# Zope Developer's Guide

*Document revision 1.2,*
*Modified September 24, 1999 by Amos Lattieir*

*For Zope version 2.0*

# Zope Developer's Guide 1

Typographical Conventions .................... v

Introduction .................... 1

Extending Zope 1

Zope development options 1

Z Classes .................... 3

Introduction to Z Classes 3

What are Z Classes? 3

Necessary Background 3

Getting Started with Z Classes 3

Overview 3

Basic Steps 4

Creating a Simple Z Class 4

Creating a Property Sheet 5

Creating a View 5

Creating Z Class Instances 6

Using Z Class Instances 6

Z Classes in Depth 7

Basic Z Class Properties 7

Z Classes and Inheritance 7

Class ids 8

Creating Methods 8

Creating Property Sheets 8

Creating Views 9

Creating and Managing Permissions 9

Subobjects 10

Classes inside Classes 11

Factories, add methods and Wizards 11

Subclassing from Custom Python Classes 11

# Typographical Conventions

The following typographical conventions are used in this manual:

- *Italic Font* is used to indicate object names, key concepts and URLs.
- `Courier Font` indicates source code and user defined information.

# Introduction

**Welcome to Zope, the premier platform for developing dynamic web application.**

### Audience: Developers

.To use the Zope Developer's Guide you need to have a working knowledge of Python and Internet programming. You should also be familiar with Zope from a content manager's point of view.

## Extending Zope

One of Zope's greatest features is the fact that it is released a open source software. Zope's complete source code is available for you to peruse and extend. Zope is mostly written in Python which makes it fairly easy to understand and change. However, Zope is a large system which performs many functions and includes many different components. It can be difficult to get a handle on how it works without some guidance. This guide is meant to provide you with the backaground, examples, and technical information you need to understand how Zope works and extend it to suit your needs.

## Zope development options

You don't need to leap into modifying Zope's source code right away. In fact, Zope can be extended on many levels, and you may not need to even write any code to modify Zope to meet your needs. Here's some of the ways you can develop with Zope:

- You can script Zope using Document Template Markup Language (DTML) in DTML Document and DTML Method objects. This gives you limited access to Zope internal in an easy to use form.
- You can write Python methods and import them into Zope using External Method objects. This allows you to add complex Python code to your Zope objects.
- You can use Z Classes to create new types of Zope objects through the web. By creating your own types of Zope objects you can tailor Zope to your needs.
- You can write new types of Zope objects in Python. This option provides the same benefits as Z Classes but in pure Python. You may wish to combine this method with Z Classes.
- You can use Zope source packages in your own Python programs. This allows you to publish objects without the Zope mangement framework, or you may wish to use other Zope components to provide object persistence, or text generation capabilities to your Python programs.
- You can script remote Zope installations with the ZPublisher.Client package. This allows you to connect to remote Zope objects as though they were local objects.
- You modify the Zope sources to make Zope itself function differerently. If you don't like how Zope does something, you can change it.
- You can extend the Zope sources to add new functionality to Zope. You can extend Zope's reach by building new Zope packages which you may choose to share with the Zope community.

Your choices are many, and in fact you will probably find yourself combining several of these approaches.

This Guide will cover these development options in different chapters. While it is not essential to read all the chapters of this guide in order, the material is arranged so that reading earlier chapter should help you with understanding later chapters.

# Z Classes

## Introduction to Z Classes

### What are Z Classes?

Z Classes herald a new era in Zope development. Z Classes allow you to build your own types of objects through the web. In effect, instead of being limited to existing objects like *Folders* and *DTML Documents*, when building your site, you can now define your own classes of objects. For example, if your site includes things like employee manuals, bug reports, shopping malls, etc., then you can actually create Employee Manual objects, and Bug Report objects, and all the rest. In short, Z Classes let you create and store “your kinds of things” instead of Zope things.

You define your classes in the Control Panel, and then you can create and manipulate your objects just as you create and manipulate *Folders, DTML Documents*, and all the other standard Zope objects. As you change and update your classes, all the existing instances of those classes are updated. So if you add a new method to your Employee Manual class, all your Employee Manual objects will be updated.

