Zope Content Manager's Guide

Document revision 2.1.1,
Modified March 20, 2000 by Pamela Crosby

For Zope version 2.1.0

Table of Contents

Introduction 5

Quick-Start Tutorial 7
- Design and Implement a Skeleton Site 7
  - Task 1: Creating Folders 8
- Creating Images and DTML Methods 10

Delegating Control 15
- Security 15
- Introduction 15
- Architecture 15
- Managing Users 17
- Authorization 18
- Local Roles 19
- Managing Roles 20
- Security Examples 21
  - Creating a DTML Method which is publicly viewable but editable only by Managers 21
  - Task 2: Delegating day-to-day management of a Folder 22

Zope Applications and Products 23
- Creating Zope Applications using DTML Scripts 23
- Creating a Configuration Interface 27
- Turning an Application into a Zope Product 33
  - Factories 35
- Creating Product Distributions 39
  - Tutorial Part One - Summary 41

Tutorial - Part Two 42

Working with Versions 43
- Versions 43
  - Creating a Version 43
  - Joining a Version 43
  - Working in a Version 44
  - Leaving a Version 44
- Folders 44
  - Creating Folders 45
  - Working with Folder contents 46
  - Setting and using Folder Properties 46
- Files 47
  - Creating Files 48
  - Uploading new File data 49
- Images 51
  - Creating Images 51
  - Editing Images 51

Setting and using Image Properties 52
Viewing Images 52
User Folders and Users 52
Managing security 53
Creating User Folders 53
Adding New Users 53
Managing Users 53
Editing Users 54
Deleting Users 54
When to Use DTML Method vs. DTML Document? 57
DTML Methods 57
Creating DTML Methods 57
Editing DTML Methods 58
DTML Documents 60
Creating DTML Documents 60
Editing DTML Documents 60
Viewing DTML Documents 60
Setting and using DTML Document Properties 61
Uploading new data 61
External Methods 63
Creating External Methods 63
Working with External Methods 63
Acquisition and External Methods 65
Importing Modules and Packages 65
External Method Limitations 66
Saving or discarding Version changes 67
Zope Management 68
Control Panel 68
Database Management 68
Cache Management 68
Cache Parameters 69
Flush Cache 69
Quick Start Tutorial Summary 72
Index 73

Introduction

Welcome to Zope, the premier dynamic platform for servicing content managers.

Zope is a powerful system for dynamic content delivery. It provides:

Web to objects.	Zope provides a unique architecture for mapping URLs to object messages.
Integrated object database.	The heart of Zope is its integrated object database. This powerful object store provides seamless access through URLs, transactional updating, and automatic versioning.
Rich HTML.	Our documents add rich programming to HTML, including iteration, conditional testing, variable insertion, HTTP headers, cookies, and more. Since these rich HTML templates are objects, your users can undo mistakes.
Powerful User Folders.	Users and passwords can be added to any URL level.
Sophisticated but simple sharing.	The Zope object system implements a breakthrough in object orientation called acquisition. This lets you share information along a URL, from higher up an organization to down below.
Built-in objects.	Zope automatically provides useful objects for website builders. These include DMTL Methods, DTML Documents, Images, Folders, UserFolders, and MailHost objects. Additionally, there is a Control Panel that allows application and database management.
Application integration.	Zope provides tight integration with other products in the Zope line.
Cross platform.	The Zope line has great reach and high performance across a number of operating systems and Web servers.

The purpose of this document is to aid the content manager in utilizing the Zope application. Since learning is facilitated by action, this guide provides a tutorial format for the content manager to become familiar with Zope web application.

The guide starts with a tutorial to show the dynamics of the Zope product. The user is guided through design and implementation of a web site. The tasks of creating folders, incorporating images (company's logo), and incorporating Document Template Markup Language (DTML) methods into web pages are explored.

This guide addresses the aspects of security, managing roles and managing users. Step by step screen descriptions show the content manager how to allow access to various users and assign them security roles. Manager roles allow the content manager to assign trusted staff members the responsibility to managing a Zope website or a portion of the website, while the anonymous role lets the custormers of these trusted site managers have viewing access to the web site.

This guide provides the content manager with building blocks to create powerful web applications.

Quick-Start Tutorial

The scenario in this tutorial involves the creation of a website for ***ZAcme, Incorporated***. Stan has been appointed the task of creating the public website for **ZAcme, Incorporated** to give the public online information on its products, research, and news. Stan is also busy as head of the Products department and does not want to presume anything about the Research and Public Relations (PR) departments, so he needs to delegate responsibility for content to the departments themselves. Yet, he also needs to make sure that the site looks and behaves similarly at all points, and some common, consistent information needs to be displayed on all pages. As the manager responsible for public information online, Stan needs to be able to make changes at any location.

Each department is responsible for its own content. Each department has varied groups and committees overseeing different areas of that department. Finally, each department requires individual control over who can be assigned privileges to add and manage contents in their subareas.

Stan has already installed Zope. He created a folder named *ZAcme* under his root installation. The public interface and users folders options will not be used at present. (Make sure you turn off the "Create Public Interface" and "User folder" options when you create this sample folder.)

Stan will create the basic elements that the site will be built on: documents, images, and folders. Since a skeleton site needs to be seen by others quickly, Stan gives only enough to show what will be coming.

Right away, Stan organizes some of the basic content that all of the sites will be able to use. He also gives access to the PR and Research departments for their respective folders. So follow along as we do the following with Stan:

Design and Implement a Skeleton Site

In this stage, Stan needs to do the following:

- Create *Folders* for each of the three departments.
- Give those *Folders* some sort of public interface for people to see, and a place to store future information about who has access to the *Folders*' contents.
- Create a home page with a logo for the company.
- Give information in that page about the date the full site is expected to come online.

It may sound complicated, but with Zope objects it's simple.

With the installation of the Zope, you will already have QuickStart Folder and the standard index_html, standard_header_html, standard_footer_html. The index_html (Welcome to Zope) is available as a sample.

1. For this tutorial to work best, create a folder named ZAcme under your root directory.
2. In the *Available Objects*, select *Folder.*
3. In the *Add Folder* screen, place `ZAcme` in the *ID.*
4. Select the ❑ **Create public interface** and ❑ **Create user folder** options to turn them off.
5. Press the *Add* button.

Task 1: Creating Folders

Stan’s first task is to create Folders for the three departments within the ZAcme Folder, along with a public view of those *Folders*. Zope joins these two tasks together when creating *Folders*.

The first *Folder* Stan creates, he decides to call *News*, the *Folder* in which the PR department will be responsible primarily for updating. To add the new *Folder* Stan follows these steps:

1. Stan goes to the */manage* URL in the Web browser. The Zope *management screen* appears as shown in Figure 1. The

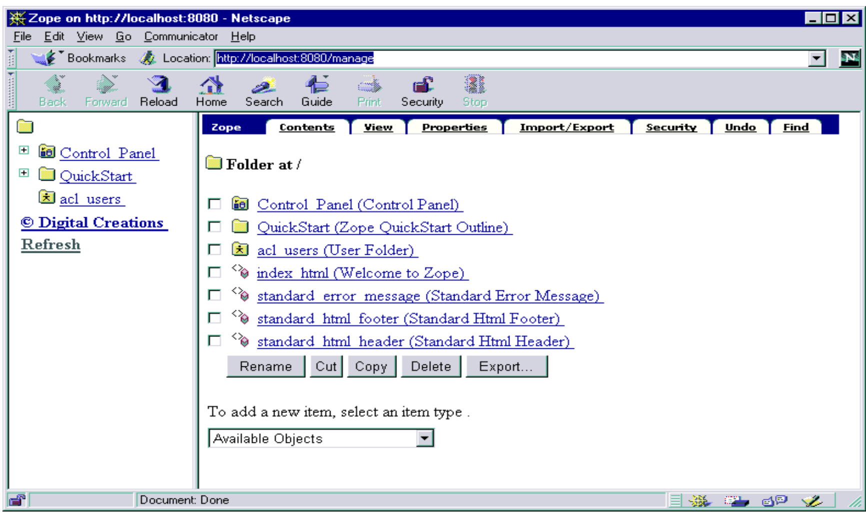

Figure 1. Zope management screen and ZAcme Folder Contents view

Zope management screen is a two-frame interface. The frame on the left, known as the *navigation* frame, shows all the *Folders* and subfolders or any folder-like object. It allows easy movement between different areas of a Zope site. The frame on the right is known as the *workspace*. This is where work on individual objects is done. Screens inside the workspace are known as *workspace views*.

The grey boxes at the top of the workspace are the *object tabs*. Object tabs are used to switch between the different views. The tab of the current view selected is a lighter color than the other object tabs. See "Object Reference" for an explanation of the different views for each object.

The main view used in Zope is a Folder’s *Contents view* (Figure 1). The list of items with checkboxes in front of them are the *Folder items*. These are the objects in the current Folder. The checkboxes are used to select objects for *Folder actions*. The buttons just below the Folder items are the Folder actions. These buttons operate on items whose checkboxes are selected. The drop-down list lets you add a new object to the current Folder.

2. By selecting “Folder” from the "Available Object", an “Add Folder” form appears in Stan’s workspace as shown in Figure 2.

3. Stan provides the Folder information as follows:

- He sets “Id” to `News`. The id appears in the URL for the new folder.
- He sets “Title” to `The ZAcme News`

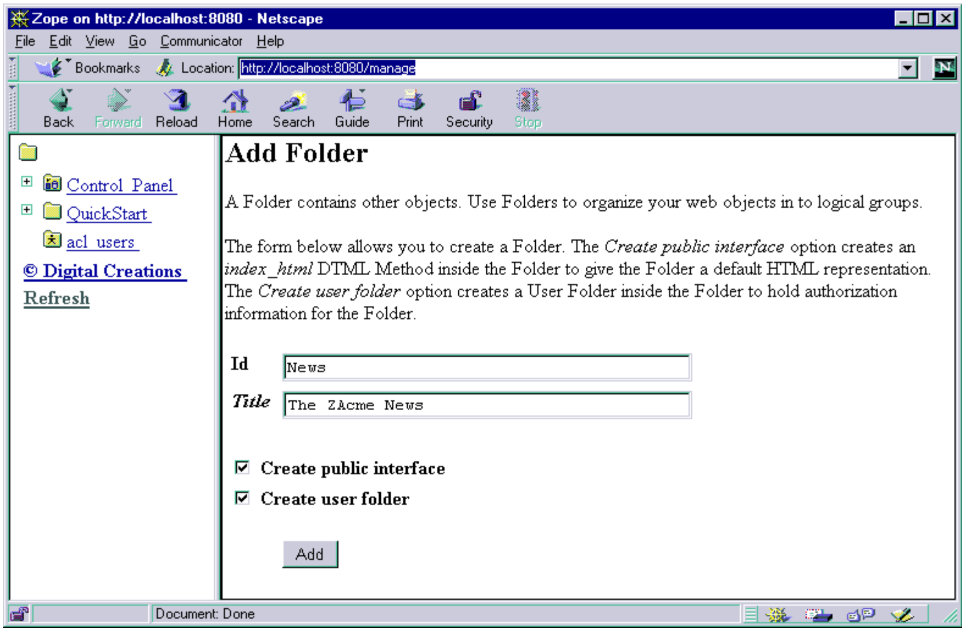

Figure 2. Add Folder

He makes sure the “Create public interface” and “Create user folder” options are selected. With these selected, the "Create public interface" will create a new DTML Document with the id, `index_html`, and a User Folder created in the under the new Folder. The *index_html* document is displayed when the Folder object is called via a URL. This is what is also known as the Folder’s *Public Interface*. Those familiar with traditional Web publishing will recognize this as the same behavior as an *index.html* or *default.htm* document. The "Create user folder" option will create the User Folder with *acl_users* under this added folder.

4. Stan then clicks “Add” to create the Folder. The new Folder is added to the Folder Items list and appears in Stan’s navigation frame.
5. Stan follows the same steps to create the products folder with the id *Products* and the title *The ZAcme Product Line* and the research folder with the id *Research* and the title *The ZAcme’s Research Unit.*

The first part of his task is completed! His */manage* screen should now look like Figure 3.

Notice the “+” marks next to the new folders in the navigation frame. These marks indicate that the corresponding folders are expandable. Remember that Stan chose the option to create User Folders when adding the new Folders. Since User Folders are a special type of folder-like object, they too will appear in the navigation frame when the department Folders are expanded.

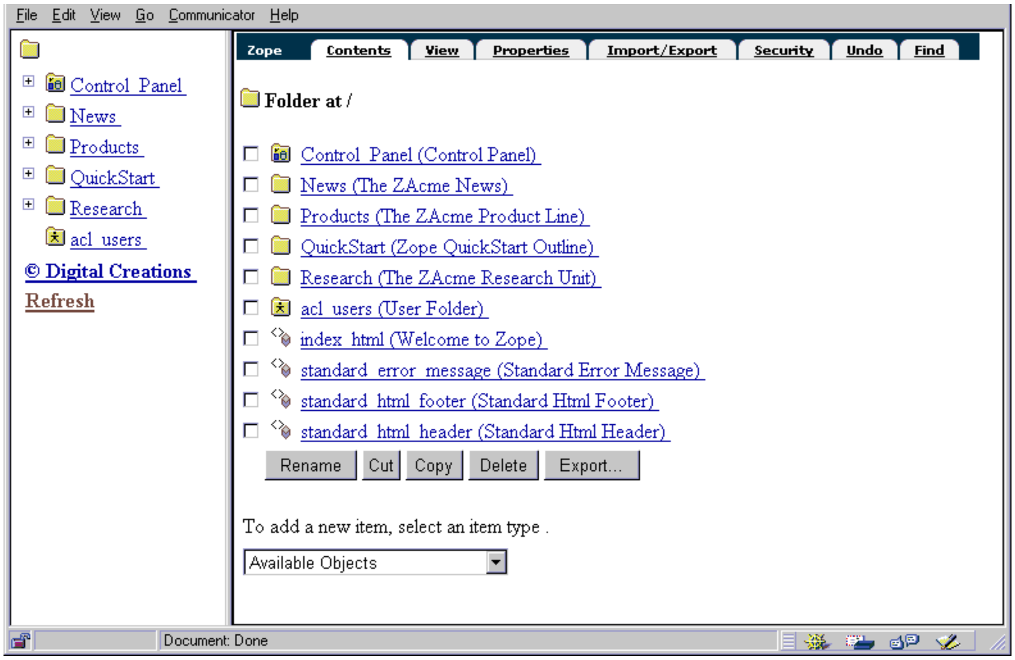

Figure 3. Manage screen after adding department Folders

Creating Images and DTML Methods

Now that Stan has completed the first part of this task, he has to create a simple company Web page. This page will link to the newly created Folders, will contain a company logo and will indicate when the full site is expected to come online. First, Stan needs to find a company logo graphic to use in the home page. With some help from the graphics department, he finds a disk containing the logo. Now, all Stan has to do is get the logo into Zope, in particular into the ZAcme Folder. Stan creates an *Image* object in the ZAcme Folder by completing the following steps:

1. Selecting “Image” from the Add List , Stan is presented with the workspace as shown in Figure 4.
2. Stan sets the “Id” to `ZAcmeLogo` and the “Title” to `The ZAcme, Inc. Logo`. He clicks the “Browse...” button to select the logo file from his disk. Since the logo is a GIF file, the height and width of the image are determined automatically.
3. Finally, he clicks “Add” to create the Image.

Stan’s next task is to create a home page. Since the ZAcme Folder was initialized during the Zope installation, it does not contain a public interface, or *index_html* DTML Method. Thus, he needs to add a *DTML Method* to the ZAcme Folder. *DTML Methods* are how information in Zope objects is presented to users accessing Zope as a normal website. By taking on the properties of the objects they are contained in, *DTML Methods* can be used to manipulate and display the objects’ data . To create the *DTML Method*, Stan does the following steps:

1. Selecting “DTML Method” from the "Available Objects", Stan is presented with the *Add DTML Method* form as shown in Figure 5.
2. Stan sets the “Id” to `index_html` and sets the “Title” to `Welcome to ZAcme, Incorporated`.
3. Stan could upload an HTML file from his computer at this point by clicking the “Browse...” button, but he wants to type in the DTML Method contents within Zope. The file field remains empty.

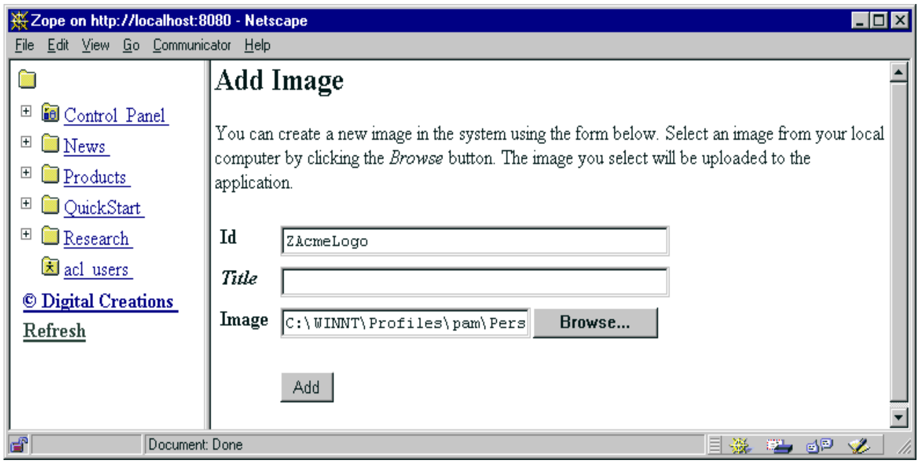

Figure 4. Adding Images

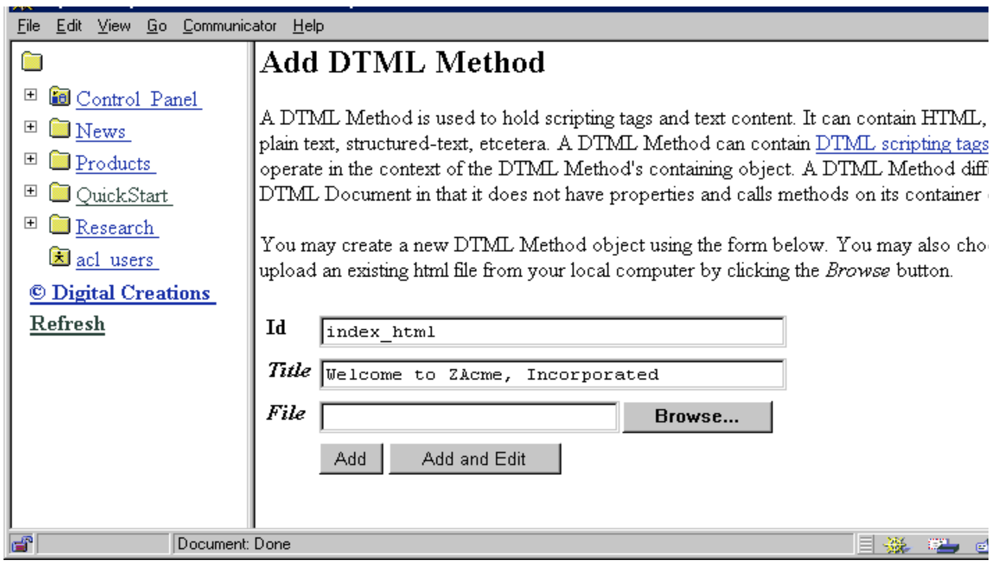

Figure 5. Adding DTML Methods

4. He clicks "Add" to create the DTML Method.
5. In the Contents view of the folder, Stan opens the DTML Method for editing by clicking on its id in the Folder Items list. The screen which then appears is shown in Figure 6.

