Zope-Web-Sites. Jeff Shelton startete Zope Newbies, und die Site wird gegenwärtig von Lukas Tymowski weitegeführt.

Zope-Dokumentation

Zope.org bietet eine Menge an Dokumentation einschließlich offizieller Dokumentationsprojekte und Dokumentationsbeiträge der Zope-Gemeinschaft.

Das Zope Documentation Project ist eine von der Zope-Gemeinschaft geführte Dokumentations-Website. Es stellt Originaldokumentationen bereit und verweist zu anderen Dokumentationsquellen.

Der Zope Developer's Guide zeigt Ihnen wie man Zope-Produkte schreiben kann.

Mailing-Listen

Zope.org verwaltet eine Sammlung der zahlreichen Zope-Mailing-Listen.

Zope-Erweiterungen

Zope.org bietet eine riesige Sammlung an Zope-Erweiterungen von Drittherstellern, die "Produkte" genannt werden.

Zope Treasures ist (war?) eine große Sammlung von Zope-Produkten (und ist offensichtlich mit beehive den Weg alles irdischen gegangen).

Python-Information

Python.org hat eine Menge Informationen über Python, einschließlich eines Tutoriums und eines Referenz-Handbuchs.