You can even export your Z Classes and use them in different Zope installation, thus allowing you to leverage your work across Zope installations.

Basically Zope classes make extending Zope much easier. In fact there is no Python programming required. This is a big change from the days of using the Product API and Python to extend Zope. Now building new classes is easy enough that you can justify making several specialized classes to solve specific problems.

### Necessary Background

To start with you should be familiar with creating *Products* though the *Zope Control Panel*. This process is outlined in the *Zope Content Manager’s Guide*.

Working with Z Classes builds on Zope’s existing Product Creation facilities but extends it in important ways. The basic difference between creating traditional Products through the Control Panel and creating Z Class Products is that with classes your objects retain a connection to their classes. So when you make changes to your class, your instances are updated. With traditional Control Panel Products no such link between the Control Panel and the created object is retained, the Factory simply creates an object in the Zope object hierarchy.

## Getting Started with Z Classes

### Overview

Let’s begin using Z Classes by building a simple example class that represent a music CD.

In this scenario, we want to let people fill out forms that collect information about CDs in their music collection. To create new “records”, people just choose your newly-defined “thing” from the kinds of “things” to add to their folder.

## Basic Steps

Here are the basic steps required to build a simple Zope Product with Z Classes.

1. Create a Z Class.
2. Provide some information about the Z Class.
3. Define a Property Sheet for the Z Class.
4. Create a View.

## Creating a Simple Z Class

Start by going to the *Control_Panel/Products* Folder and creating a new Product. Click “Add” and specify “CDProduct” for the Product Id. Then click “Generate”. You’ve now created a new Zope Product. Your product should be visible inside the *Control_Panel/Products* Folder.

Now go the “CDProduct” Product Folder and create a Z Class by selecting “Z Class” from the product add list. For the class Id specify “CDClass”, and for the “Meta type” specify “CD”, and make sure “Create constuctor objects.” is checked. Don't worry about the “Base classes” widget for now. Click “Add”, and you've just created your first Z Class.

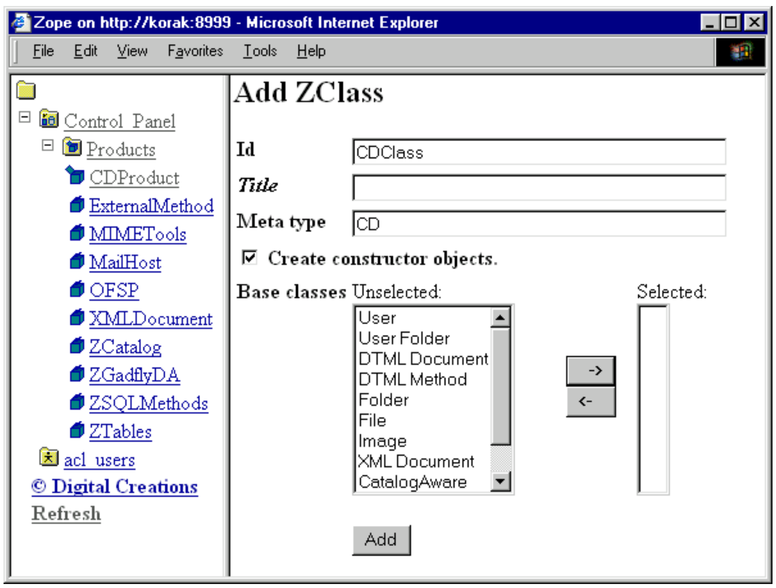


**Figure 1.** Z Class add form

Now let’s set up our newly minted CD Class. You’ll notice that after you add your Z Class, Zope creates five new objects for you:

- A Z Class called “CDClass” which is our Z Class,
- a DTML Method called “CDClass_add”,
- a DTML Method called “CDClass_addForm”,
- a Zope Permission called "CDClass_add_permission",
- and a Factory called “CDClass_factory”.

The constructor objects are created by Zope to help you create new instances of your class.