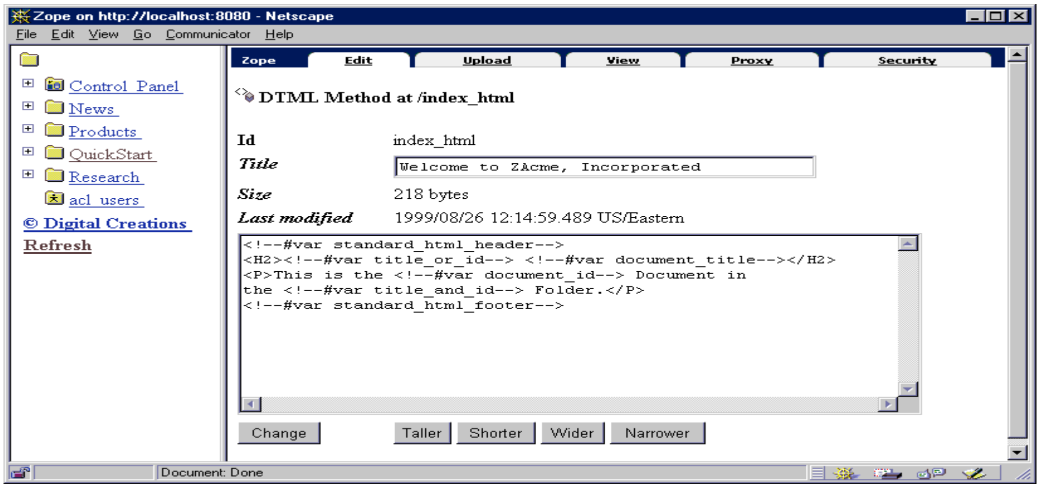

Figure 6. Managing a DTML Method

6. He deletes the text that is automatically inserted by Zope and enters in the following:

```
<HTML>
<HEAD>
<TITLE><dtml-var document_title></TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<P><dtml-var ZAcmeLogo></P>
<H4>Welcome to ZAcme Web, the official web site of ZAcme Incorporated</H4>
<P>This site is currently under construction. Be sure to visit us soon when the
full site is ready.  We expect to have information online concerning:
<UL TYPE=SQUARE>
<LI><A HREF="News">Company news and press releases</A>
<LI><A HREF="Products">Company product information</A>
<LI><A HREF="Research">What we have brewing in our research labs</A>
</UL>
</BODY>
</HTML>
```

Here we have a simple HTML page with Zope *Document Template Markup Language* (*DTML*) instructions interspersed with normal HTML text. We use two **DTML** instructions here:

- `<dtml-var document_title>`:

  `dtml-var` is a DTML tag used to insert a variable. The variable here is *document_title*, which is the title specified for the DTML Method. Since a DTML Method is used to display the information of other objects, its information needs to be distinguished from the information of the objects it's contained in. If the variable was *title*, the title of the folder containing the DTML Method would be inserted.

- `<dtml-var ZAcmeLogo>`:

Here the variable is the Image object which Stan created. Remember that objects are always referred to in Zope by their ids. An alternative way in which Stan could have placed this image is by using traditional HTML, `<IMG SRC="ZAcmeLogo" ALT="The ZAcme, Inc. Logo">`. In fact, this is what `<dtml-var ZAcmeLogo>` generates.

7. After modifying the source code for the DMTL Method, Stan clicks the "Change" button. A message appears at the top of the editor screen signifying the changes. He clicks on the View tab and sees the screen shown in Figure 7.

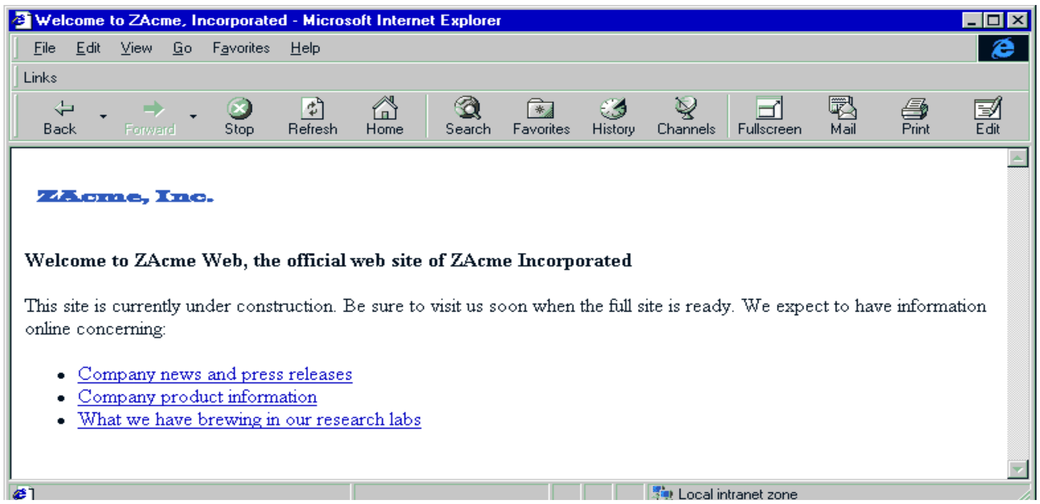

Figure 7. Viewing a DTML Method

Tip: Many browsers feature "pop-up" menus, accessed by right-clicking (Windows, OS/2), Control-Clicking (MacOS). When a pop-up menu is brought up on a hyperlink, there may be an option to "Open link in new window." Choosing this option on the View tab opens the rendered DTML Method in a separate window instead of the normal workspace frame. Here the DTML Method can be seen as others will see it.

Mission accomplished!

Except for one thing, Stan needs to put a date in the home page telling when the full site is expected to come online. He could do this manually by entering a date into the home page DTML Method. However, if the date was entered manually in other pages, then a change in the expected date would require a great deal of editing. Fortunately, by using DTML and Folder properties, the expected date can edited in all the pages by editing only one date. Stan takes the following steps:

1. He goes back to the main folder by clicking on it in the navigation frame, and clicks on the Properties tab in the workspace. The Properties screen appears as shown in Figure 8.

The top portion of the properties screen is where existing properties can be edited or deleted.

The bottom portion is where new properties can be added by selecting their type, giving them an id and a value.

Stan needs to add a *Date* property. He goes to the add property portion of the screen and gives this property the id, *expected_date*. He then selects the type "date", types in a value of `January 27, 2000` and clicks "Add". The screen refreshes with a new property named *expected_date*. Note that the date appears in the format `2000/01/27`. This is because it is a Zope *date-time* value and Stan chose to have the property be of type date instead of string. Zope date-time values can be formatted and calculated according to the rules of dates and times. Now, whenever the expected date needs to be changed within the home page, Stan can easily make the change here.

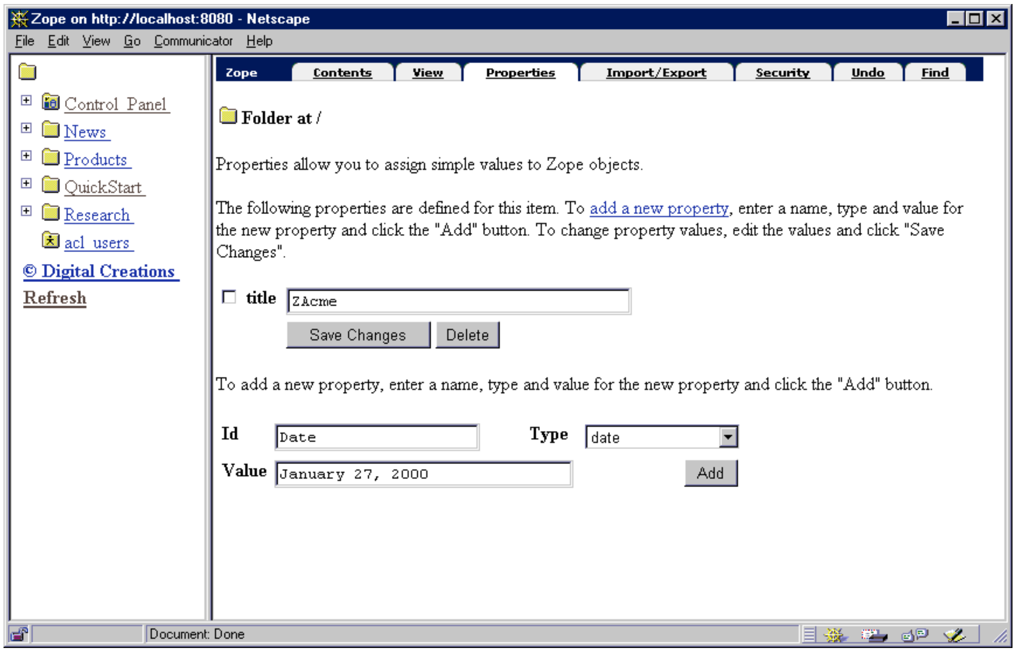

Figure 8. Properties screen

2. To put *expected_date* in the home page, Stan returns to the Contents view of the Folder and opens the *index_html* DTML Method by clicking on its id in the Folder Items list. He adds the following paragraph to the DTML Method after the unordered list (UL).

```
<P>The full site is expected to come online on
<dtml-var expected_date></P>
```

Since *expected_date* is a property of the Folder containing the home page, it can be accessed in a *dtml-var* tag. In Zope, *expected_date* can also be accessed in a *dtml-var* tag within DTML Methods in sub-folders. This technique, known as *acquisition*, is one of the unique features of Zope. Acquisition easily allows information sharing across a website.

Delegating Control

Security

Administering the security of a website is an important part of a content manager’s job. In fact, as the site becomes more sophisticated and more interactive, security becomes crucial.

Unfortunately, most Web platforms make administering security difficult. First, HTML and graphic files might have their security tied to the operating system, but their user database is off-limits for the content manager. Secondly, the commands used to modify security are quite cryptic and thus too complex for most content managers. Finally, data that is dynamically delivered, for instance by CGI scripts and database logins, introduce a host of new headaches concerning users and permissions.

Site managers also have a hard time being responsive. How much control will be given to the content manager? Will giving the content manager a login unacceptably lower security? What policy is followed when the content manager wants a login for his customers? How much work will be involved in deleting the customer and all the tendrils that have spread throughout the platform?

Zope security was designed to address these content and site manager’s issues. Additionally, the security model leverages the unique features of Zope to mirror unique Web business models.

Introduction

The Zope security model offers a flexible architecture designed to complement the platform’s object model. Some of the security model’s design goals are as follows:

- Mimic the business idea of customers who have customers
- Reinforce Zope ideas of containment, acquisition, and dynamism
- Allow new objects to extend the security model at any level in a URL
- Separate authentication and authorization
- Project an intuitive GUI that balances power and simplicity

Architecture

The security architecture concerns itself with two basic ideas: who are you (authentication) and are you allowed to do this operation (authorization). *Authentication* involves determining who the user is based on an HTTP authentication protocol and what types of things that user is allowed to do. Authorization, on the other hand, is concerned with the operation on the object. Namely, what privileges are needed by this user to perform this operation? Managing Zope security involves managing both information about users and permissions on objects.

At first it would appear straightforward. Just keep a list of users and a list of things they can do. This model, however, would quickly break down as the number of object operations and users grew. Thus, an abstraction is inserted between users and object operations. This abstraction, discussed below, allows users and operations to be generalized.

A fundamental idea in Zope security is that administration should be turned over to others as you traverse the folders in a URL. The administrators at each level can define new administrators below their folder, thus passing the work down the hierarchy. To do this effectively, one must understand the four key components of Zope security: *users*, *roles*, *permissions, and acquisitio*n.

Users	People (or groups of people) who interact with Zope are represented by Zope “users”, or “user objects.” Zope user objects provide management of authentication information and the kind of access people have.
Roles	Roles represent kinds of responsibility and authorization, such as “Manager” or “Author.” Roles provide the linkage between authentication and authorization. They are functionally similar to “groups” in other security systems.
Permissions	Permissions represent like operations on objects and provide an organized mechanism for setting access control on objects. Permissions correspond to the high-level permissions, like “read,” “write,” and “execute,” found in file systems. However, permissions are specific to objects. Different types of objects can provide different, object-specific permissions.
Acquisition	Acquisition is the mechanism in Zope for sharing information among objects contained in a folder and its subfolders. The Zope security system uses acquisition to share permission settings, so that access can be controlled from high-level folders.

These concepts will be discussed in more detail in the sections that follow.

Managing Users

People, whether internal staff members or customers, are represented in a Zope site by user objects. These user objects are managed in special folders called *User Folders*.

For the next task in Stan's assignment, let's create a user object, *Sally*, under the News folder. In the *navigation* frame, select the *News* object to retrieve the subfolder available. To add Sally, click the *acl_users* item under the News folder. Figure 10 shows the Contents view of the User Folder in the *News* subfolder is displayed. Enter a password and confirm it by entering twice. Make sure you highlight the "Manager" field in the *Roles* display and then press *Add*. Where a user is defined is important, as it will define the scope in which their privilege applies. Because *Sally* is defined in the *News* Folder's user folder, she has manager access only to objects in the *News* Folder and subfolders.

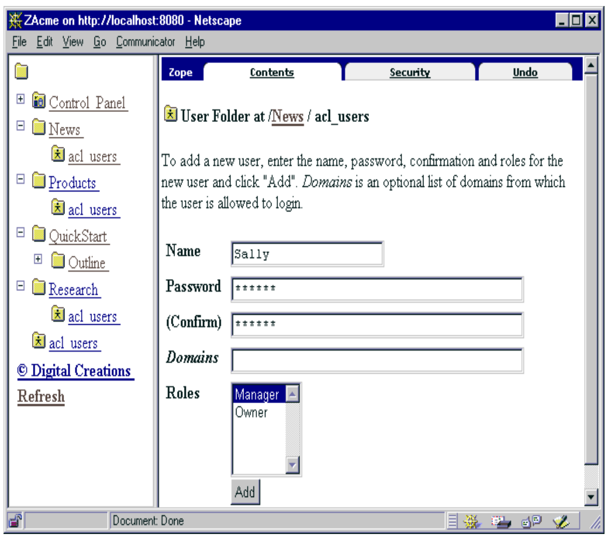

Figure 9. User Editing view

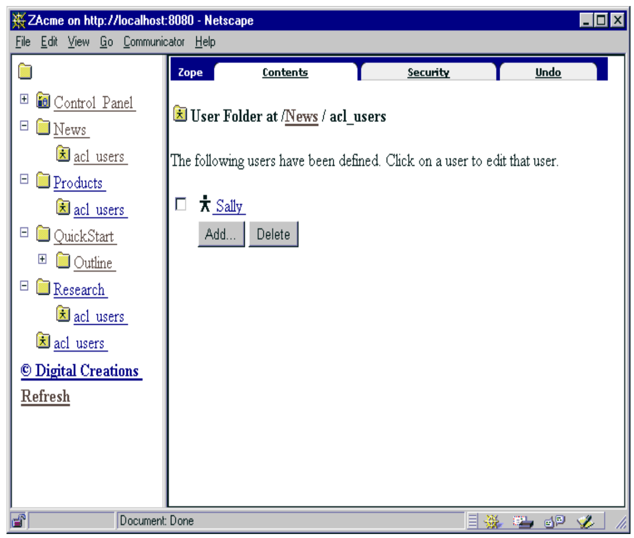

Figure 10. User Folder Contents view

User Folders can be defined in any folder. For example, the navigation frame in Figure 9 shows User Folders in each of the site's four Folders. The ability to put a User Folder in any folder provides support for highly decentralized user management. It is also possible to create custom User Folders that can be used to interface with external user management systems, such as directory servers.

User objects contain the information needed to identify and authenticate users, and to tie them into the authorization system. Users are identified by a name. DTML Methods can refer to the current user using the variable, *AUTHENTICATED_USER*. When this variable is inserted in a DTML Method, the user's name is inserted.

Users also have authentication information. In standard Zope User Folders, the authentication information consists of a user's password and/or an internal domain name. Other user folders may use different authentication information.

Users are tied into the authorization system by their assignment of roles. Roles describe the kind of responsibility and authorization possessed by a user.

Authorization

Figure 11 shows a Security view for a DTML Method. To get to this screen, access the *html_index* in the *Content* frame. Select the *Security* object tab. The Security view is a standard object view for controlling access to objects, or authorization. As the figure shows, authorization is defined in terms of which roles are assigned to which permissions. In addition to specifying which roles have each permission, each permission has an option for acquiring permission settings from containing folders.

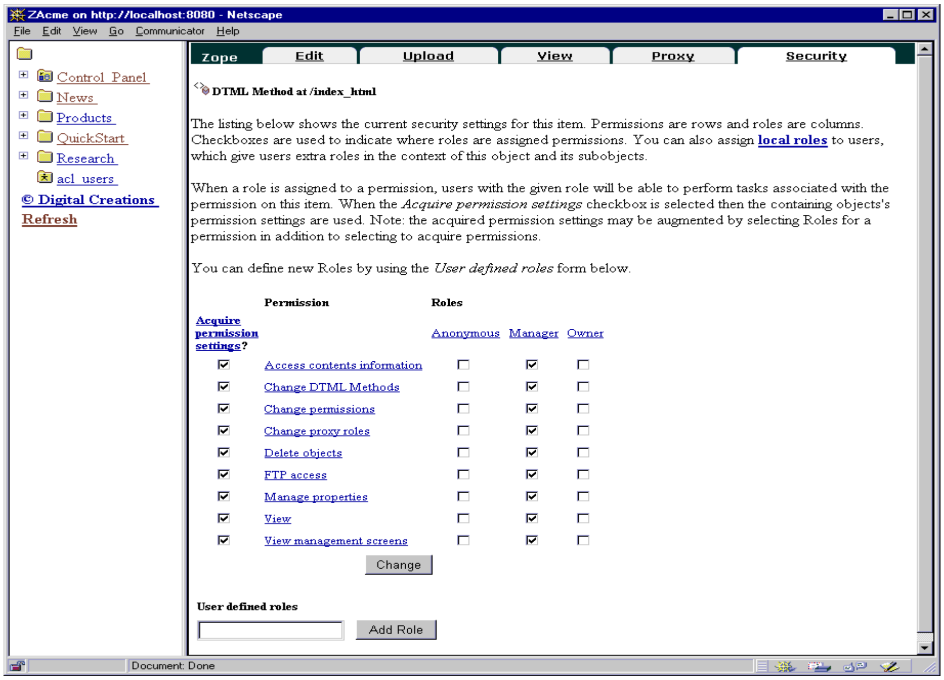

Figure 11. Default permission settings for a DTML Method

The settings shown in Figure 11 are typical for DTML Method permission. All permissions acquire permission settings and no roles are assigned to permissions explicitly. Access to objects with permission settings like those shown in figure 11 can be controlled entirely from folders containing the object.

Figure 12 shows the security view for a top-level folder. To retrieve this window, click on the main folder in the navigation window. This will return you to the top level folder list. Press *Security* to view the permissions set. Unlike other security views, the security view of a top-level folders does not contain a column of checkboxes to control acquisition of permission settings, since there are no higher-level folders to acquire permission settings from.

The top-level folder is where Zope’s default permission settings are defined. Access control for an entire Zope installation can be controlled from the top-level folder. By default, only the “Manager” role is able to perform most operations. The “Anonymous” role, which all users have implicitly, has “View” and similar innocuous permissions.

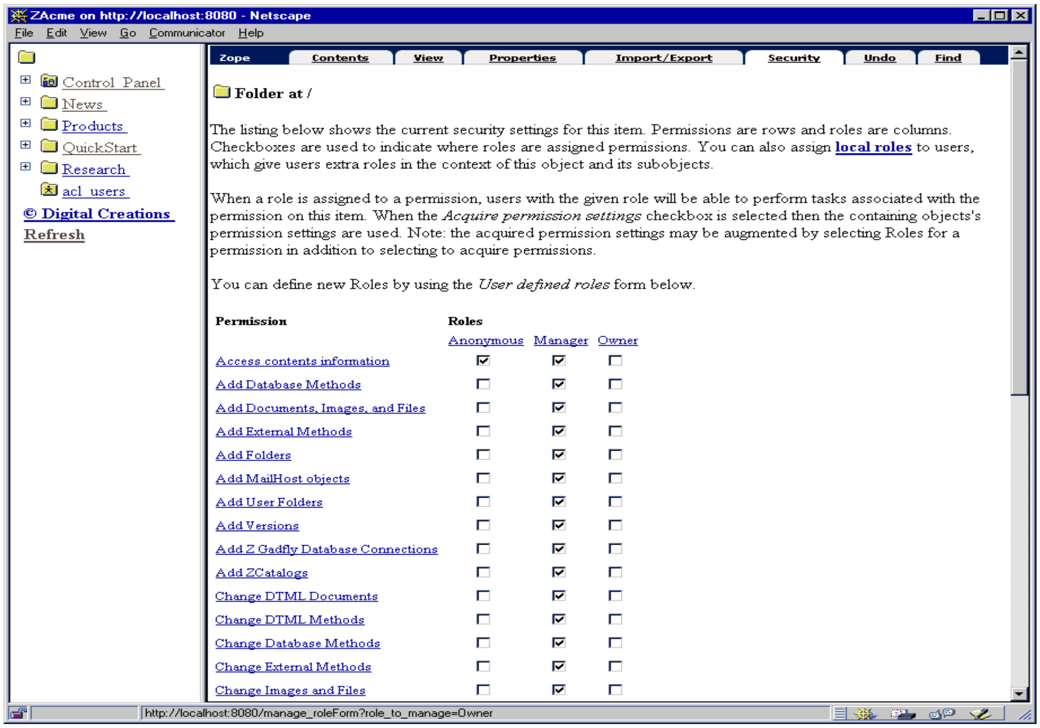

Figure 12. Security view of top-level folder

Different types of objects in Zope define different kinds of permissions. For instance, all Zope objects have a permission called *View Management Screens*. However, while Folder objects have a permission called *Add objects*, DTML Method objects do not since they cannot contain other objects. Similarly, an object with specific functionality like a Confera Topic might have a permission called “Add messages”. Thus, the permissions available on an object depend on the capabilities of that object. Note from Figure 12, that security views for folders show many permissions. Folder security views show all permissions for operations on all objects, not just those for folder operations. This is necessary so that permission settings may be defined centrally and acquired by contained objects.

