

11

THE ZCATALOG



You use the *ZCatalog* product, part of the standard Zope installation, to arrange objects in Zope according to your own indexes. ZCatalogs allow you to index and then search for objects based on the values of the indexed attributes of those objects. These attributes can be the content of documents, the values of properties, even the values returned by Python scripts and methods. In this chapter, we look at how to create a new ZCatalog and examine the individual components of a ZCatalog.

11.1 Creating a ZCatalog

To create a new ZCatalog, open the folder's add menu and select ZCatalog. The screen shown in Figure 11-1 appears.

The screenshot shows a dialog box titled "Add ZCatalog" with a "Help!" button in the top right corner. Inside the dialog, there are three input fields: "Id", "Title", and "Vocabulary". The "Vocabulary" field is a dropdown menu with "Create one for me" selected. Below the dropdown menu is an "Add" button.

Figure 11-1: Screen for creating a new ZCatalog

As with all Zope objects, you must specify an id for the ZCatalog object; you may enter a title if you wish.

11.1.1 Choosing a Vocabulary

A *vocabulary* is a list of searchable words culled from objects cataloged in the ZCatalog. As objects are indexed in the Catalog using a TextIndex (see section 11.2.5.1), the text from the object is split into words and stored in the vocabulary in a manner that allows the catalog to quickly find which words are contained in which objects. When you create a ZCatalog, you may select an existing vocabulary or create a new one (see section 11.5, “The Vocabulary”). To do the latter, select the Create one for me option from the Vocabulary list, which will automatically create a vocabulary object inside your new ZCatalog to store the word list.

This option allows you to share a vocabulary among several ZCatalogs—an important feature if you have several ZCatalogs that contain TextIndexes, because it is more efficient for them to share a single vocabulary than for each to have a separate one. To share a vocabulary between ZCatalogs, create a vocabulary object (using the add menu) in a folder that’s at or above the location of your ZCatalogs in the Zope hierarchy; then choose this vocabulary when you create each ZCatalog.

11.1.2 Globbing

If you create a vocabulary independently of a ZCatalog, you have the option to enable *globbing*, which allows you to use wildcard characters (such as asterisks [*] and question marks [?]) in your ZCatalog searches to match partial words. Generally, you will want to enable globbing to give you greater search capabilities, but you can save memory and disk space by disabling it.

NOTE *Vocabularies created for you by ZCatalog automatically have globbing enabled.*

11.2 The ZCatalog Screens

You are probably already familiar with several of the ZCatalog’s tabs common to other objects (shown in Figure 11-2), including Content, Properties, View, Find, Security, Undo, and Ownership. But Figure 11-2 also shows five new tabs: *Catalog*, *Indexes*, *Metadata*, *Find Objects*, and *Advanced*—which will be explained in sections 11.2.2 through 11.2.6.



Figure 11-2: The tab bar in a ZCatalog

11.2.1 Contents Screen

Because ZCatalogs are folderish objects, you can create objects in a ZCatalog using the ZCatalog Contents screen (Figure 11-3). For example, you could store objects to be cataloged by the ZCatalog or the ZCatalog's search and report pages (see section 11.3, "Z Search Interface").

NOTE

Cataloged objects do not need to be physically contained in the ZCatalog. In fact, they can be scattered about in many different folders throughout the Zope hierarchy (which is often the case). This ability to bring together and search objects throughout your Zope application is one of the ZCatalog's greatest strengths.

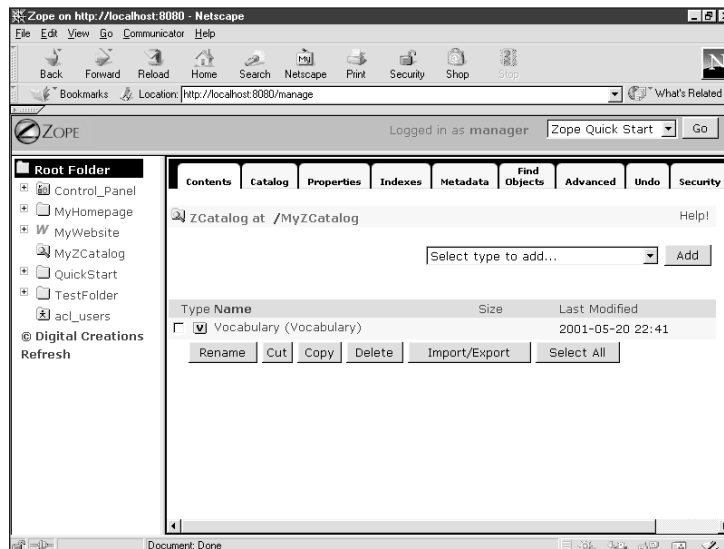


Figure 11-3: A ZCatalog Contents screen

11.2.2 The Catalog Screen

The Catalog management screen, shown in Figure 11-4, contains a list of all objects currently cataloged in the ZCatalog. You can determine which objects are cataloged by specifying the appropriate criteria on the Find Objects screen. By default, it will find and catalog all objects at or below the ZCatalog object.

You may choose to manually catalog objects using Find Objects or to allow instances of ZClasses that have the base class `CatalogAware` to catalog themselves automatically when they are created. These objects are called *catalog aware objects*.