Let's look at the Z Class object. Click on the "CDClass" object inside the "CDProduct" Folder. You are presented with the "Methods" Z Class management screen. Right now our Z Class has no methods, so no method objects are listed on this screen. Now click on the "Basic" management tab. Here we can set some Z Class properties, like a class's "meta type" and icon. A "meta type" is the name of your class as it appears on the product add list. Your class's "meta type" should be "CD". The class icon can be defined here by uploading a file. You'll also notice a class id which you can safely ignore for now.

## Creating a Property Sheet

Now let's give our class some properties. We can do this by creating one or more Property Sheets for our class. A Property Sheet is a schema that describes what kind of properties instances of our class will have. To create a Property Sheet click on the "Property Sheets" tab. This takes you to the Property Sheets management screen.

Create a Property Sheet by clicking "Add" on the Property Sheets management screen. Then specify "cd_info" as the Id of the Property Sheet. Now you should have a new Property Sheet object listed on the Property Sheets management screen. Click on "cd_info" to edit your Property Sheet.

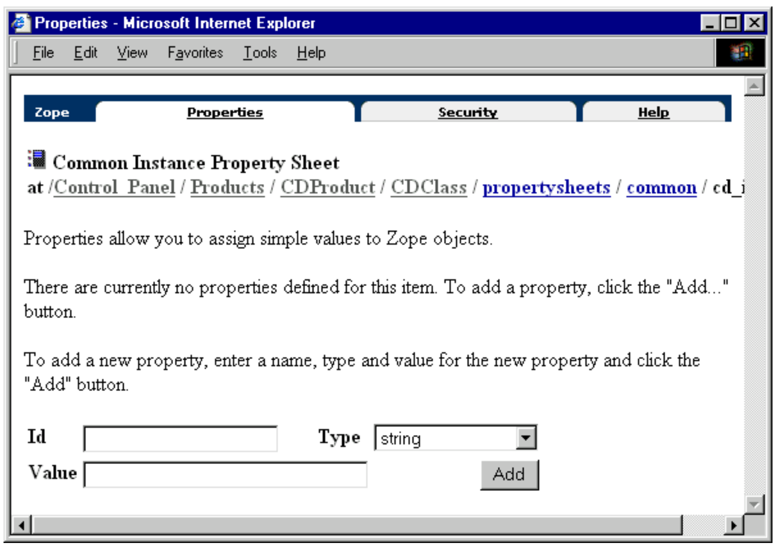


**Figure 2.** Property Sheet management screen.

Now you should see a familiar Zope property setting management screen. It's important to remember that right now we are not setting the properties of an individual CD, but we are defining properties that all CD instances will have. Create a "title" and "artist" property, both of type string. Then create a text property named "description".

## Creating a View

Now let's add a View to our class which will let use manage these properties. A View is a management screen which is available via a management tab. Click the "Views" tab to edit a class's views. Now you should see that our class currently has only one View, "Security". This view calls the class's *manage_access* method.

Create a new view named “Properties” which calls the class’s *propertysheets/cd_info/manage* method. Fill out the “New” form and then click “Add”. Now you should see that your class has two Views. Let’s make the “Properties” View the first or default view by checking the checkbox next to it and clicking “First”. Now “Properties” should be at the top of the list of Views.