Local Roles

Local roles allow users to be bound to roles at the level of an individual object. Local roles are not acquired.

A common use of local roles is to identify the owner of an object. Zope enables this arrangement by automatically assigning local roles when objects are created. Zope creates a local role of Owner for the user who creates an object. The upshot is that the user who creates an object is given authorization to manage that object.

Local roles are not limited to controlling ownership. You can assign any user any role on a given object. However, it is probably a good idea to restrict one’s use of local roles to exceptional cases, since it is much more time consuming to assign local roles for objects than it is to assign roles to folders and allow contained objects to acquire those role settings.

To edit local roles for an object go the "Security" management screen and click on the local roles link. You will then be presented with the local roles management screen.

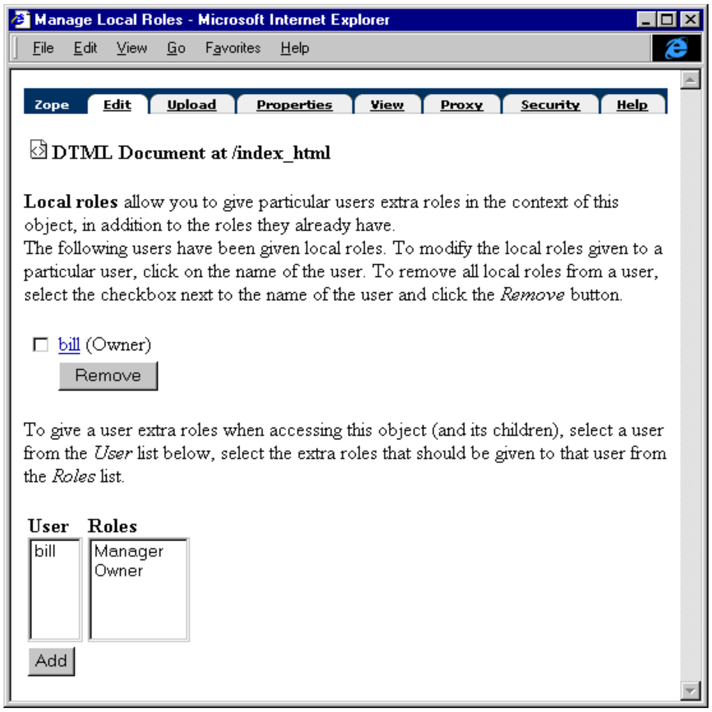

Figure 13. Managing local roles

This screen shows you which users are assigned local roles, and allows you to delete them. It also shows you the currently defined users and roles, and allows you to set a local roles by assigning a role to a user.

Managing Roles

Administrators can use the Security tab to create new roles and apply them to security settings on objects. When a role is added at a particular point in an object hierarchy, it is made available for use at that point and below. Zope has pre-defined roles that have either a special meaning or a conventional usage. These roles are:

- *Anonymous* -- All users, including users that do not supply authentication credentials. Giving a permission to the Anonymous role makes the corresponding operations publicly accessible.
- *Manager* -- By convention, the Manager role is assigned to users with responsibility for managing a Zope website or portions of a website. By default, all permissions associated with modifying Zope objects are given to the Manger role.
- *Owner* -- When an object is created, the user who created it is given a local role of Owner. By default, the Owner role has the same permissions as the Manager role.

Security Examples

The following examples demonstrate the use of Zope security to accommodate some common access control requirements.

Creating a DTML Method which is publicly viewable but editable only by Managers

A basic public website built with Zope is made up of a number of documents which should be viewable by the general public, also known as anonymous users. Anonymous users should not, however, be able to modify these documents or even view their management interfaces. The default security settings of new documents ensure this by using acquired permissions, but it is useful to demonstrate how this can be done explicitly.

First, create a DTML Method. Next, click on the newly-created DTML Method to visit its management screen, then click on the Security tab to view the current security settings, as shown in Figure 11. By default, for most objects, there are no explicit roles set for any permissions. All permission settings are acquired.

To specifically make the DTML Method viewable by anybody and editable only by managers, unselect all the acquired permission settings and explicitly set roles shown in Figure 14 then select “Change”. Note that it does not matter whether

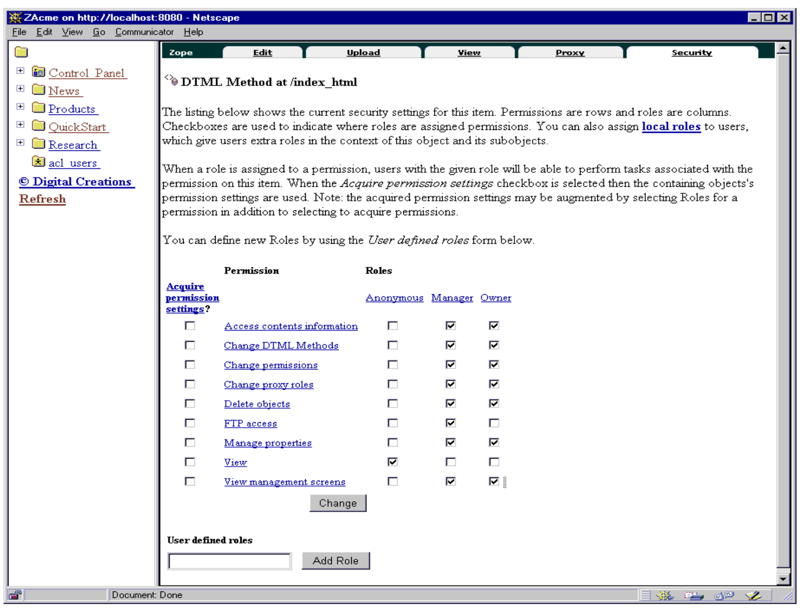

Figure 14. DTML Method Security view after explicitly setting security

or not the Manager role has the *View* permission, since all users implicitly have the Anonymous role. Unselecting the acquired permissions settings prevents additional roles from being assigned to management operations through acquisition. Disabling acquisition of permission settings eliminates the ability to manage security for the object centrally and is generally undesirable.   However, when specific access is absolutely required, then disabling acquisition does provide a finer level of control. Additionally, explicitly setting the desired roles assures that the desired roles have the permissions, regardless of settings in higher-level folders.

Task 2: Delegating day-to-day management of a Folder

It is often useful to delegate the day-to-day management of a Zope Folder to another person without giving that person complete control over that Folder. The fine-grained permissions defined by Zope objects make it easy to delegate as much or as little control as your situation requires.

Our company has a Product, Public Relation (News) and Research department, and that Stan are responsible for our company intranet site. It would be inefficient for each department to have to go through us to make changes to their areas, so Stan will delegate most (but not all) Folder management abilities to a selected user in each department.

To do this, Stan create a new role, “Admin”, in the top-level folder, which will be usable by all subobjects contained in the site.

- First, Stan views the top-level folder by clicking on its entry in the navigation frame and then click on the Security tab to display the Security view. At the bottom of the Security view is a small form to add a role.
- Stan use this form to add the Admin role. After entering the new role name and clicking the “Add Role” button, the Security screen is updated to include a new column for the Admin role and to provide an additional form for deleting user-defined roles.

Finally, Stan adds permissions for the role Admin.   Stan wants the department caretaker to be able to do all of the things a manager does, except change the security setting of the of other folders. To do this, Stan gives the Admin role all permissions except “Change permissions” and “Manage users.”

Next, Stan will go into the *Research* Folder, and create a new user, *Phil*, in its User Folder. This will be the user who is going to manage the day-to-day activities of the Folder. Stan will assign this user the “Admin” role as shown in Figure 15.

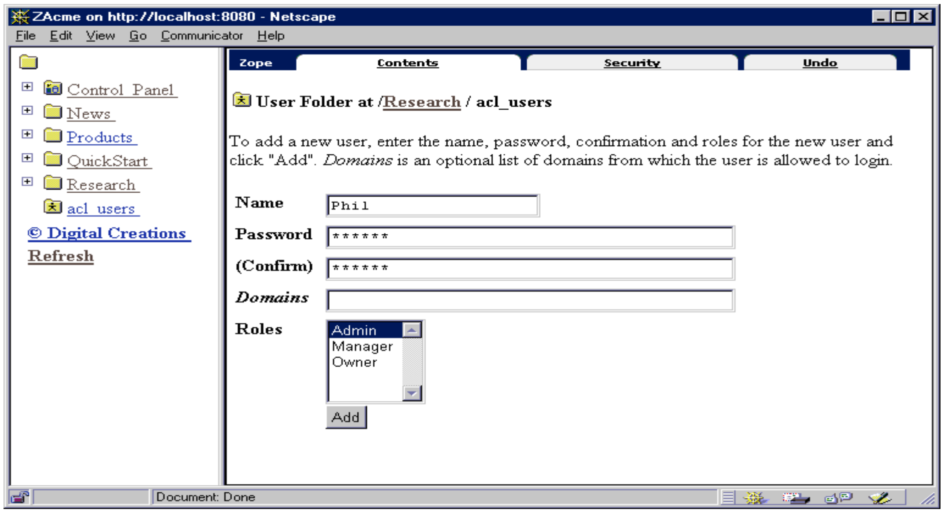

Figure 15. Adding an Admin user

By repeating this process for the Public Relations and Products Folders, each department can now take care of its own section of the intranet.

Zope Applications and Products

There are currently two ways to create products for Zope. Products can be written as Python[1] packages containing Python[2] modules, or they can be created using Zope's management screens and Document Template Markup Language (DTML) Scripts. This chapter describes how applications built using Zope management screens and DTML scripts can be turned into Zope products that can be easily reused in other Zope installations.

Creating Zope Applications using DTML Scripts

A Zope application is simply a collection of Zope objects that solves a particular problem.

To illustrate the use of Zope applications, we will use a very simple *website application*. The simple website application, in this example, consists of a folder (Figure 16) containing an *index_html* DTML Method (Figure 17) and a logo. The source code of the *index_html* DTML Method uses several properties that are defined in the folder (Figure 18). When the Home folder is visited by a browser, the *index_html* DTML Method is rendered to display the home page shown in Figure 19.

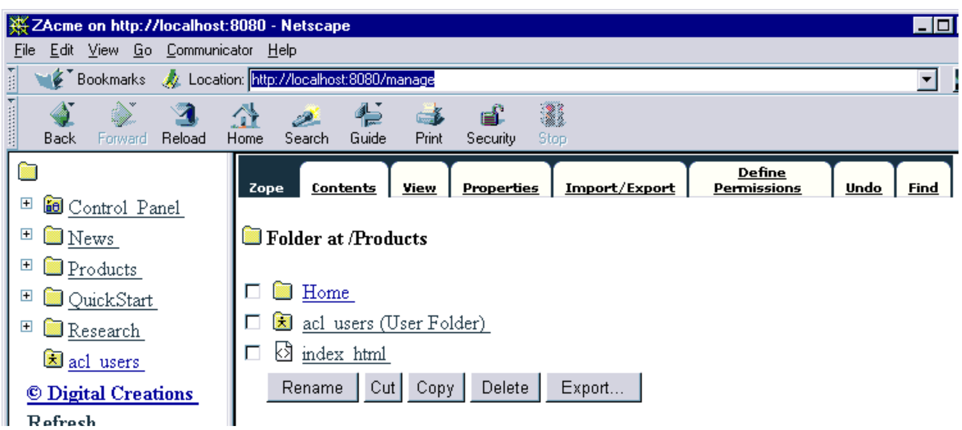

Figure 16. Simple website application folder

1. Python is a high-level object-oriented programming language. To find out more about Python, visit *http://www.python.org*, or look at one of the many books on the language.
2. Modules can also be written in C or in programming languages callable from C, such as C++ or Fortran. Modules can also use distributed protocols, such as CORBA, ILU, or COM.

Create a folder under Products with the "id" `Home` and "title" `Products Home Page.` Add a `logo` for the Products department. Go to the *index_html* edit page shown in figure 17. Change the information in the html document

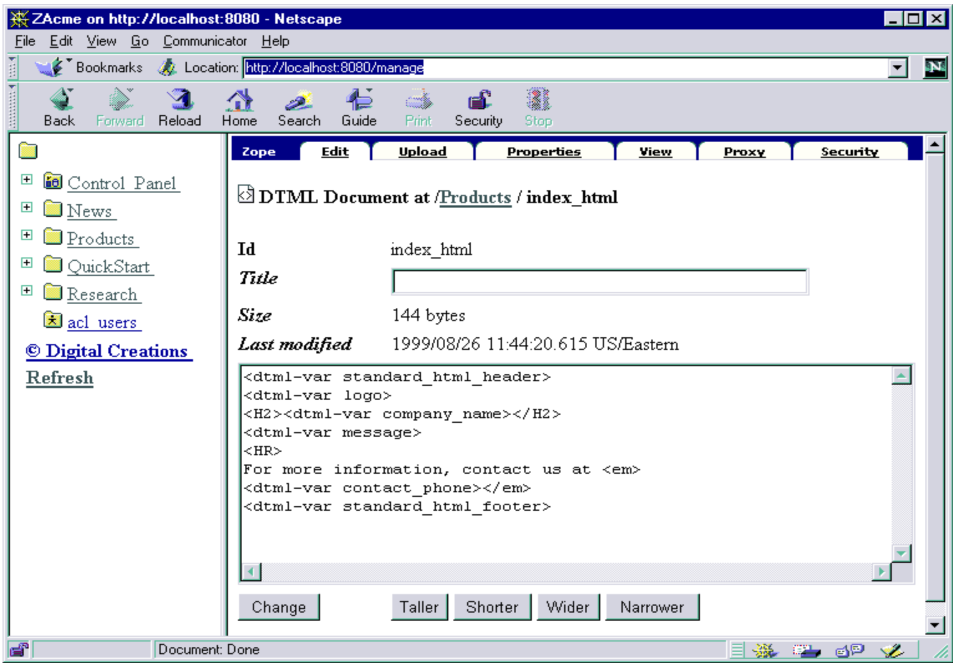

Figure 17. index_html DTML Method

with the follow:

```
<dtml-var standard_html_header>
<dtml-var logo>
<H2><dtml-var company_name></H2>
<dtml-var message>
<HR>
For more information, contact us at <em>
<dtml-var contact_phone></em>
<dtml-var standard_html_footer>
```

Now we have a simple web page set up for the research department, but the *dtml-var* objects have not been defined. In figure 18, the properties can be added to the **Home** folder. Click on the *properties* object tag for **Home.** As a DTML Method, it will access the properties from the folder it resides or higher up the hierarchy. To add a property, enter the "id" `message`, type in the message shown in the figure18. Continue adding properties for the above html, you will need a `company name` and a `contact phone`.

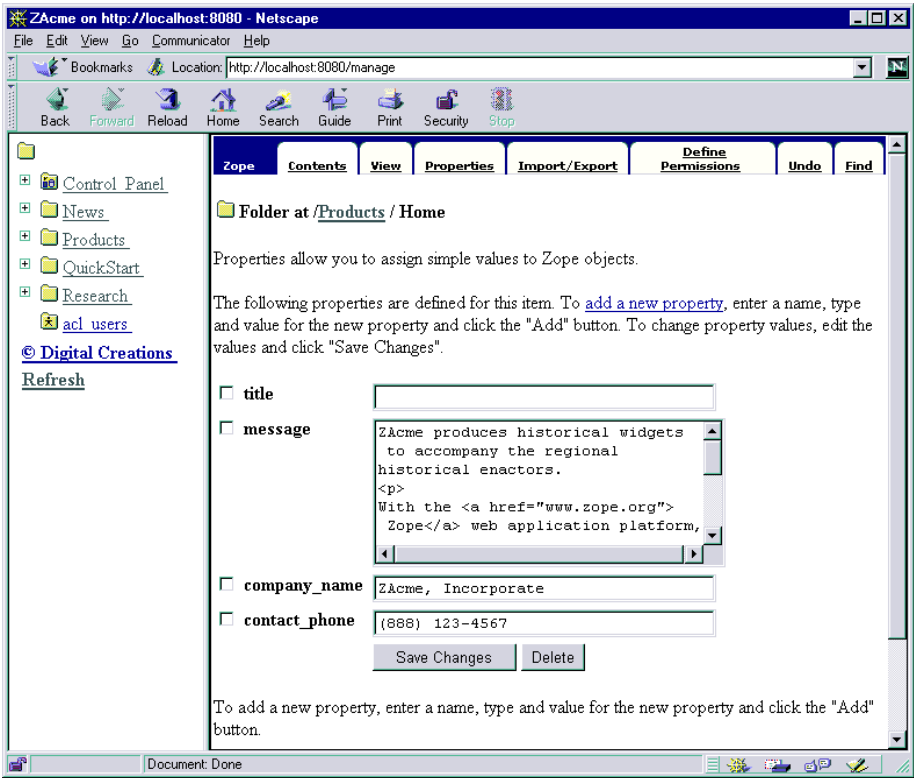

Figure 18. Folder properties used by the simple website application

After creating the **Products web site**, open another web browser and view your masterpiece. Creating the properties allows changes to the appearance of the site without having to change its HTML. .

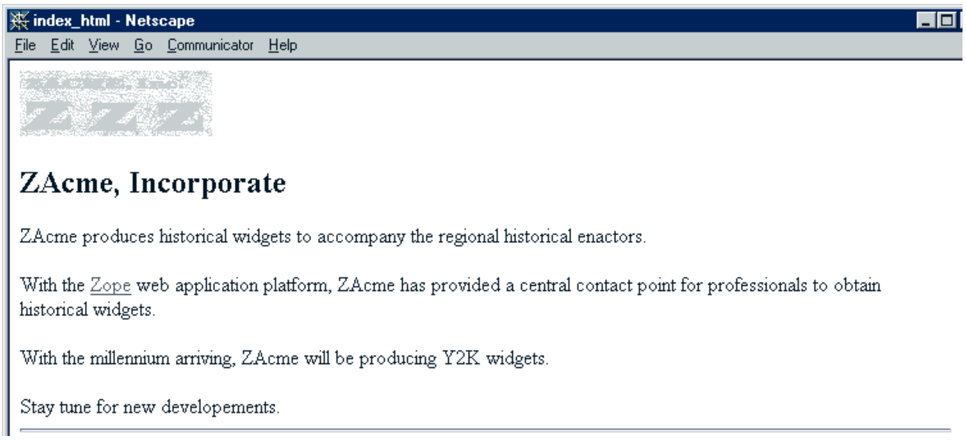

Figure 19. The home page displayed by the simple website application

Stan has completed his two tasks. He has created a public website for the Products, Research and Public Relations (News) departments. The second task of giving individuals the ability to change manage their own department's content has been established.

Creating a Configuration Interface

Stan is having fun, now, and wonders if he can automate the simple website application. It may be advantageous to re-create the application for other departments. We can copy and paste the Home folder, but then we would have to manually edit properties and upload a new image each time we made a copy. A simpler method is to automate this process by creating a *configuration interface*. A minimal configuration interface consists of a document that displays a form to collect configuration information and a document that uses DTML scripting to copy the folder and apply the new configuration.

Thus, the original application object serves as a template for creating new applications. While scripts could be written to build an object from scratch, it is often easier to start with an existing object, which is used as a template. The use of properties in the original application make the customizing task easier.

To illustrate the use of Zope applications, we will create a very simple website application. In the root folder of your Zope application, create a new Folder called `Home` (figure 20).

The Home folder will need some properties that we are interested in. Enter the *Properties View* of the Home Folder and

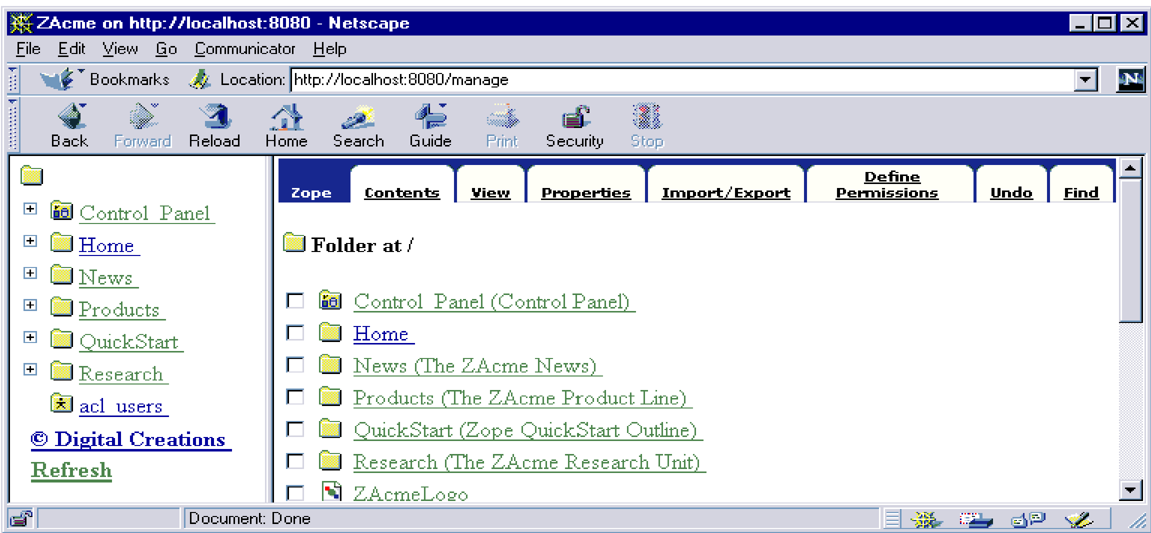

Figure 20. Home folder under root

add the following properties:

- Enter `company_name` into the "id" field. Select "string" type and *add* to complete the transaction.
- Continue adding the `contact_phone` with the type "string" and *add* to complete the transaction.
- The final property need is the `message` with the type "text" and *add* to complete the transaction

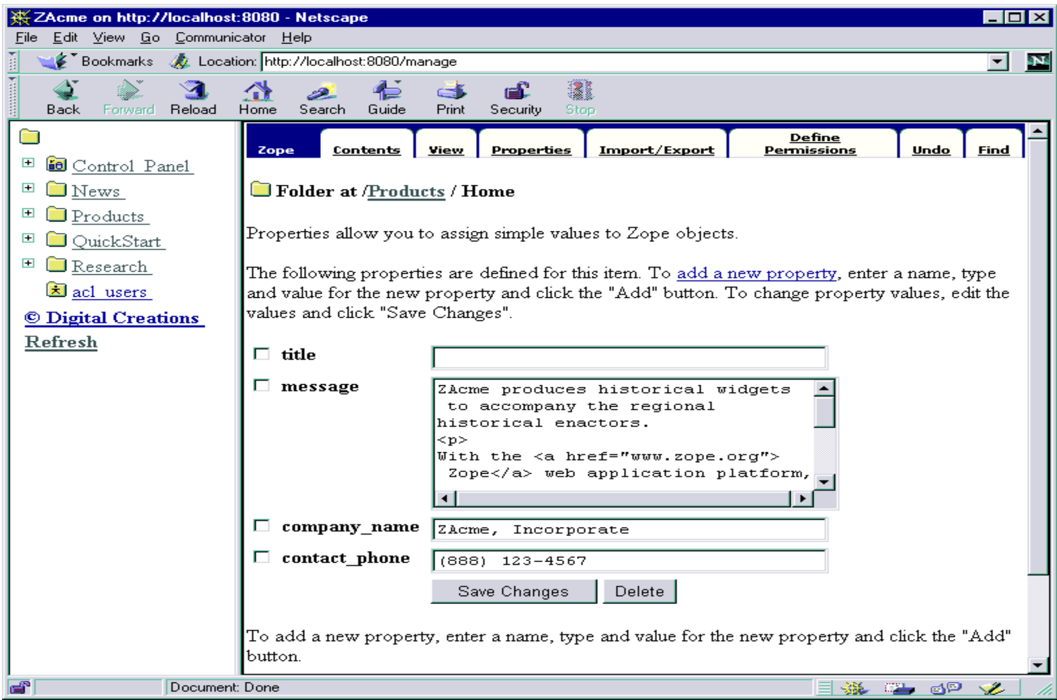

Figure 21. Home Folder Properties

In the *Home* Folder, create a DTML method named *index_html*. Edit the *index_html* and place in the copy of the home page we created in the previous section. You can copy and paste your previous DTML or type in this one:

```
<dtml-var standard_html_header>
 <dtml-var logo>
 <h2><dtml-var company_name></h2>
  <p><dtml-var message></p>
  <hr>
  <p>For more information, contact us at <em>
  <dtml-var contact_phone></em>
<dtml-var standard_html_footer>
```

To create a configuration interface for our simple website application, we need to create a *Product*. Go to the C*ontrol Panel* and select "add Product". Name the "product id" `InstantSite` with the "title" `Instant Site Web Site Builder` in Figure 22.

On the *navigation frame*, click on the *Control Panel*. Figure 23 shows the Control Panel with the product `InstantSite.` The icons next to the product indicate whether the products were external or internal. The open product icon indicates the internal product Phil created.

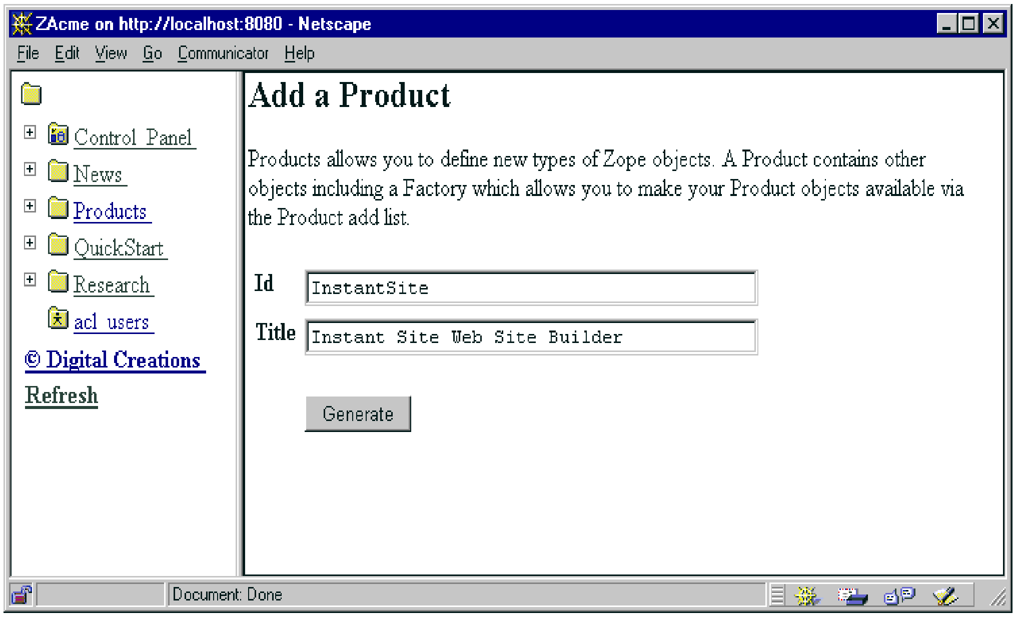

Figure 22. Add InstantSite Product

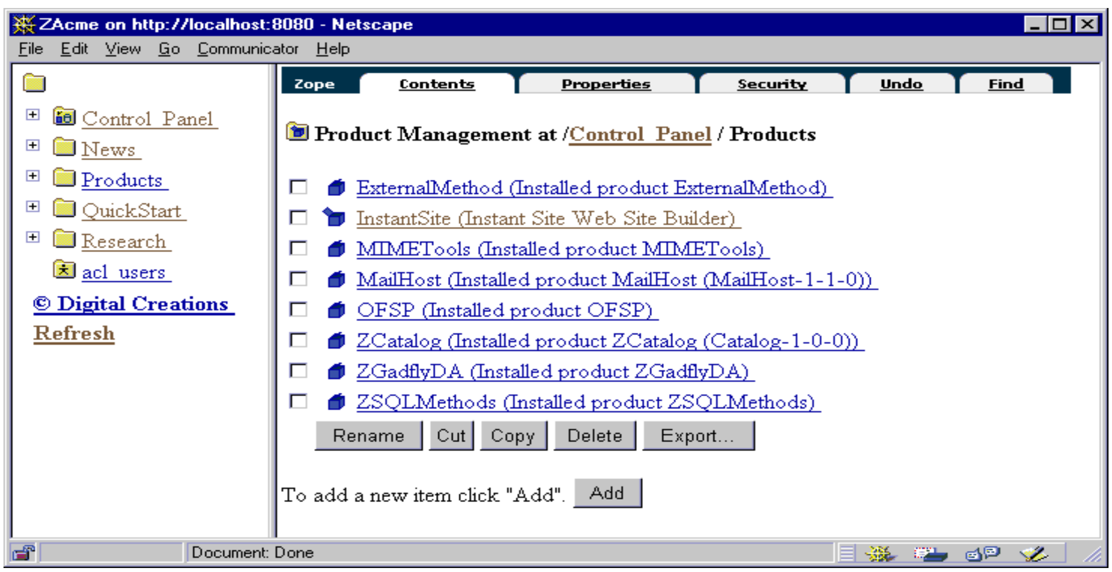

Figure 23. Product listed under Control Panel

Under the `InstantSite` product folder, add a DTML Method that displays a form prompting for information about the company. Figure 24 shows the source for a DTML Method, *`Designer`*. Add the method, name it "`Designer`", then cut and paste the source shown in figure 24. .

```
<dtml-var standard_html_header>

<H2>Home Page Designer</H2>

<FORM ACTION="Builder" METHOD="POST"
      ENCTYPE="multipart/form-data">
<TABLE CELLSPACING="2">
<TR> <TH ALIGN="LEFT" VALIGN="TOP">Id</TH>
     <TD ALIGN="LEFT" VALIGN="TOP">
     <INPUT TYPE="TEXT" NAME="new_id"
SIZE="40">
     </TD></TR>
<TR> <TH ALIGN="LEFT" VALIGN="TOP">Company
Name</TH>
     <TD ALIGN="LEFT" VALIGN="TOP">
     <INPUT TYPE="TEXT" NAME="new_name"
SIZE="40">
     </TD></TR>
<TR> <TH ALIGN="LEFT" VALIGN="TOP">Phone</TH>
     <TD ALIGN="LEFT" VALIGN="TOP">
     <INPUT TYPE="TEXT" NAME="new_phone"
SIZE="40">
     </TD></TR>
<TR> <TH ALIGN="LEFT"
VALIGN="TOP">Message</TH>
     <TD ALIGN="LEFT" VALIGN="TOP">
     <TEXTAREA NAME="new_message" ROWs="10"
COLS=40>
     </TEXTAREA>
     </TD></TR>
<TR> <TH ALIGN="LEFT" VALIGN="TOP">Logo</TH>
     <TD ALIGN="LEFT" VALIGN="TOP">
     <INPUT TYPE="file" NAME="new_logo"
SIZE="25">
     </TD></TR>
<TR><TD></TD>
    <TD><BR>
    <INPUT TYPE="SUBMIT" VALUE="Create Home
Page">
    </TD></TR>
</TABLE></FORM>

<dtml-var standard_html_footer>
```

Figure 24. Source of the HomeDesigner DTML Method

The form that is created with the DTML source is displayed in Figure 25

The form collects the configuration information that will be applied to the copied version of the Home folder.[1]

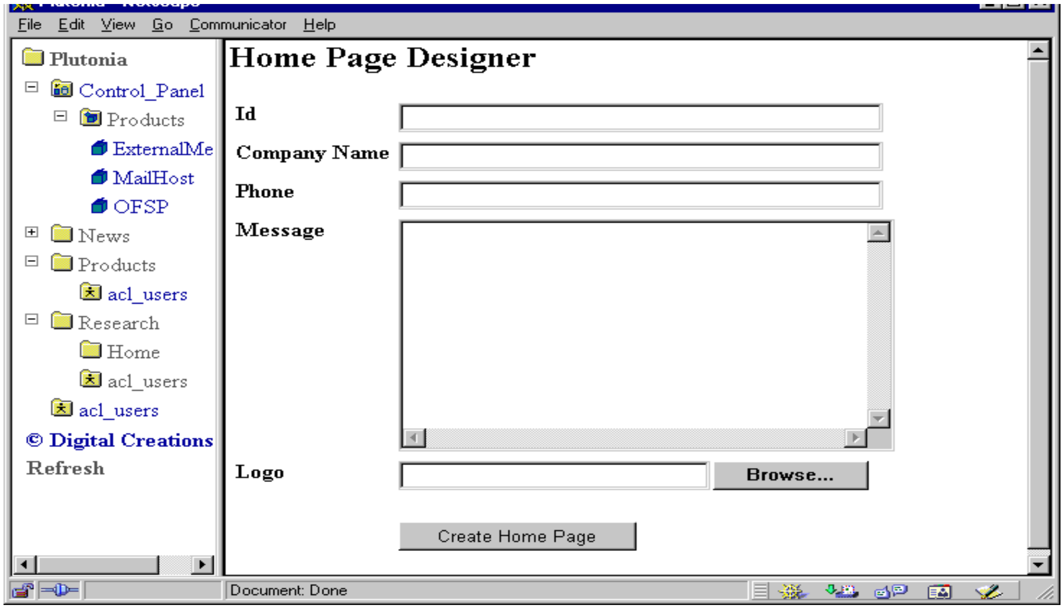

Figure 25. Designer Form

Let's go through the Builder script step by step. After the insertion of the standard header, there is a *with* tag that evaluates an expression involving *manage_clone*.

The Folder method *manage_clone* allows an object to be copied into a folder with a given id. In the expression in Figure 26, the *manage_clone* method is being invoked on the Folder in which the builder is run. The object being cloned is the template object, *Home*.

The new *id* is given by the expression *REQUEST['new_id']*. This expression is used to make sure that the id collected by the Designer form is used, rather than the id of the folder. The *manage_clone* method returns the object clone, which allows the object to be manipulated within the *with* tag.

Inside the *with* tag, the *manage_changeProperties* method is called on the newly cloned object to replace original property values with values from the request. The *manage_upload* method of the logo of the cloned object is called to supply a new logo image from the image provided in the Designer form. This completes the configuration of the new home page.

The action of the Designer form is another DTML Method, *Builder*, that uses DTML scripting to copy the original home page folder and apply the customization. As shown in Figure 26, the source for Builder sets the properties of the Home folder to the properties defined by Designer.

1. Note, to identify *new_logo* as a file upload, it is given the type *file*. In order to send file uploads, the Designer form must be configured with the attribute, *ENCTYPE = "multipart/form-data"*.

```
<dtml-var standard_html_header>

<dtml-with "manage_clone(Home,REQUEST['new_id'],REQUEST)">
  <dtml-call "manage_changeProperties(
      title=REQUEST['new_name'],
      company_name=REQUEST['new_name'],
      message=REQUEST['new_message'],
      contact_phone=REQUEST['new_phone'],
    )">
  <dtml-call "logo.manage_upload(REQUEST['new_logo'])">
</dtml-with>

<dtml-call "RESPONSE.redirect('manage_main?update_menu=1')">

Congratulations!

<dtml-var standard_html_footer>
```

Figure 26. Source of the Builder DTML Method

After the *with* tag, the redirect method is called on the *RESPONSE* object to redirect the browser to the Contents view of the destination folder. A query string, *"?update_menu=1"* is provided to cause the navigation pane to be updated too.

Finally, a message is output indicating success. Note that the text of the success message, as well as the text output by the standard header and footer is not output unless, for some reason, the browser ignores the redirection or shows the output followed by the redirection.

To use the simple website application as a template, the original Home folder along with the configuration objects must be accessible by other folders, such as through acquisition. By turning the website application into a Zope product, however, we avoid filling up top-level folders, while at the same time making the website application accessible to all folders. As product HomeSite, a new website could be added to almost any folder.

Turning an Application into a Zope Product

A Zope *product* is an object that defines information. Stan wants to make his web site builder application available throughout ZAcme's Zope installation. The most common use for a product is to provide facilities for adding specialized objects. Products are managed in the *Products Management* folder of the **Zope Control Panel** .

The first step in converting an application to a product is to create a new Product in the Products folder. To create a new product, Stan selects the Products Management folder in the *Control Panel's Contents view*, as show in Figure 26. Here we

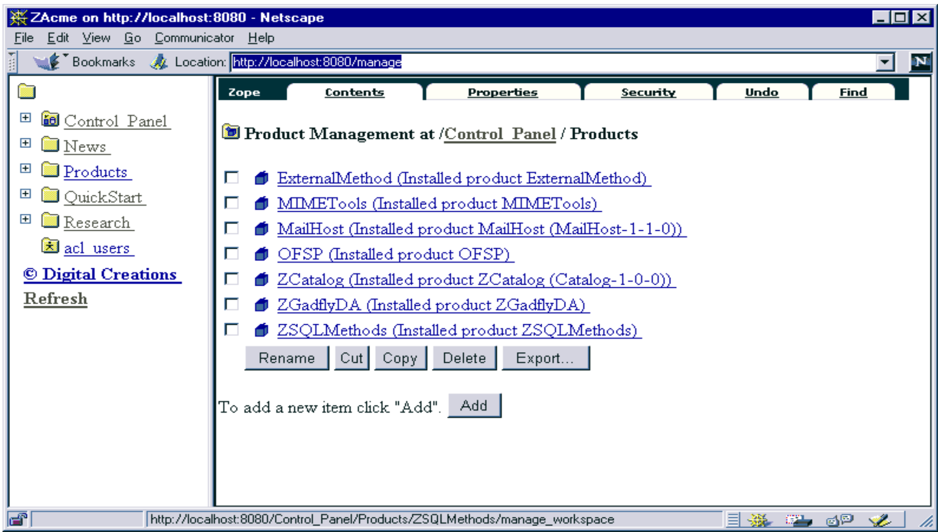

Figure 26. Products folder

add a product by clicking on the "Add" button and entering the "Id" as `InstantSite` and "Title" as `Instant Site Web Site Builder`, shown in Figure 27.

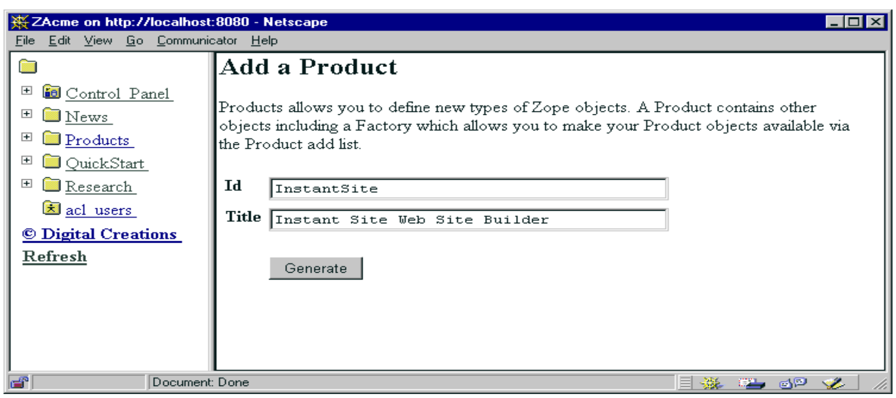

Figure 27. Add product form

After clicking the “Generate” button, we select the new product in the products folder to display its Contents view (Figure 28)

The product is managed much like an ordinary Zope Folder. However, products have two unique features, an extra item in their Add List to create factory objects and a Distribute view for creating product distributions.