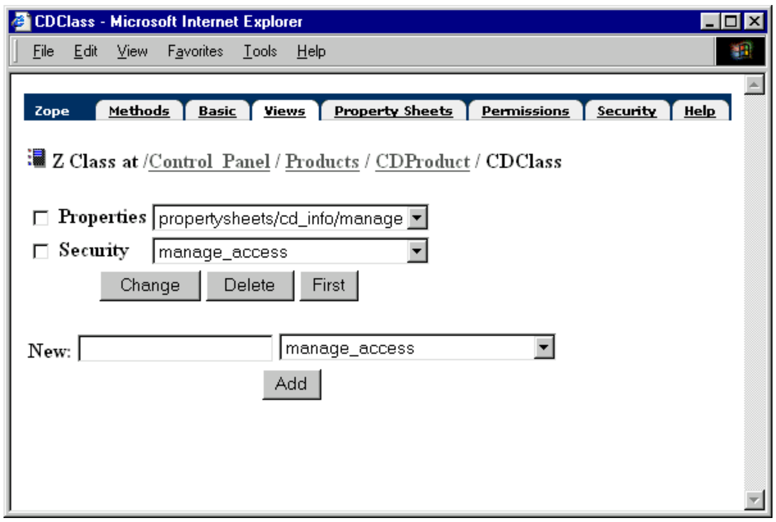


**Figure 3.** View management screen after you add a “Properties” view.

## Creating Z Class Instances

Now that you’ve fleshed out your class, it’s time to create an instance. To have your object appear in the Zope product add list you must create a Factory along with an add form and an add method. Recall that Zope already created these objects for us when we first created our Z Class. Later we may wish to go back and customize these objects, but they’ll do for now.

Go to the top-level Zope Folder and choose “CD” from the product add list. You should now be prompted for an `Id`. Call it “myCD” and click “Add”. Congratulations, you’ve created your first instance of the “CDClass” Z Class.

## Using Z Class Instances

To find out more about our class, let’s explore an instance.

Click on the “myCD” CD object you just created. You'll note that you can edit the CD’s properties with the “Properties” tab, and can edit its security settings with the “Security” tab. Through the management interface you have access to the two management tabs that you defined as “Views” of the Z Class.

As an experiment go back to your Z Class and add another property to the “cd_info” Property Sheet. For example, you could add a "genre" property to your property sheet. Now go back to your CD object, “myCD”. You should notice that the new property is now available via the “Properties” management tab. This is the magic of classes. When you change a class, all the instances of the class are effected.

In addition to being able to manage your class instances through the web, you can also access them through DTML. For example, suppose you created a CD object called “SuperOldies_vol42”. In a DTML Document you can query this object like so:

```
<p>Information about a CD:</p>
<dtml-with SuperOldies_vol42>
<p>Title: <dtml-var title></p>
<p>Artist: <dtml-var artist></p>
<p>Description: <dtml-var description html_quote newline_to_br></p>
</dtml-with>
```

In other words, your CD object is a normal Zope object and you can access its properties just like you access a Folder’s properties. You can also manipulate its properties in the same way you manipulate the properties of other Zope objects. For example, here’s a DTML fragment that changes some properties of our CD object:

```
<dtml-call "SuperOldies_vol42.manage_changeProperties({
  'title'       : 'Super Oldies volume 42',
  'artist'      : 'Various Artists',
  'description' : 'Greatest hits of many has beens.'
  })">
```

Instances of Z Classes inherit all the methods of their base classes. In our simple example, our CD class inherits the basic methods available to all simple Zope Products automatically without having to specify this. In technical terms, all Z Classes inherit from *SimpleItem.Item*.

# Z Classes in Depth

Now that we have a basic example of a working Z Class, let’s return to some of the details that we glossed over creating our example CD class.

## Basic Z Class Properties

On the “Basic” Z Class management screen you can set the class’s icon in addition to defining a Z Class’s “meta type”. However, you cannot change a Z Class’s base class or base classes.

A Z Class icon is used in the Zope management interface to identify instances of the Z Class. You should use a GIF file for your icon which is 16 by 16 pixels. It should have a transparent background. You can set a class’s icon by uploading the icon file. This will create the icon and define the icon’s URL. You can change the icon by uploading new icon files. If you need to, you can change the URL of the icon, but in general this shouldn’t be necessary.

## Z Classes and Inheritance

When you first create a Z Class you are presented with a complex widget which allows you to specify your class’s base classes. Your Z Class inherits the methods and attributes of all its base classes. If you do not specify any base classes your class will still inherit basic Zope features which are defined by the *OFS.SimpleItem.Item* class. This is generally what you want, and this allows instances of your class to perform normal Zope functions such as properties management. If you want your class to be able to contain other objects, like a Folder does, then you'll have to subclass “Object Manager”. This will provide you with all the standard methods to support object containment. Other base classes may be available depending on what Zope extensions you have installed.

In addition to subclassing existing classes, you can also subclass other Z Classes. This is a very powerful feature. It allows you to build complex and related classes.