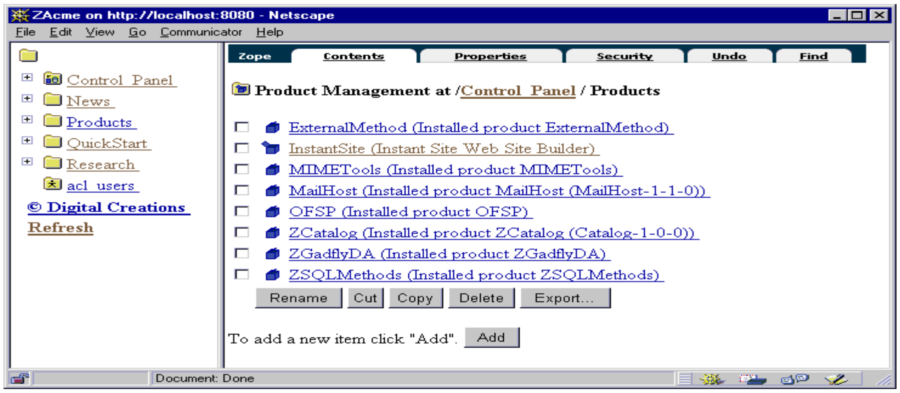

Figure 28. A new product folder

Factories

Factories provide the connection between objects in a product and the Zope Folder Add List. An entry appears in the Add List for each factory defined in a product. A product may have any number of factory objects.

Creating a factory for our InstantSite product would enable our product to be duplicated simply by selecting the factory from the Add List. Before creating the factory for our product, we need to copy the Home folder and the configuration DTML Methods, `Designer` and `Builder`, into the `InstantSite` product using the Zope copy and paste mechanisms.

Stan completes the following steps to duplicate the `InstantSite` product.

- Step One: Return to the root level and select the Home Folder, Designer and Builder DTML methods by placing a check in the block next to each item.
- Step Two: Scroll down to the bottom of the screen and select *Copy.*
- Step Three: Return to the Product Management screen under the InstantSite Product and select *Paste*.

After copying and renaming these objects, the `InstantSite` product appears as in Figure 29.

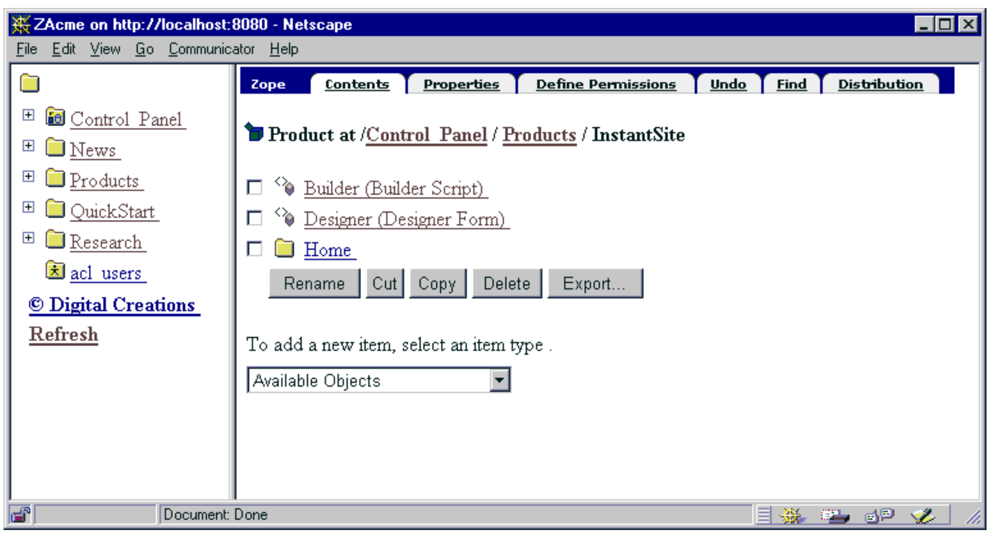

Figure 29. Home, Designer and Builder In InstantSite

To create a factory, simply select the "Zope Factory" entry in the Available Objects and click. The "Create an Object Factory" form is presented, as shown in Figure 30. Stan entered `Instant Web Site` as the "Add List Name" and selected "Designer" as the method to be invoked when the factory is added. Thus, when "Instant Web Site" is selected from the Add List, the Designer DTML Method prompting for the configuration information is rendered.

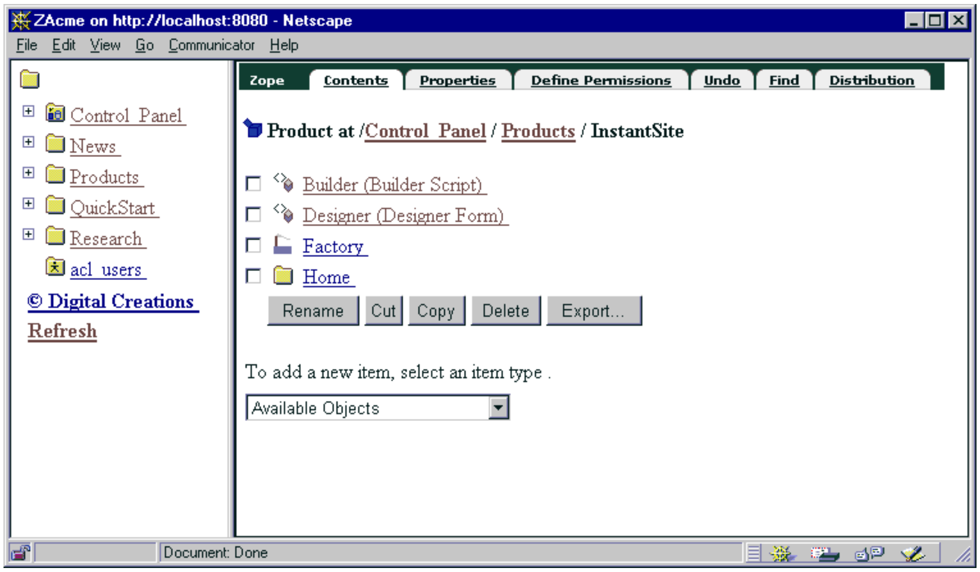

Figure 30. Factory available in Control Panel

When the factory is created, the permissions for that product must be established. The `initial method` should be the method invoked when using the product. Upon reviewing the "Define Permissions" view, you can disable or enable permissions based on the outcome of the product (Figure 31).

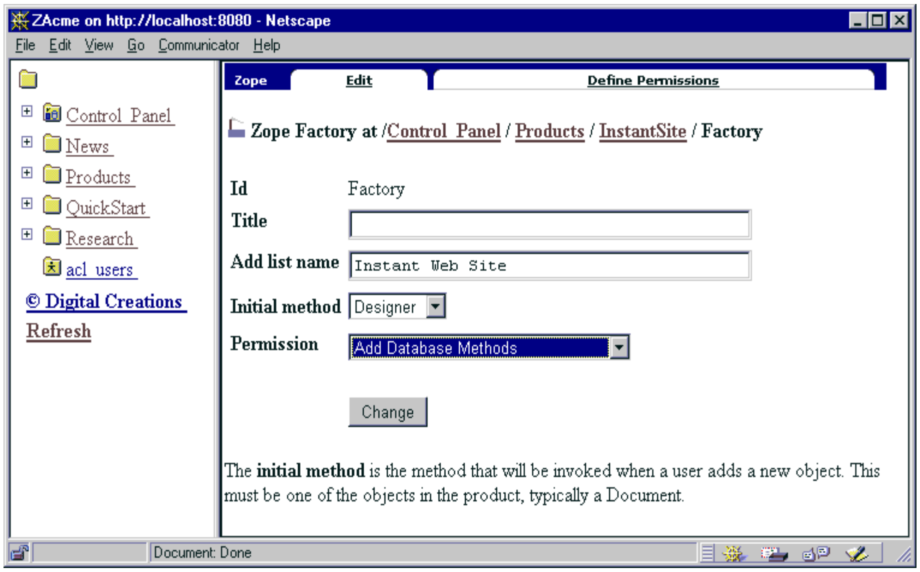

Figure 31. Product Permissions defined.

Placing the template object and configuration DTML Methods in a product folder made it unnecessary for the objects to be in the destination folder. When an object is added, however, these objects must be somehow accessed. When a new website is added to a destination folder, the Designer script is accessed by the effective address:

```
manage_addProduct/product_name/factory_name/object_name
```

Once the Designer script calls the Builder, the Builder script redirects the browser to the *manage_main* view of the address above. To redirect the browser to the contents view of the destination folder, the Builder's redirect method needs to be modified using function *DestinationURL*. This modification can be seen in Figure 32.

```
<dtml-var standard_html_header>

<dtml-with "manage_clone(Home,REQUEST['new_id'],REQUEST)">
  <dtml-call "manage_changeProperties(
      title=REQUEST['new_name'],
      company_name=REQUEST['new_name'],
      message=REQUEST['new_message'],
      contact_phone=REQUEST['new_phone'],
    )">
  <dtml-call "logo.manage_upload(REQUEST['new_logo'])">
</dtml-with>

<dtml-call
"RESPONSE.redirect(DestinationURL()+'manage_main?update_menu=1')">

Congratulations!

<dtml-var standard_html_footer>
```

Figure 32. Source of the modified Builder DTML Method

Creating Product Distributions

A product can be turned into a distribution file[1] that can be distributed and installed in Zope installations just like other Zope Products. The product *Distribution* view (Figure 33) is used to create a distribution file. Stan accesses

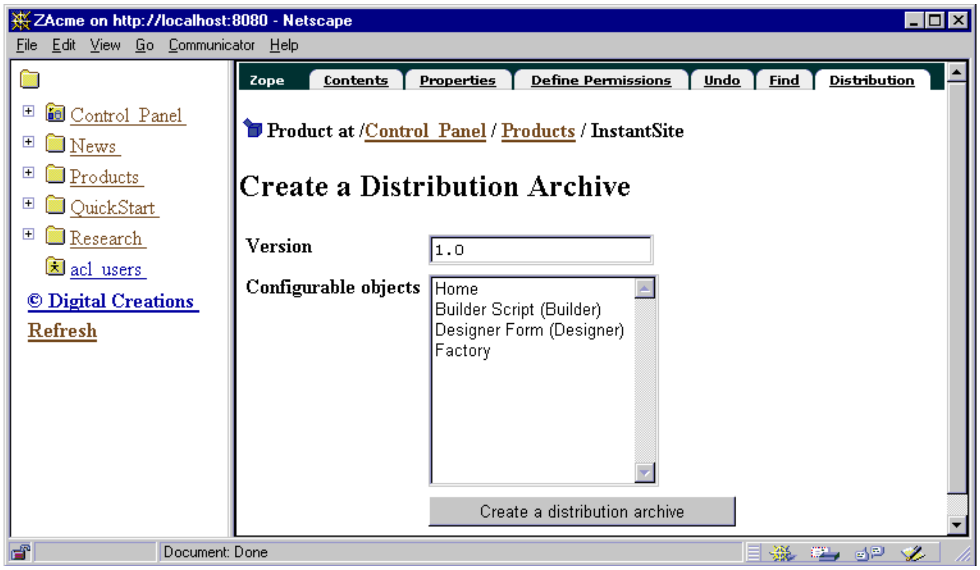

Figure 33. Product Distribution view

"*Distribution" tab* on the Instant Site Product.

When creating a distribution, a version must be specified. The version is a product property that can be edited using the Properties view. If a numbered version scheme is used, then an incremental version is provided as the default entry in the Distribution view.

When your product distribution is installed, it will appear in the products list in the control panel products folder display after Zope is restarted. The objects you wish to constitute as configuration objects of the distributed product can be selected from the list of available objects.

When you click on the “Create Distribution” button, your browser is redirected to a URL for the distribution file and the distribution can be saved from your browser to your local file system. The distribution file is generated with a name of the form:

```
product-version.tar.gz
```

To install this distribution in another Zope distribution, simply extract[2] the distribution file in the Zope installation directory.

1. The distribution file is a GNU zipped tape achieve (tar) file. Products are “installed” by simply extracting the distribution file in a Zope installation.

2. The exact system command used to extract a product distribution file depends on your system. On Unix systems, use the command:

```
gunzip <distribution-file | tar xovf -
```

On windows system, use a utility such as Nico Mak Computing's WinZip.

Tutorial Part One - Summary

Stan has accomplished his assigned task; to create a public web site for ZAcme, Incorporated and to delegate responsibility to the three departments. Along the way, Stan created a Zope product to allow others to build websites and folders customized for the users. But, there are a multitude of Zope items that Stan has yet to explore with Zope. What else can he do? Stan wants to do more!

Tutorial - Part Two

Lets kick this tutorial into fast gear. We will go into several object reference items of Zope and learn how to use them. Hopefully, this will take Stan further into some of the operations he has already accomplished and allow him to explore.

Stan's goals are to modify the company web sit by adding a sidebar to all the pages and allowing the contact person on the web site be consistent with each department. Now that the web site is public, he wants to make changes without an audience. Stan outlines his experiment below;

- Working in Private with Versions
- Working with Folders and folder contents.
- Creating and managing files
- Creating and managing images
- Setting User Folders and Users
- Creating and Managing DTML Methods
- Creating and Managing DTML Documents
- Working with External Methods.

Working with Versions

Stan will be experimenting with the Zope application, but doesn't want anyone to see his work or affect anyone else working on the application. No Problem, the Zope application has a feature named **Versions** that will allow anyone to experiment with the application before saving the transactions.

Versions

Often when fixing something, something else gets broken as a result. Updating a website can involve making many changes across multiple objects, and until all of those changes are done, parts of the site might not function properly. The Version object create a private environment to work in where changes are not seen by other users until specified. This allows for a Zope site to be modified in Zope itself without having to take the site off-line or breaking links and object references.

Creating a Version

To create a new Version, Stan displays the *Contents view* of the folder to create the new **Version.** Select "Version" from the Available List, then click the left mouse button. On the *Add Version* screen give the "id" `Upload` and a "title" `Upload New Docs` and click the "Add" button. The Version in the folder contents is marked with a large red diamond.

Stan creates a Version named "Upload" as the root directory. Placement of the version does effect where work can be accomplished. For example, if Sally in Research wanted to do some work in the Research folder, she could create a version in that folder that would effect items in the Research's folder hierarchy down.

Joining a Version

To join a Version, open the version to join by clicking on its id or icon in the Contents view of the folder containing it. The default screen of a Version is the "Join/Leave" screen, as shown in Figure 34.

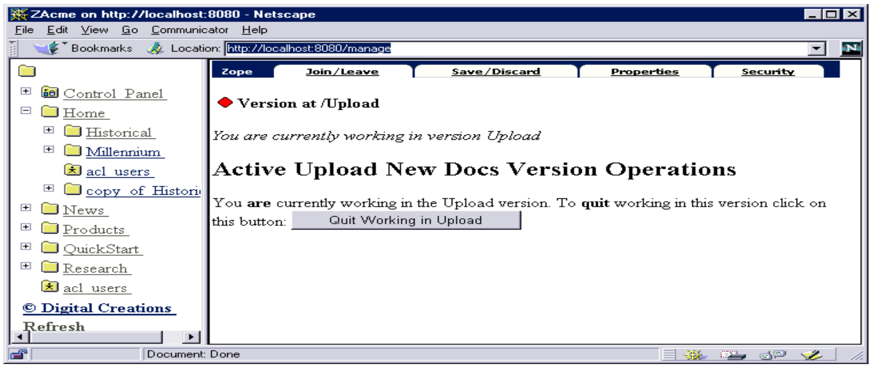

Figure 34. Joining a Version

The Join/Leave screen displays the active/inactive state of the Version, and contains a button to start or stop working in the Version. Click "Start working in [version id]" to start working in the version.

Working in a Version

While a Version is active, a line beneath the current object identifier (the icon and path to the object presented in the workspace) will indicate that a Version is active and give the Zope path to the active Version. Stan notices after joining the version `Upload` the message "You are currently working in version Upload" shows on each manage screen.

If there is no active Version, nothing is displayed. While a version is active, objects that are modified in the Version will be marked with a small red diamond following their title and id in a folders contents (Figure 35). .

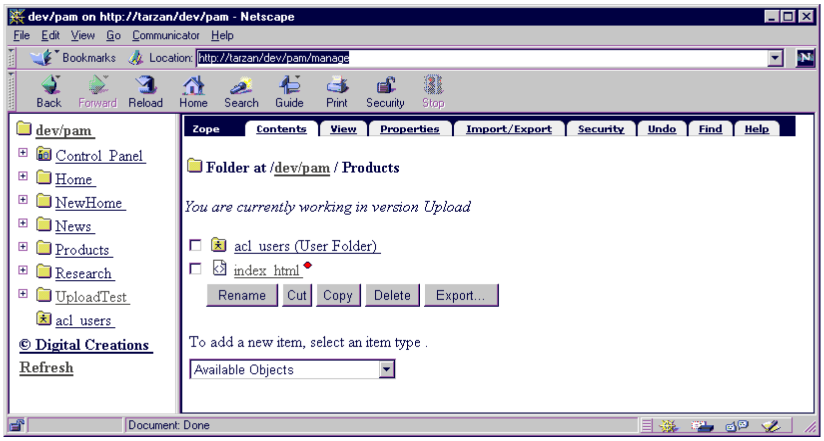

Figure 35. Folder with items changed in a Version

To ensure safety, when an object is modified in a Version it becomes “locked” outside of the version. Another user will not be able to modify that object until the changes in the Version have been committed. This helps to prevent conflicts when the between Versions.

The Undo list in an active version only allows actions to be undone that were done inside the version. This prevents version users from accidentally undoing an action that would affect the outside site and possibly the changes occurring within the version.

Leaving a Version

Anything Stan has to stop working on this experiment before finishing, he can leave the version by clicking on its id or icon in the Contents view of the folder containing it. The default screen on a Version is the Join/Leave screen. The Join/Leave screen displays the active/inactive state of the Version and a button to start or stop working in the version. Click “Quit working in [version id]” to quit working in the Version. Quitting a version does not save the changes Stan has made, this option will be discuss in the "Saving or Discarding Version Changes" section.

Folders

Folders are objects that contain other objects and folders. Folders also have *properties* which are available to all of the objects which are contained in the folders. The default view on a folder is its Contents view. The Contents view is the most common interface to work with Zope since it is where objects are created, destroyed and accessed for editing.

Creating Folders

- To create a new Folder, display the Contents view of the object to create the new Folder in.
- Then select “Folder” from the "Available Items" and click. The “Add Folder” form will be displayed.
- Enter the id and title for the new Folder.
- The Add Folder form will also allow you to select to have a “Public Interface” and/or “User Folder” created for the Folder automatically.
- Stan decides to create two folder under Products; `Historical Widgets` and `Millennium Widgets.`

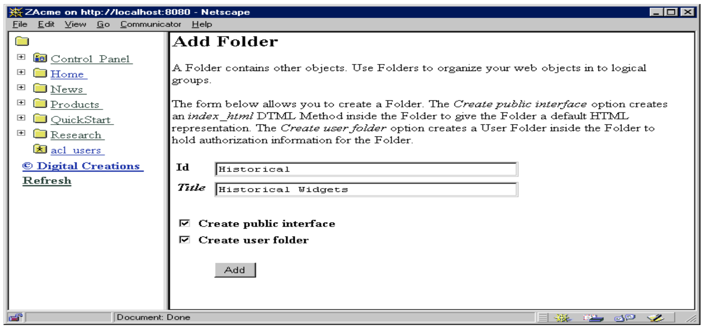

Figure 36. Add Folders

Notice that the folders have small red diamonds next to them indicating that they were created under a version (Figure 37).

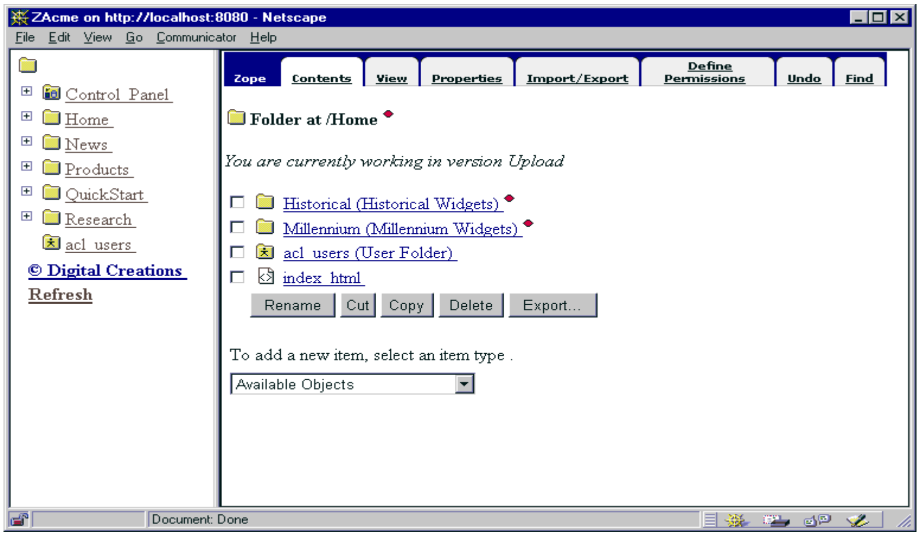

Figure 37. Content View with Folders in Versions

Working with Folder contents

The Contents view of a Folder is the primary way to interact with the Folder's contained objects. At the top of the view is the object's location (from the URL) and type. If a version is currently active, text signifying the active version is displayed here also. If there are any objects in the Folder, they will be listed next with a checkbox (for the purpose of selecting the object), an icon, and the object's id and title. This is known as the *Object List*. Finally, there is an *Available Object* which is a pop-up list containing the types of objects.

To manage an individual object, click once on its icon or id and title text. To copy an object, select it by clicking its respective checkbox followed by a click on the "Copy" button. Once there is an object in the clipboard, a "Paste" button will be displayed. To paste a copied object, click the "Paste" button. A screen prompting for a new id (the objects original id already inserted) will appear. Set the id and click "OK" to paste the copied object into the folder. Objects may be copied and pasted to different folders in the same system (Figure 38). To delete objects, select the objects to delete by clicking their respective checkboxes and click the "Delete" button.

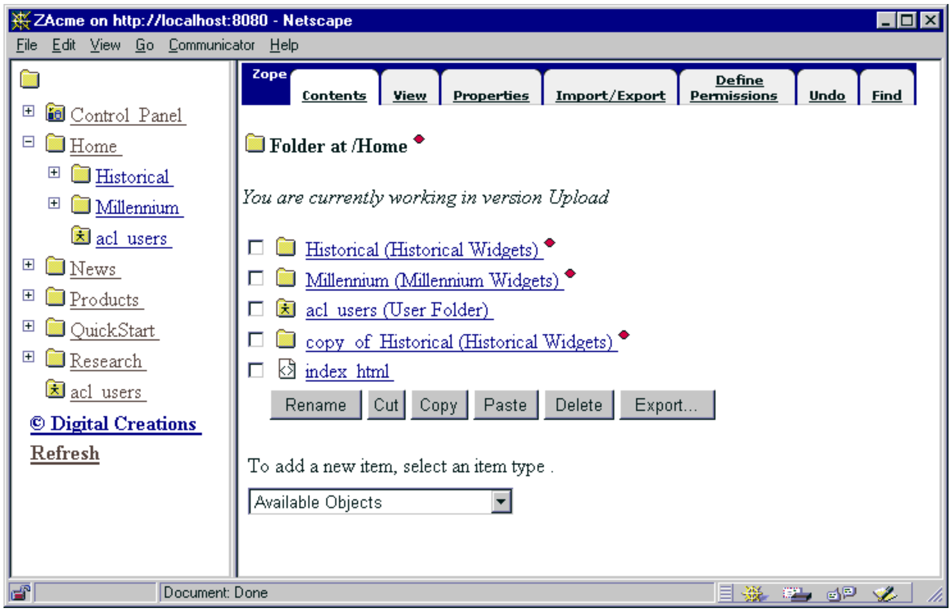

Figure 38. Copy and Pasted Historical Folder

Setting and using Folder Properties

Stan has decided to add a property to the folders. He creates a standard description for the widget product. Properties can be used by any contained object. To set properties, go to the Properties view on a Folder. Current properties are listed first, along with a checkbox and a text area. To change properties, simply enter the value changes in the text area and click "Save Changes". To delete a property, click its check box and "Delete". Adding a new property requires that you enter an id, select the property type from the drop down-list, and fill in the value of the new property. Once the new information is entered, click "Add". The Properties view of a Folder is shown in Figure 39.

Properties can be one of several "Types." For a detailed description of each type see Table 40.

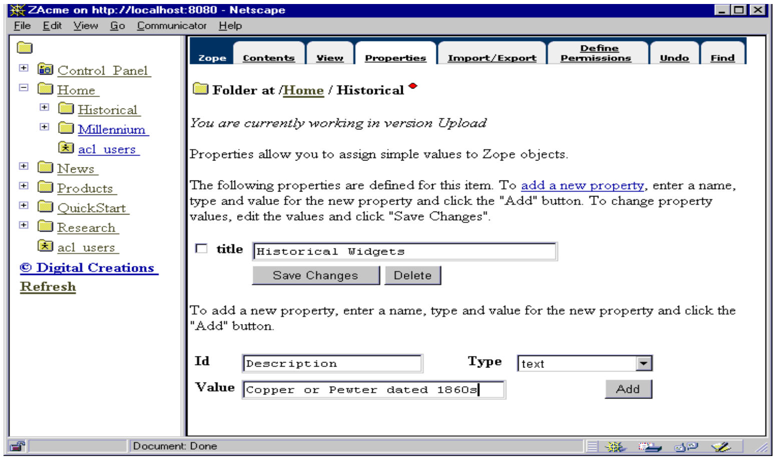

Figure 39. Properties of a Folder

TYPE	Description
String	A string of characters, like a title or an email address
Boolean	Represents two truth values, either true or false
Date	A DateTime value. DateTime values can be entered and formatted in objects in a variety of ways. See "DateTime Formatting" in the DTML Guide for details
Float	A floating point number, like 4.55
Int	An integer
Lines	Lines are lists of items that can be looped over the Document Templates. After a Lines property has been added it is editable in a large text area instead of a small field. Each line is a new element in the sequence.
Long	A long integer (for really large numbers)
Text	After adding a new Text property, it will be editable in a large text are instead of a small field
Token	Tokens are a spaced-separated list of items that can be looped over the Document Templates. For example, adding a token with the value of "eeny meeny miney mo" can be rendered in HTML as a bulleted list, a select box, or any other sequence. A "yes no sometimes" list can be changed in one place to a "yes no often rarely."

Figure 40. Description of Property types

Files

Zope File objects are miscellaneous binary objects that are accessed just like normal files over the Web. File objects can be any object and are presented in their raw form. They can be Java classes, ActiveX controls, word processing documents, PDF (Acrobat) documents, spreadsheets, archives, and so on.

Creating Files

To add a new File object, select the Contents view of the folder to add the file in. Then select “File” in the Add List and click the “Add” button. On the Add File screen there are fields to set the id and title, and to select a file on a local hard disk to create the File object out of. If no id or title is set, the uploaded file name is set as id and the file path the is set as title. After the Add File screen is completed, click the “Add” button.

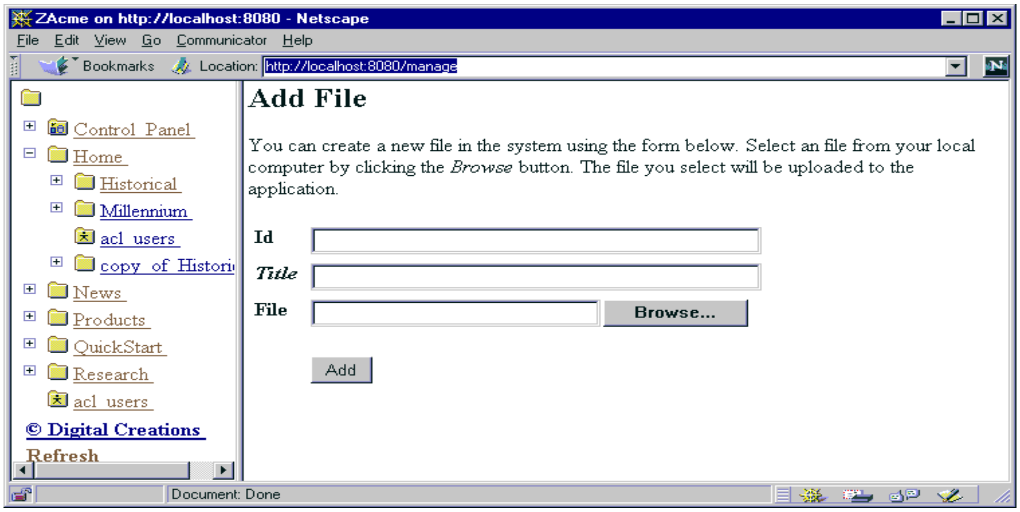

Figure 41. Add File

Uploading new File data

Stan decides to upload HTML files he has created into this new file. He has two choices. First, he can upload the file when he creates the file with the "Browse" button. His second choice is to create the empty file and upload using the "Upload" view (Table 41). To upload new data, click on the "Upload" tag, then click the “Browse...” button to select the file to upload, and click “Change.”

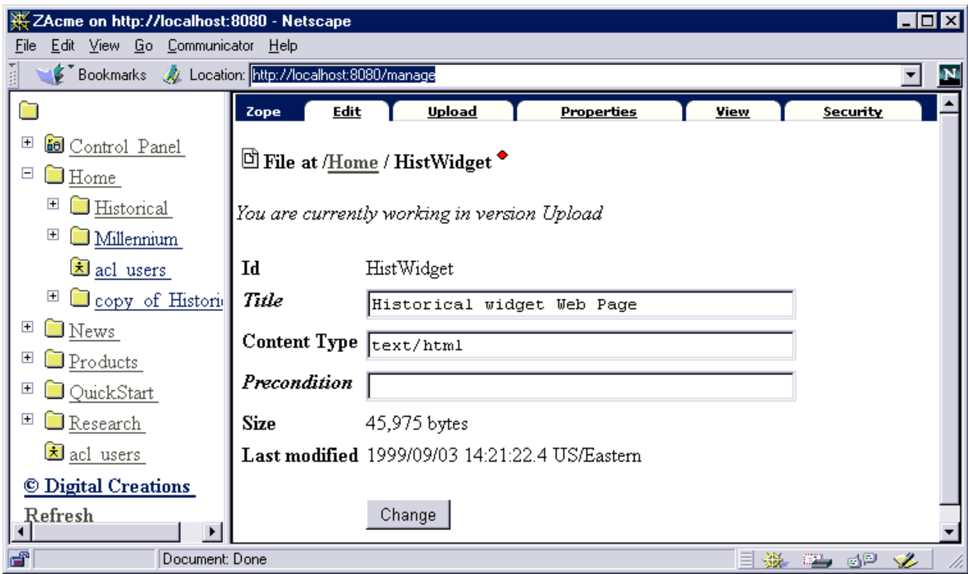

Figure 42. File Edit with Content Type

Stan enters `HistWidget` as is "id" and `Historical Widget Web Page` as the "title" for his file. When added, the content text of the file is shown as unknown. When changing to the "Upload" view, Stan can browse his hard drive and select his html file. After uploading the html file, the "Content Type" automatically updates to reflect the type of file (Table 42).

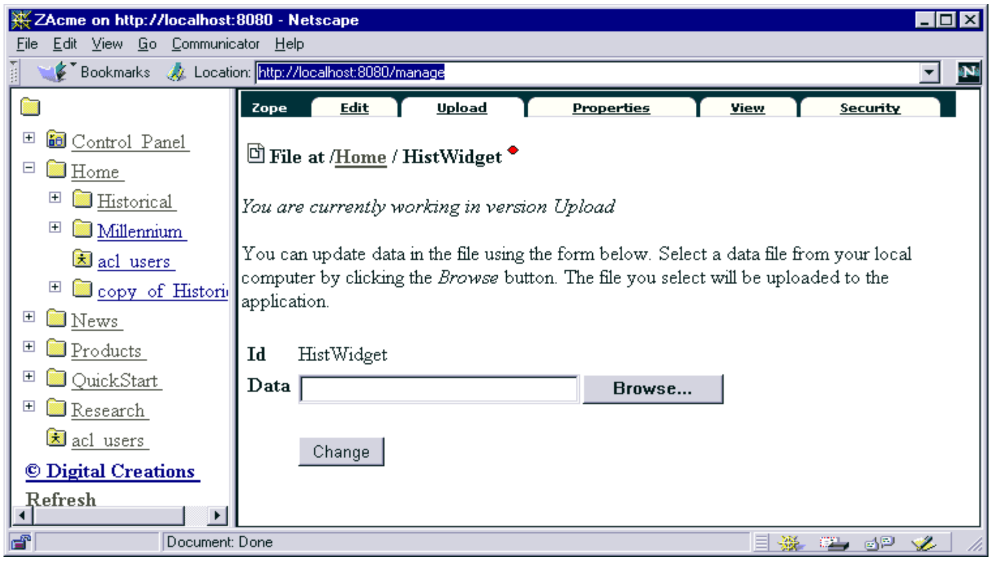

Figure 43. File Upload

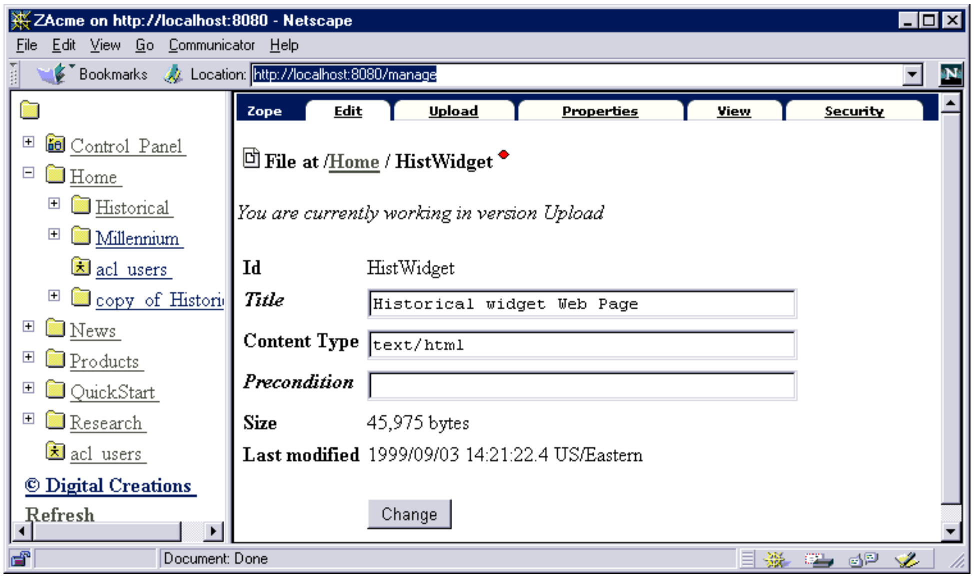

Figure 44. File Uploaded from Hard Drive

Images

Zope Image objects are binary image files and are accessed just like a normal image over the Web. Zope Image objects, however, can automatically present the HTML, such as HTTP headers, needed to access them.

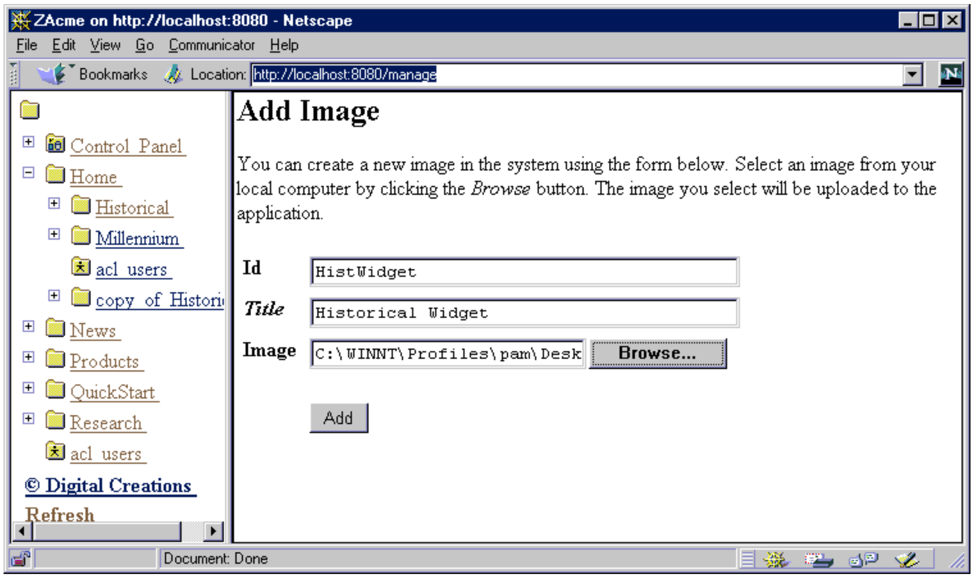

Figure 45. Creating and Uploading Images

Creating Images

To add a new Image, select the Contents view of the folder to add the Image in. Then select "Image" in the Add List and click the "Add" button. On the Add Image screen there are fields to set the id and title, and to select a file on a local hard disk to create the Image object from. If no id or title is given, the uploaded file name is set as id and the file path is set as the title. When creating GIF or PNG images, the height and width are determined automatically, but can be changed in the Properties view. After the Add Image screen has been filled in, click the "Add" button.

Stan add the gif file that contains a picture of the Historical Widget. He browsed his hard drive and uploaded the image when he created it (Table 45).

Editing Images

To edit an Image, click on its id in the Folder's Contents List. The Edit view on an Image is the default view for Image objects. Here, the title and MIME-Type may be modified. For example, an uploaded image from a Macintosh might be interpreted as mime-type *"application/x-macbinary"* and would need to be modified to *"image/jpeg"* if it were a JPEG image. The object's size and date last modified are also given.

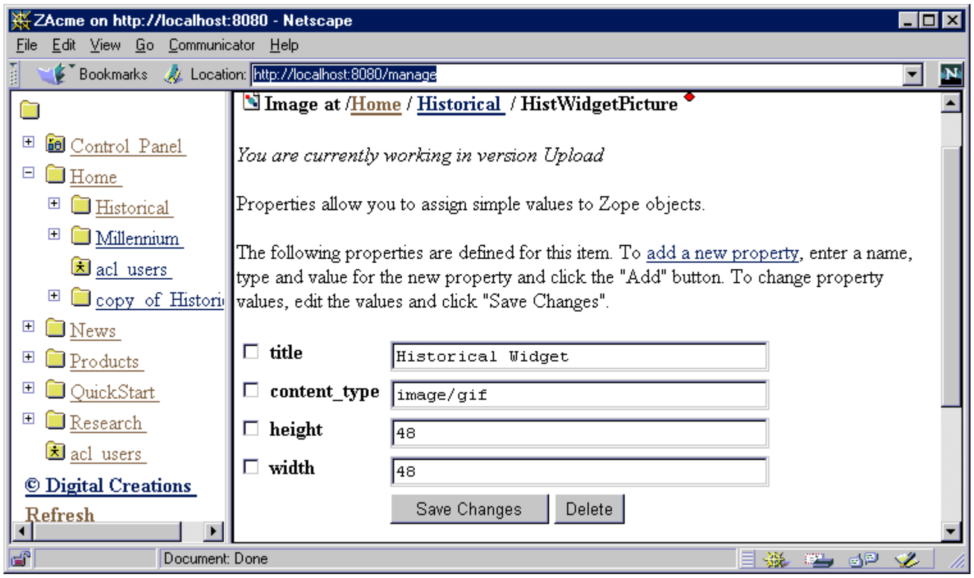

Figure 46. Image Properties

Setting and using Image Properties

To set properties on an Image, go to the Properties view of that Image. Previously set properties are displayed first and can be edited or deleted. Next, there is a form to add a new property. To add a new property, set the id, type and value for the new property and click "Add" (Figure 46). For a description of property types, see Figure 40.

Viewing Images

In order to see what an Image looks like, click the View tab. The Image will be displayed in the workspace area. An Image can be put in document templates using a *var* tag identifying its id, such as <dtml-var mypictureID>. This will insert the HTML, <img src="mypictureID">. Alternatively, the Image can be used in an HTML image tag with its id as the source, such as <img src="mypictureID">.

User Folders and Users

The Zope security model allows you to create user databases inside a Folder. These user databases are a special kind of Folder called a *User Folder*. In addition to acting as a user database for a Folder, User Folders are special because the only thing you can add to them is user objects. This section describes the management of User Folders and users.

As ZAcme website grows, it is often desirable to give different people special privileges to sections of the site that apply to them. For example, a Public Relations department might want to keep a *Company News* folder. Only the PR staff should have access to that area, not Research and Development, and vice versa. User Folders allow users to be added at different points in the object hierarchy.

Managing security

The Zope security model is administered using the same management interface for managing other object information. The activities involved in managing security include the following:

- Adding users and setting their privileges with roles
- Setting permissions on objects
- Adding custom roles to users
- Using acquisition to centralize common security settings
- Configuring proxy information to make some objects have higher privilege
- Handling common problems

Creating User Folders

To create a new User Folder, display the Contents view of the folder to create the new User Folder in. Select “User Folder” from the add list, then click Add. User Folders always have the id “acl_users” and the title “User Folder”. There can only be one User Folder per folder.