Another thing to notice is that you can subclass more than one base class. This is one way in which Python differs from many other object-oriented languages. When you subclass more than one class, the order of the base classes is important, because it determines the order in which classes are searched when acquiring method and attributes. To control the order of base classes when creating a Z Class move the base classes back and forth between the unselected and selected boxes until you have them in the correct order.

## Class ids

All Z Classes have class ids which are editable from the "Basic" management screen. Zope will assign a unique class id to your Z Class. Under normal circumstances you will not have to change this id. However, this class id is important to Zope, as it is how Zope identifies what Z Classes go with what Z Class instances. About the only time you'll care about class ids is if you want to replace one Z Class with another and have all existing Z Class instances recognize the new Z Class. In this case you'll have to make sure that the new Z Class has the same class id as the one it is replacing.

## Creating Methods

A class's heavy lifting is done by its methods. In addition to defining a property schema, your main task in crafting a Z Class will be to build a collection of methods which manipulate your class's properties and do useful work.

To create methods for your class, you simply create objects in the "Methods" tab management view. For the most part your methods will consist of DTML Method objects, SQL Method objects and External Method objects. These Zope objects will provide all instances of your class with services just as they would provide services to their enclosing Folders if they were not defined as Z Class methods. In other words, there is nothing new or unusual about method objects; they are simply made available to class instances through the Z Class rather than through acquisition.

One useful method to create is the special *index_html* method which is used to display an object when called from the web. Just as Folders manage their default view with an "index_html" object, so you can control the default representation of your class. For example our CD class might define a DTML Method with Id "index_html" like so:

```
<dtml-var standard_html_header>
<h1>CD</h1>
<p>Title: <dtml-var title></p>
<p>Artist: <dtml-var artist></p>
<p>Description: <dtml-var description html_quote newline_to_br></p>
<dtml-var standard_html_footer>
```

Methods which you wish to be accessible through the web and DTML, need to be mapped to permissions. See "Creating and Managing Permissions" on page 9.

## Creating Property Sheets

We've already covered most of what is involved in creating Property Sheets. To recap, Property Sheets allow Z Classes to maintain simple typed properties. Property Sheets are managed via the "Property Sheets" management tab. One common use for Property Sheets is to create management Views for Z Class instances.

Right now there is only one accessible form of Property Sheet, the "common instance property sheet". This type of Property Sheet stores its properties as attributes of Z Class instances. So for example if an instance has a property named "title" defined in a common instance property sheet named "stuff", setting this property will effect the "title" attribute of the instance. This is what you would normally expect of properties.

Property Sheets also allow more indirect access to properties via the *propertysheets* attribute. So in our above example we could access the “title” property via *propertysheets.stuff.title* attribute in addition to simply the *title* attribute. The sort of access is handy in the case where two different property sheets define properties with the same name.

Like Z Class methods, Property Sheets are mapped to class permissions in order to define security settings for a Z Class. See “Creating and Managing Permissions” on page 9.

## Creating Views

We've already covered almost all there is to creating Views. Views define management screens which are accessible via tabs form the Zope management screen. Views make it easy to make your Z Class instances manageable through the web. The main thing to add over our previous discussion is that by default your Z Class starts with the same views as its subclasses.

In addition, Views are only visible from the management interface when the methods associated with the Views are mapped to permissions that the user is entitled to access. In this way users with different levels of access are presented with different management Views.

## Creating and Managing Permissions

Zope manages security though a system of permissions and roles. Users are assigned roles and objects have permissions. Through the management interface you can bind permissions to roles for specific objects. See the *Zope Content Manager’s Guide* for more information about security.

For our purposes, you should understand that a class’s permissions define collections of methods which are grouped together in functional groups. In other words, you should think of you class as providing a number of discrete services. For each service you should provide a permission which controls access to the service.

Using Z Classes your job is to define permissions for your class so that access controls can be set on instances of your class.

The first step in defining your class’s permissions should be to create methods and Property Sheets for your class. After your class can do something, then you should decide how to describe what it can do in terms of permissions. You must figure out if your methods should be covered by existing permissions of whether it would be better to create new permissions. For example if your class has an “index_html” method, this method is used to view instances of the class, so you can probably use the existing “View” permission with this method. If your class defines a number of methods for accessing remote servers, you might want to create a new permission for these methods called something like “Access remote server”. A common Zope convention is to have a permission for adding objects of a given class and another for general managing of these objects. If your class’s meta_type is "My Object" then these permissions would be named "Add My Objects" and "Change My Objects".

To control what permission a given method is bound to, click the “Define Permissions” tab while managing the class’s permission. You will be presented with a form which maps the method’s permissions to the class’s permissions. This looks complex but in general you will only want to bind one permission of the method to a permission. In the case of DTML Methods this will be the “View” method. The reason for this is that you really only want instances of your class to be able to “View” a DTML Method. An instance should not be able to perform other actions on its methods such as editing them.

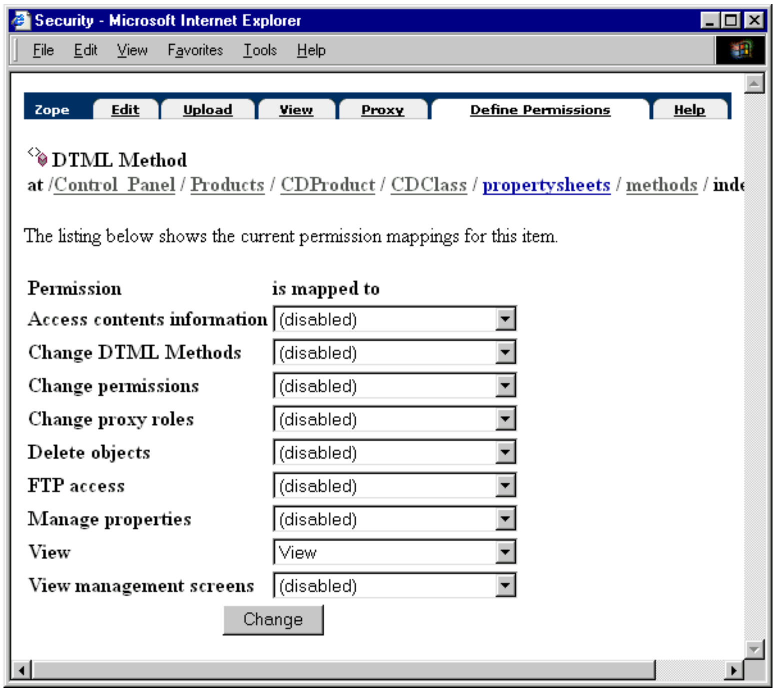


**Figure 4.** “Define Permissions” management screen for a Z Class method.

Property Sheets can be mapped to class permissions in exactly the same way as Methods are mapped to class permissions. While editing a Property Sheet click the “Define Permissions” tab. You will be presented with permission mapping management screen. In general you will want to map all Property Sheet permissions to identical class permissions in order to make your Property Sheets fully accessible from class instances.

To create entirely new permissions you should add Zope Permission objects to your Product folder. When creating new permissions you should keep in mind that the permission names need to be unique. In general you should limit the number of new permissions you create to the smallest reasonable number. By doing so you make it easier for managers to control the security settings of instances of your class.

You can control which permissions are available to your Z Class by going to the “Permissions” management tab. Your class inherits permissions from its base classes, so it may not be necessary to add new permissions.

Additionally Z Classes have a "Define Permissions" view which allows you to map permissions to operations of the Z Class itself. In general the only operation you should expose is the "Create class instances" operation which should be bound to a permission which is used for initialization. Zope will normally handle this detail for you, for example, in the CD class example, Zope automatically creates a "Add CDs" permission and maps "Create class instances" to this permission.

## Subobjects

If you create an a Z Class which is a subclass of “Object Manager” you’ll be able to control what types of objects it can contain. The task is performed from the “Subobjects” management screen. From this management view you can select object types which can be added to instances of your class as subobjects.

If the “Objects should appear in folder lists” option is checked then instances of the class will appear expandable in the left frame of the Zope management interface. In general this option is appropriate when you wish to make your object act like a normal Zope container.