Adding New Users

To add users to a User Folder, open the User Folder by clicking on its id or icon in the Contents view of the folder containing it or in the Navigation frame. As with folders, the default view of a User Folder is the Contents view. If there are Users defined in the folder, they are listed in the contents list with controls to select and delete them, similar to folder contents. Since only User objects can be added to User Folders, there is no Add List. To create a new User, click the “Add” button. Once the “Add” button is clicked, an *Add User* screen comes up, as shown in Figure 47. Give the user a name and password (enter the password in both fields to confirm) and assign the user his *Roles*. The roles define the users access to objects and what he may do with the objects. When the form is complete click “Add”.

Managing Users

As mentioned above, Zope uses User objects to represent people interacting with the system. These User objects have information that identifies the user, such as a username and password, as well as information about what they can do in the system.

In Zope, User objects are stored in a special kind of Zope Folder called a *User Folder.* This Folder acts as a database of Users that exist in the Folder in which the User Folder is defined. For instance, consider the following Folder:

```
/ZAcme/
    /ZAcme/Marketing/
:       /ZAcme/Marketing/East/
```

If you have a Folder called *Marketing*, and you want to turn over the control of Marketing to Mary, then you would create a User Folder in Marketing and add a User to it for Mary.

It is important to note, though, that Mary is defined only at the level of Marketing, is which she might be a manager. If Mary tried to do a privileged operation in ZAcme the operation would fail because she does not exist in ZAcme. However, Mary can do privileged operations in East, because East is contained in Marketing.

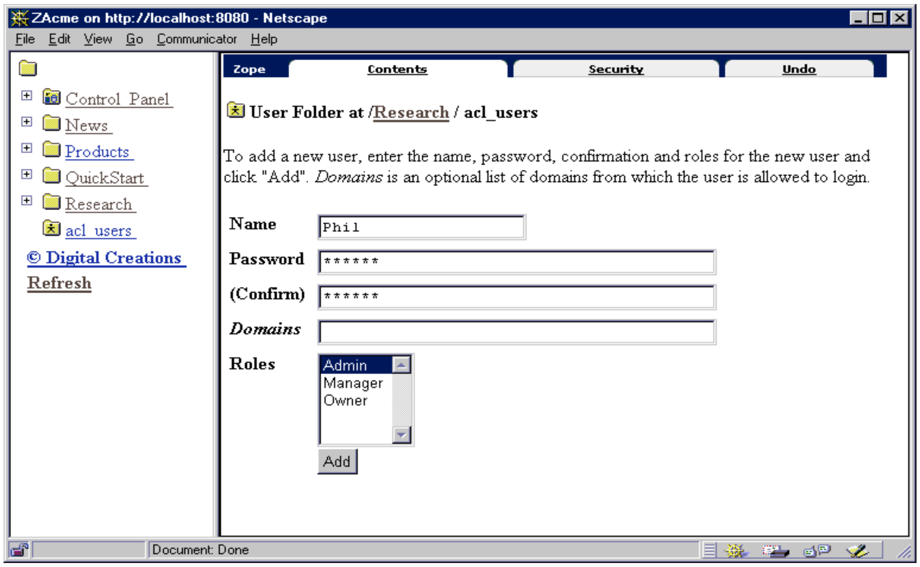

Figure 47. Adding a User to a User Folder

Zope Users can be identified by HTTP Basic Authentication, a location, or both. For HTTP Basic Authentication, the user will be prompted for a username and a password, which Zope will use to try to find a matching User. For a location, Zope will look at the host/domain name of the requesting computer, or IP address, and look for configured Users allowed to come from that location.

When Zope needs to verify provided authentication, it first checks the contents of User Folders defined at the level of the operation. For instance, if Mary tried to visit the manage screen of `East` in the above example, and then provided a username and password, Zope would check first in the User Folder for `East` for a matching username and password. Failing that, it would look next in *Marketing*, then finally in *ZAcme*. If all of these fail to produce a Mary with the correct password, Zope would return an *Unauthorized error.*

Editing Users

To edit Users in a User Folder, open the User object by clicking on the User's name in the Contents view of the User Folder. The *Edit User* screen is very similar to the *Add User* (See "Adding New Users") screen, only the Users name (Figure 48).

Deleting Users

Deleting User objects is like deleting regular objects in folders. Select the Users to delete by checking the corresponding checkboxes and clicking the "Delete" button.

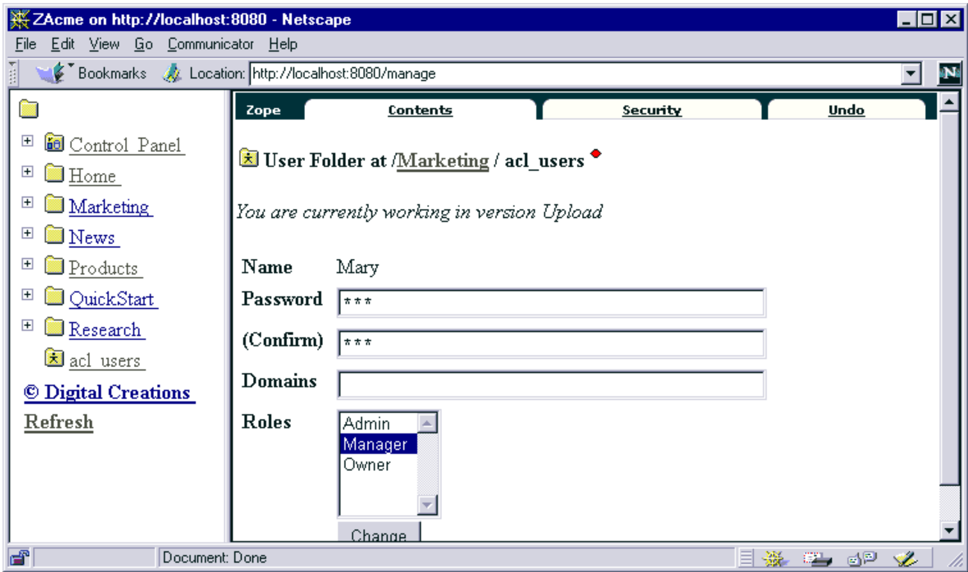

Figure 48. Editing Users

When to Use DTML Method vs. DTML Document?

DTML Documents are used to hold content. If you want to store a large document, or lots of text, or a body of information, use a DTML Document.

DTML Methods are used to carry out actions. If you want to render another object (like a DTML Document), execute a DTML script, or generate lots of dynamic content, use a DTML Method.

Also, you may notice that the DTML Document management interface has a Properties view and the DTML Method management interface does not. This is because DTML Documents can have properties like all other Zope objects. DTML Methods share the properties of the folder than contains them.

DTML Methods

DTML Methods in Zope are a way of presenting information in objects. While they are objects, DTML Methods are really more of a “view” of information in an object. DTML Methods are similar to HTML methods in that they are used to modify and display the properties of the objects they are contained.

Creating DTML Methods

To add a DTML Method, select the Contents view of the folder to add the DTML Method. Select "DTML Method" from the Add List. The Add DTML Method screen, shown in Figure 49, allows you to set the `id` and `title`, as well as optionally select a file on a local hard disk to upload. To select a file from the local hard disk, select the "Browse..." button. After choosing the file and clicking "Open", the file name or path will appear in the file are.

Once the Add DTML Method Screen is completed, you have two options: "Add", to add the DTML Method and return to the Contents view of the current folder, and "Add and Edit", to add the DTML Method and go directly to its Edit view. A DTML Method is denoted in the folder Contents view by a square icon with an open diamond.

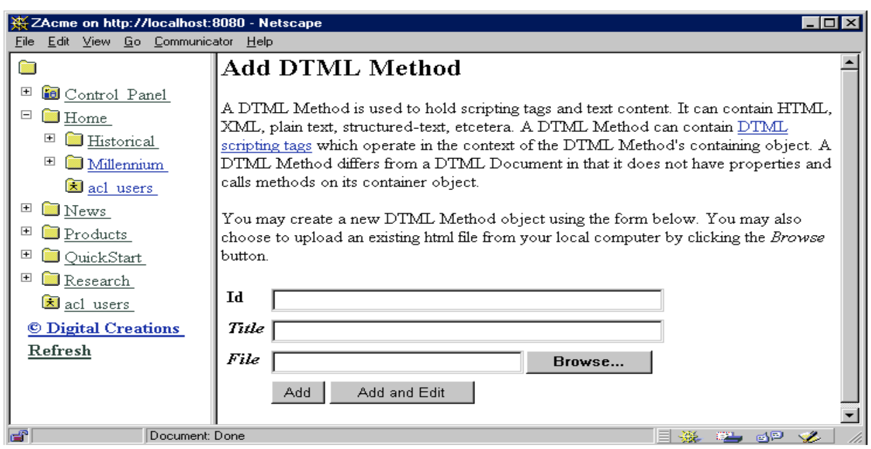

Figure 49. Adding a DTML Method

Editing DTML Methods

To edit a DTML Method, click on its name in the Folder’s Contents List. The *Edit* view is the default view of the DTML Method, as shown below in Figure 50. On the Edit screen there are fields to edit the DTML Method’s title and source code

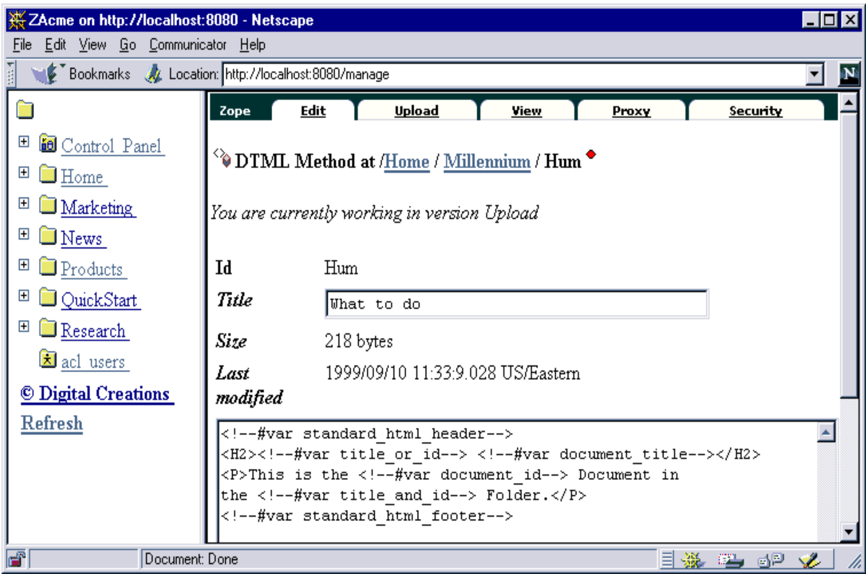

Figure 50. DTML Method editing

(default source code shown here) and to change the size of the source code area. The new DTML syntax uses the format <dtml-var standard_html_footer>. The older DTML syntax<!--#var standard_html_footer--> that is shown in the "Edit DTML Method" is still valid.

The size of the object and the date the DTML Method was last modified is also provided. To commit a change to the DTML Method, simply click "Change". A dialog message appears informing either that the changes have been made, or that there are DTML errors.

New data may be uploaded from a local hard disk. This allows for the use of normal text editors to create DTML Methods. To upload new data, click on the Upload tag, then click the “Browse...” button to select the file to upload, and click “Change.”

Stan wants to experiment with a DTML Method that will provide an user a message if they login the to the site but forgot their password. This DTML Method will email the user their password, if requested by the user. Copy the DTML script shown in Figure 51 into the source code area. There are several properties that have to be established before this method will work in this tutorial. It is only provided here to give you a place to start experimenting on your own.

```
DTML method for mailing back password in case user forgot
<dtml-var standard_html_footer>

<dtml-var standard_html_header>
<dtml-sendmail smtphost="localhost">
From: "Zope.org" <info@zope.org>
To: "<dtml-var full_name>" <<dtml-var email>>
Subject: Membership reminder
Your password: <dtml-var password>
Request made by IP <dtml-var "REQUEST.REMOTE_ADDR"> at <dtml-var
ZopeTime></dtml-sendmail><p>
Your password has been mailed.  It should arrive in your mailbox
momentarily.</p><dtml-var standard_html_footer>
```

Figure 51. DTML Method Mailing Password

In order to see what a DTML Method looks like when it's rendered, click the *View* tab.

DTML Documents

Unlike DTML Methods, Zope DTML Documents are meant to display their own content. No distinction is needed to differentiate a DTML Document's information from its containing objects. DTML Documents can be assigned properties that will allow for categorizing and searching. These properties, along with content are displayed using Document Template Markup Language (DTML) with HTML. DTML tags insert Zope objects and properties into Web pages, along with sequence control, control flow (if else constructs) and more active tags for tree displays and sending email.

Creating DTML Documents

To add a DTML Document, select the Contents view of the folder to add the DTML Document in. Select "DTML Document" from the Add List. The Add DTML Document screen, shown in Figure 52, allows you to set the id and title, as well as optionally select a file on a local hard disk to upload. To select a file from the local hard disk, select the "Browse..." button. After choosing the file and clicking "Open", the file name or path will appear in the file area. Once the Add DTML Document Screen is completed, you have two options: "Add," to add the DTML Document and return to the Contents view of the current folder, and "Add and Edit," to add the DTML Document and go directly to its Edit view. DTML Documents are denoted in the Contents list with a plain document with open <> on the page icon.

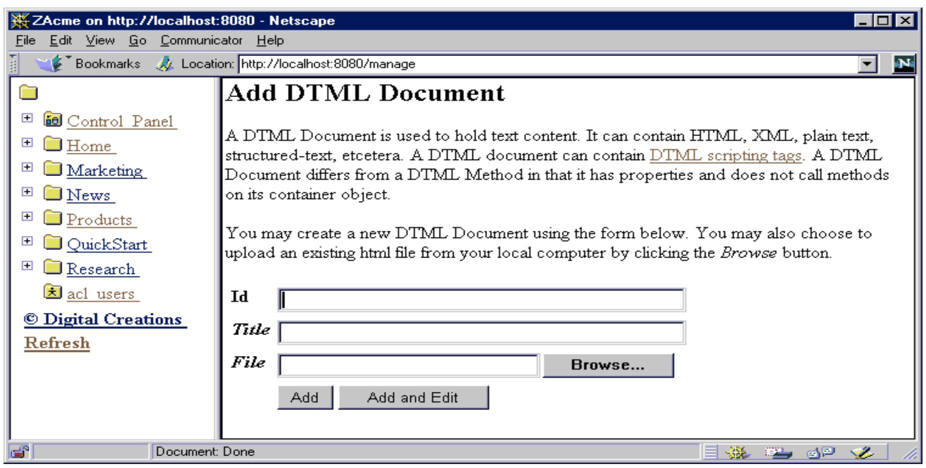

Figure 52. Add DTML Screen

Editing DTML Documents

To edit a DTML Document, click on its id from the Contents list of the folder it is contained in. The Edit view is the default view of the DTML Method, shown in Figure 53. On the Edit screen, there are fields to edit the DTML Document's title and source code (default source code shown here) and to change the size of the source code area. The size of the object and the date the DTML Document was last modified is also provided. To commit a change to the DTML Document, simply click "Change". A dialog screen appears on the content view page informing either that the changes have been successfully made, or that there are DTML errors. .

Viewing DTML Documents

In order to see what a DTML Document looks like when it's rendered, click the *View* tab.

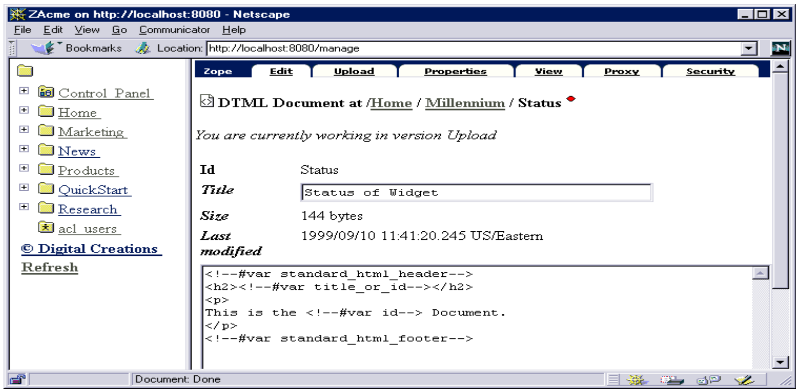

Figure 53. DTML Document Edit screen

Setting and using DTML Document Properties

A distinguishing feature of DTML Documents is that arbitrary properties can be assigned to them. Properties provide identifying information that enables the DTML Document to be categorized or searched. To set properties on a DTML Document, click on the Properties tab to go to the Properties view of that DTML Document. Previously set properties, including title, are displayed first and can be edited or deleted. To edit an existing property, simply change its value in the text area and click "Change". Deleting properties requires that you select the check box of the property or properties to be deleted and click "Delete". To add a new property, set the id, type and value for the new property and click "Add".

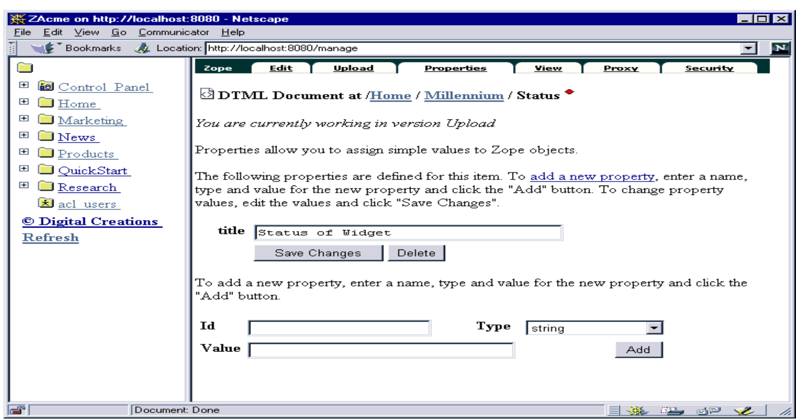

Figure 54. DTML Document Properties view

Uploading new data

New data may be uploaded from a local hard disk. This allows for the use of normal text editors to create DTML Documents. To upload new data, click on the Upload tag, then click the “Browse...” button to select the file to upload, and finally, click “Change.”

External Methods

Basically, an External Method is a way of drawing in a function from an external Python file into Zope's object hierarchy. Zope Folders can hold different types of behavior objects or "methods." External Methods are one of a number of ways to create methods. Other method types include DTML Methods, and SQL Methods.

In programmer's terminology, a method is a function that is bound to an object. This is exactly how External Methods work. External Methods are Python functions which are not stored in Zope's database, but are defined in external files. When you create an External Method object with Zope's management interface, you bind that external function to the current Folder. Since the external function is bound to the Folder, it can be called a *method*.

Creating External Methods

To use an External Method, you need to place your Python source code file in the *Extensions* directory in of your Zope directory (you may need to create this directory), or in an Extensions directory inside a Product directory, e.g. *lib/python/Products/MyProduct/Extensions*.

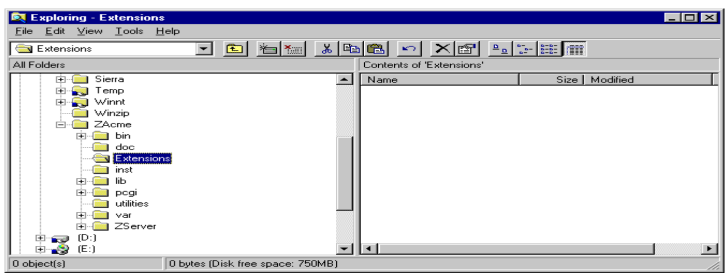

Figure 55. Extension directory

Next, create an External Method with Zope's management interface. Choose "External Method" from the list of addable objects. Then specify your function name and the name of your source file (without suffix). Voila, your Python code runs in Zope without having to master writing a Zope Product from scratch!   Stan creates the *text.py* file under the *Extensions* directory, just copy and paste from the example the function

External Methods need not actually be bound to the current Folder, but can operate as a function if you wish. Figure 57 provides two examples showing the difference between a function, *KnowNothing* and a method, *FolderTitle*.

When adding external methods, you will have to option of trying out your code before implementing the code. The "Try It" tag, allows you to experiment with your python program. Of course the powerful, "Undo" option can wipe away your mistakes with a single key stroke.

Working with External Methods

External Methods operate like any normal method, which allows you to manipulate them in many neat ways.

You can call your External Methods from within DTML, like normal methods. You can also publish External Methods through the Web, like normal methods. But why stop there, you can call your External Methods within Python code in other External Methods. The possibilities are endless.