## Classes inside Classes

A common scenario is to create a Z Class which inherits from ObjectManager, and then to create several classes which are meant to be used exclusively inside the container class. An example might be a CDLibrary class and a CD class. CD objects should never appear outside CDLibrary objects.

A simple way to accomplish this sort of thing is to create classes inside other classes. The contained classes should be created inside the “Methods” view of the containing class. Zope will automatically add the contained classes to the list of sub-objects of the container class.

Another reason to create classes inside classes is to avoid having the inner classes appear on the global product add list. Only classes with associated Factories which are created directly inside product Folders will appear on the global product add list.

## Factories, add methods and Wizards

To make a Product appear in the global product add list you need to create a Factory for it. This process is covered in the *Zope Content Manager’s Guide*, so it will only be briefly explained here. A Factory binds a “meta type” to an add form and an add method. Basically the process is this: If the user has adequate permissions, the “meta type” appears as a choice in the product add list. When it is chosen, the add form object is called. The add form object collects information needed to initialize the object and calls the add method which initializes a new object and stuffs it in its container object.

Z Classes make the chore of creating Factories and add methods easier by allowing you to create them automatically when you create your Z Class. Simply supply a “meta type” and check the “Create constuctor objects.” check box when creating your Z Class. Zope will then create the necessary objects for you and your class will be accessible from the global product add list.

## Subclassing from Custom Python Classes

One very useful technique is to create a Python class which is subclassable by a Z Class. This way you can write your application logic in Python, and your user interface in Zope!

To create a Python class which is subclassable by a Z Class you need to create a very simple Zope Product in Python. Your Product should consist of a Python package installed inside the *lib/python/Products* directory. In the package’s *__init__.py* file you need to register your Python class as subclassable by a Z Class.

To accomplish this simply create an *initialize* function in the package’s *__init__.py* file like this:

```
import MyModule

def initialize(context):
  """
  This function is called by Zope to initialize a Product. See
  lib/python/App/ProductContext for more information about
  Product initialization.
  """
  context.registerBaseClass(MyModule.MyClass)
```

Now your Python class is ready to be used as a Z Class subclass. You should note that your subclass's `__init__` method will not be called when instances of the Z Class are created. You should also note that your subclass must obey the ZODB persistence rules, which basically state that mutable sub-objects should be treated immutably. For example,

```
self.bird='parrot' # OK

self.map['bird']='parrot' # NOT OK, will not trigger persistence machinery

m=self.map
m['bird']='parrot'
self.map=m # OK, treats a mutable object immutably
```

You should probably regard this kind of subclass as a sort of mix-in class. It's probably best to leave most of the Zope framework-specific methods to the Z Class and to concentrate on application logic in your Python class.

**A**
acquisition 8

**B**
base class 7
Basic Steps 4
Basic Z Class Properties 7

**C**
CDClass_add 4
Classes inside Classes 11
Control Panel 3
Create Factory and add methods 4, 11
Creating a Property Sheet 5
Creating a Simple Z Class 4
Creating a View 5
Creating Methods 8
Creating Property Sheets 8
Creating Views 9
Creating Z Class Instances 6

**D**
DTML Documents 3

**F**
Factories 11
Factories, add methods and Wizards 11
Factory 3, 6, 11
Folders 3

**I**
instance 6
Instances 7

**M**
manage_access 5
meta type 5, 7, 11
Methods 8, 11

**O**
Object Manager 10
ObjectManager 11
Overview 3

**P**
Permissions 10
Products 3
Properties 6
Property Sheets 5, 8
propertysheets attribute 9
Python 3, 8, 12

**S**
schema 5, 8
subclass 11
subclassing 8

Subclassing from Custom Python Classes 11
Subobjects 10

**U**
Using Z Class Instances 6

**V**
View 5
Views 5, 6

**W**
What are Z Classes 3

**Z**
Z Class 4
Z Class object 5
Z Classes 3
Z Classes and Inheritance 7
Z Classes in Depth 7
Zope Content Manager’s Guide 3