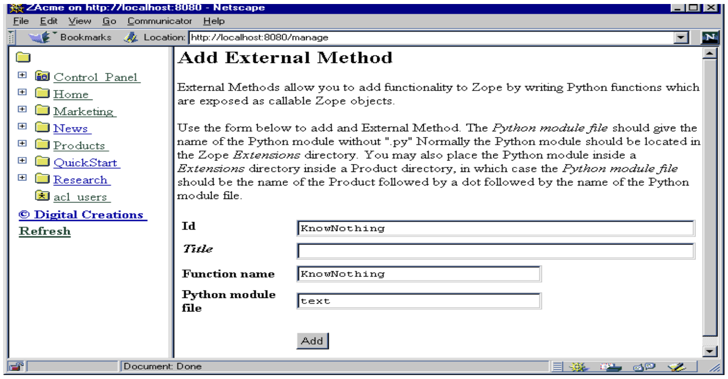

Figure 56. Add External Method

```
# in Extensions/text.py

# this is a normal function
#
def KnowNothing():
    "Unbound function"
    return "I know nothing of Folders and such."

# this is a method of a folder object
#
def FolderTitle(self):
    "self is bound to the current folder."
    return self.title or id()
```

Figure 57. Example showing the difference between a function and a method

For example, Stan defined the *KnowNothing* method (from the example above) inside a Folder called *Home*. He can access the method by going to this URL:

Home/KnowNothing

This would display “I know nothing of Folders and such.” in the browser window.

We could also create a DTML Method inside the *Home* Folder and access the *KnowNothing* method with:

`<dtml-var KnowNothing>`

This would cause the *"know nothing"* phrase to be inserted into the DTML Method.

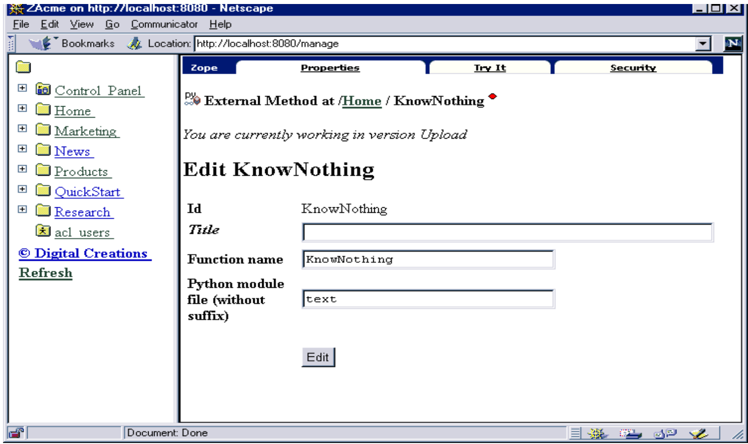

Figure 58. External Methods Try It

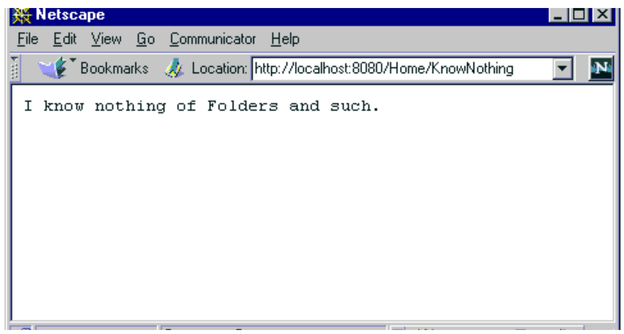

Figure 59.

Acquisition and External Methods

External Methods are acquired by sub-objects, in exactly same way as all other Zope objects. So, if you define an External Method in a Folder, it will be available in all sub-Folders. When a sub-Folder acquires an External Method, the method will be bound to the sub-Folder.

Importing Modules and Packages

External Methods may access Zope packages and modules, and/or other Python packages. So if you have a cool Python package you want to use with Zope, the easiest thing to do is to write an External Method. For example:

```
import MyCoolPackage


def MyObject(self):
    "use a cool object"
```

```
myObject=MyCoolPackage.MyCoolClass()
return myObject.myMethod()
```

This example uses a *MyObject* instance whose class is defined in *MyCoolPackage*. You may need to manipulate the Python Path to get access to your Python packages.

Zope provides a convenient solution for packaging classes that you wish to use in External Methods. This solution is the *Shared* package, located in *lib/python/Shared*. To use this Shared package, create a sub-package inside Shared package with the name of your organization (or personal name). Then create further sub-packages if you wish. For example, instead of manipulating the Python path to allow you to import your *MyCoolPackage*, you could simply relocate that package to the Shared package, *lib/python/Shared/MyName/MyCoolPackge*, where *MyName* is your organization or personal name. Now Zope has access to your package.

External Method Limitations

External Methods can cause a lot of trouble if you aren't careful. Since External Methods have full access to Zope internals you might just shoot yourself in the foot.

One important gotcha to keep in mind is that the Python modules in which External Methods are defined are not imported into Zope in the way Python modules are normally imported. Therefore, you cannot define classes or objects in your External Methods and expect to be able to assign them to attributes of Zope objects. This is because Python's pickle mechanism will not be able to reconstitute these objects from the object database, since it will not be able to find their class definitions. If you want to add your own objects to the Zope hierarchy one good way to do it is to make sure their classes are defined outside the External Method, for example, in the Shared package.

This limitation doesn't prevent you from creating and modifying attributes of Zope objects. It simply limits the sort of changes you can make. Stan is well aware he has to do more research on Python before he tackles these Zope options.

Saving or discarding Version changes

Stan wants to save his experiments. To save or discard changes made in a Version, open the Version by clicking on its id or icon in the folder contents list.The Join/Leave screen will contain additional forms and buttons.

One of these forms allows comments about the version changes to be entered and then saved. To save changes, click on the “Save” button. To discard changes made and return to the state the changed objects were in, click the “Discard” button. Changes may be saved or discarded without having to leave a Version. This allows incremental changes to be committed without unnecessary steps.

Stan returns to the Version "Upload New Doc" that was created at the start of the exercise. Access the "Save/Discard" view. In the comment section, Stan enters `Upload Python function and Added Users` to the screen and presses the "Save" button. All of the red diamonds will disappear, and all folders, users, methods and documents created during the version will be saved as well. This is a powerful function of Zope which allows change to be made while site utilization continues.

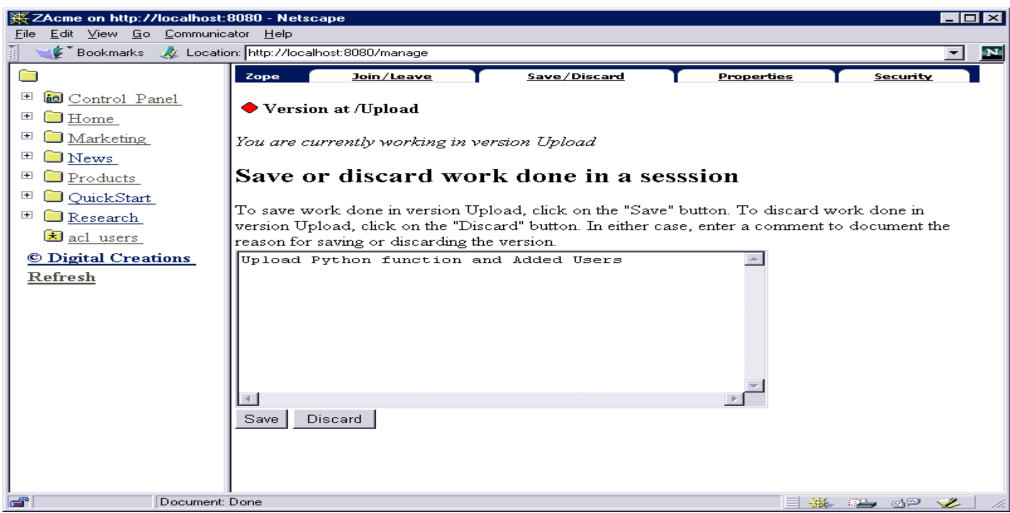

Figure 60. Save Work in Version

Zope Management

Management of your database and system shutdown is available on the Control Panel. We have already explored the Product Management section of the Control Panel. The Database Management options include packing the database, cache management, and garbage collection.

Control Panel

The control panel contains the mechanisms to create products, factories and product distributions. Along with product management, the control panel also provides database management. The default view of the control panel is the *Contents view*, as shown in Figure 61. The process ID of Zope, as well as the length of time Zope has been running are given within the text. On some platforms, the Contents view contains an option to shutdown the Zope process for system maintenance or installation. However, on platforms where process shutdown is controlled within another application, this “Shutdown” button does not exist.

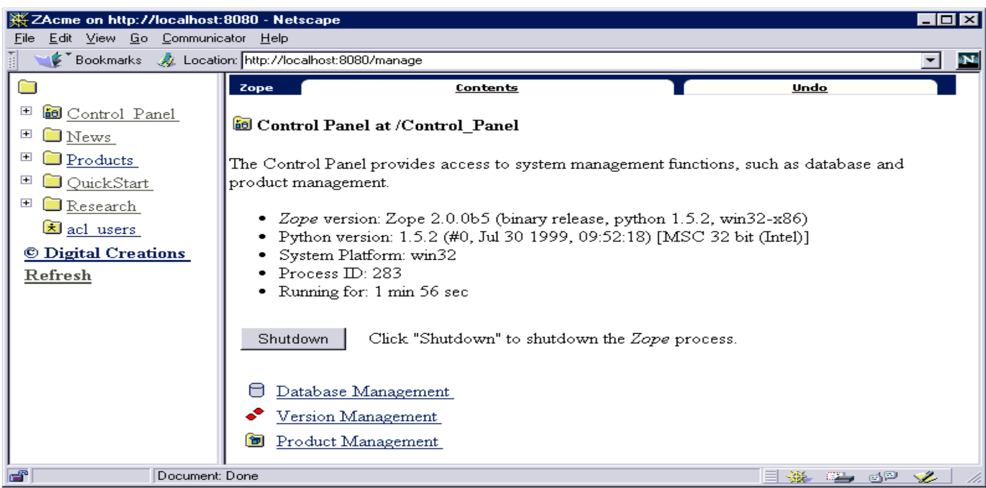

Figure 61. Contents view of control panel

Database Management

When “Database Management” is selected from the Contents view, the *Database* view is presented, as in Figure 62. This view supplies database status information, such as the database size and location within your system. The “Pack” option allows you to remove object revisions that are older than the entered number of days.

Cache Management

The database used by Zope has an object cache that helps to manage movement of objects to and from the database. When an object is retrieved from the database, it is copied to the cache. Subsequent retrievals can be performed very quickly from the cache. The cache also provides for removing unneeded objects from memory. This is performed through cache garbage collection.

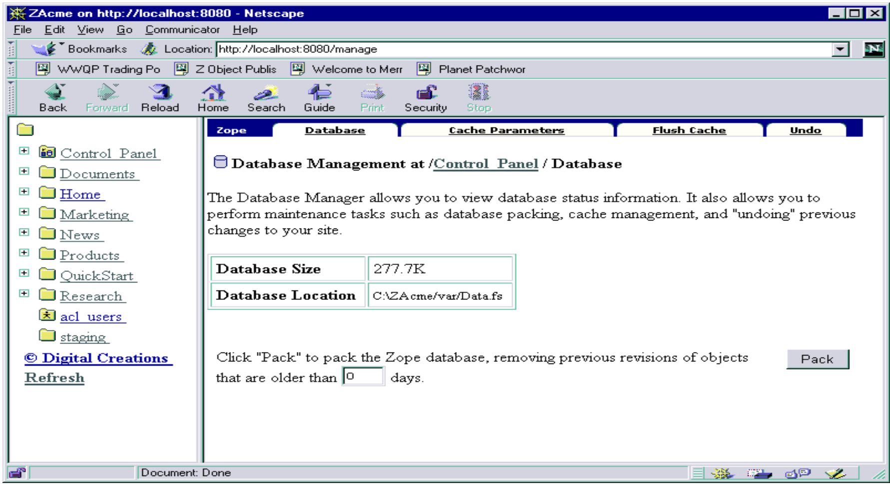

Figure 62. Database Management View

Garbage collection is performed by scanning objects in the database cache. Objects that have not been accessed in a long time are deactivated. Deactivating an object causes all of its data to be freed from memory. The object is placed in a deactivated state. Its data will be reloaded as necessary the next time the object is used. Objects that are only referenced by the cache are removed from the cache and memory. Deactivating an object often makes it possible to deallocate other objects.

Cache garbage collection is performed automatically when the database cache size exceeds a target size set in the Cache Parameters view. Automatic garbage collection is performed incrementally to avoid significant impacts on performance.

Manual garbage collection can also be used to force either one entire scan through the cache, or repeated scans until the cache size can't be made smaller.

Cache Parameters

The *Cache Parameters view*, shown in Figure 63, gives the cache parameters and statistics (Table 1).

Flush Cache

The *Flush Cache view*, shown in Figure 64, enables you to manually collect cache garbage. "Full sweep" allows you to make a single pass through the cache, removing any unreferenced objects, and deactivating objects that have not been accessed in the number of seconds given in the input box. "Minimize" allows you make multiple passes through the cache, removing any unreferenced objects, and deactivating objects that have not been accessed in the number of seconds given in the input box. After selecting either "Full Sweep" or "Minimize," the new cache statistics are given.

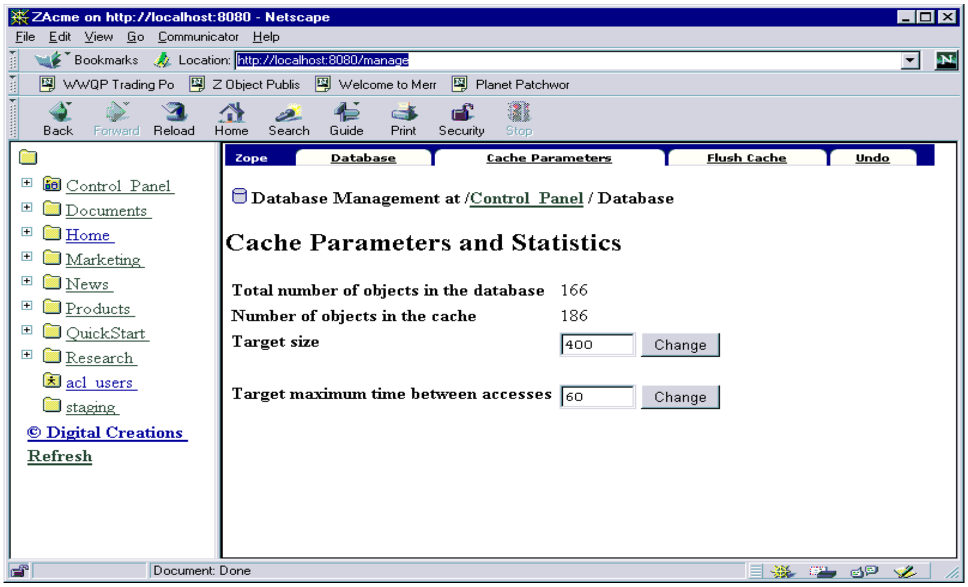

Figure 63. Cache Parameters view

Parameter or Statistic	Description
Total number of objects in the database	This is an approximate number of objects in the system.
Number of objects in the cache	This is the number of objects in memory. Some of these objects may be inactive, meaning that their state is not in memory.
Target size	This parameter sets a target for the cache manager. If the number of objects in the cache is less than this target, then no attempt is made to remove objects from the cache. When the number of objects in the cache exceeds this target, then objects are scanned periodically to see if they should be deactivated or deallocated.
Mean time since last access	For the objects that are scanned when trying to reduce the cache size to the target, this is the time that has passed since the object was last accessed.
Target maximum time between accesses	When scanning objects to reduce the cache size, objects that haven't been accessed in a span of time that exceeds this amount are deactivated.
Deactivation rate	This is the rate at which objects are deactivated.
Deallocation rate	This is the rate at which objects are removed from the cache and memory.
Time of last cache garbage collection	This is the most recent time that objects were scanned in an attempt to remove objects from the cache.

TABLE 1. Cache parameters and statistics

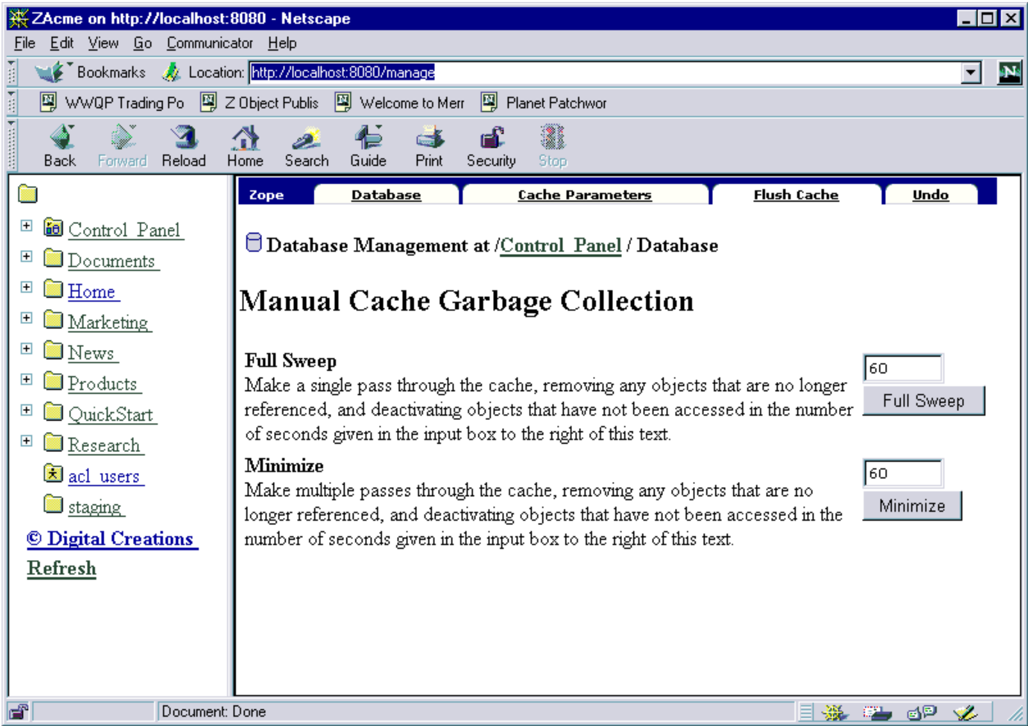

Figure 64. Flush Cache view

Quick Start Tutorial Summary

Zope provides an utility to manage diverse information through a web server. The nature of the world wide web is the centralization of information while management can be accomplished from a variety of places. You can delegate control, provide department customizations, publish various objects online, and manage all from a central location.

Stan is satisfied that he has a working knowledge of Zope. A more detailed description of Zope object reference is available in the **Zope Content Manager's Reference**. Stan has created the public website for **ZAcme, Incorporated** to give the public online information on its products, research, and news. Stan delegated responsibility for content to the departments themselves. The Zope site is online and functional. The Public Relations, Research and Products department have their own web pages to supply company information.

Index

A
acquisition 14, 16, 18, 21
Add File 48
Add Folder 8
Add Image 51
Add List 10
authentication 15
authorization 15, 18

C
Cache Parameters view 69
configuration interface 27
configuration objects 39
Contents 68
Contents view 8, 46

D
Database view 68
date-time value 13–??, 14
Deactivate 69
distribution file 39
distribution version 39
Distribution view 39
distributions 39
Document Template Markup Language 12
DTML 12
DTML Method 10
DTML Methods 47
DTML var tag 12

E
Edit view 58
Extensions directory 63
External Method 63

F
Factories 35
Figure 27 33
File objects 47
Flush Cache view 69
Flush Cashe view 69
Folder actions 8
Folder Items 8

G
Garbage collection 69

H
HTTP Basic Authentication 54

I
Id 8
image 10
index_html 9, 23

J
Join/Leave screen 43, 44

M
management screen 8
Minimize 69

N
navigation frame 8

O
Object List 46
object tabs 8

P
permissions 16
Plutonia Incorporated 7
Principia product 33
Products Management 33
Properties 13, 44, 46
  Types 46
properties 27
Public Interface 9, 45

R
roles 16, 17
  Anonymous 18, 20
  Local roles 19
  Manager 18, 20
  Owner 20

S
security 15, 21
Security tab 21
Session 44, 67
Shared package 66

T
template 27

U
User Folder 9, 17, 52, 53
User objects 53
users 16

V
Version 43

W
workspace 8
workspace views 8