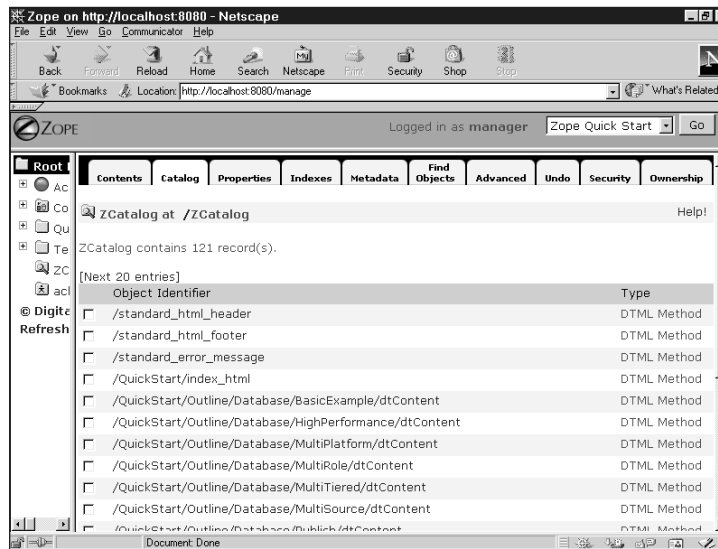


Figure 11-4: Catalog management screen for a ZCatalog

The cataloged objects are displayed in groups of 20 per page. Click the Next 20 entries or Previous 20 entries links to move forward or back through the list of cataloged objects.

11.2.2.1 The Remove and Update Buttons

The two buttons above and below the list, Remove and Update, relate only to the cataloged objects you've selected via their checkboxes. The Remove button removes the index and metadata entries for all selected objects from the ZCatalog, but does not actually delete the objects themselves. Removed objects are no longer searchable through the ZCatalog.

NOTE

If you delete a cataloged object from Zope, its entry in the ZCatalog is removed only if it is a catalog aware object. For all other objects, you must manually remove the entry from the ZCatalog on this screen.

The Update button reindexes the selected objects and updates their metadata values. When you modify the content or properties of an indexed object, you must update it to keep the ZCatalog indexes and metadata in sync. Even catalog aware objects do not reindex themselves automatically when they are modified.

NOTE

You can update all cataloged objects at once using the advanced screen (see section 11.2.6.1 for more information).

11.2.3 The Find Objects Screen

The Find Objects screen (shown in Figure 11-5) lets you search for the objects you want to catalog and index. Use it to find and index new objects. Find Objects

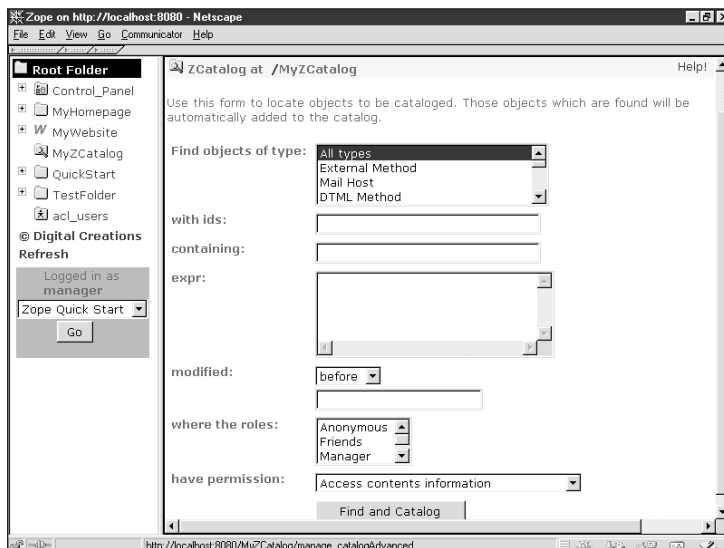


Figure 11-5: Find Objects screen for a ZCatalog

retains existing cataloged objects, so make sure the ZCatalog is empty first (see Catalog Maintenance in section 11.2.6.1) if you do not want to keep the existing entries. Here are the criteria for Find Objects:

11.2.3.1 Find objects of type:

This is a list of all available object meta types.

11.2.3.2 with ids:

Here you can specify the specific ids (separated by spaces) for each object to catalog.

11.2.3.3 containing:

Use this field to enter words, characters, and phrases that objects must contain to be cataloged. You can also enter DTML commands, such as

```
<dtml-var name="sequence-item">
```

to find and catalog DTML objects that contain that command.

11.2.3.4 expr:

Use this field to specify expressions that must be true for the objects to be cataloged. For example, you can use the following query to search for all objects larger than 1K:

```
get_size(>1024
```

Another option here is to check property values. If, for example, several objects all have the property Price, you can find those objects whose Price is \$19.95 as follows:

```
.....
Price==19.95
.....
```

11.2.3.5 modified:

This field consists of two parts: a list from which you can select either before or after (today's date), and a field in which you can enter the time and date in one of the following formats:

```
.....
2000/09/30
2000/09/30 15:00
2000/09/30 15:00:38.00 GMT+2
.....
```

11.2.3.6 where the roles:

This criterion and the next one (have permission) belong together. Here you can select from the list of roles assigned to the permission in the *have permission list*. The ZCatalog then searches for objects whose selected roles have the relevant permission.

11.2.3.7 have permission:

From this list you can select a permission that the roles in the *where the roles list* must have. The ZCatalog then finds and catalogs all objects whose roles have the specified permission.

Figure 11-6 shows an example of a completed Find Objects form. In this example, we are searching for all objects (All types) that have the id `index_html` and that contain `<dtml-var title_or_id>`. The objects must have been edited after 16:50 on August 28, 2000 (2000/08/28 16:50:00.00). In addition, the objects must have the permission Change DTML Methods for the Anonymous role.

11.2.4 Metadata Screen

ZCatalog metadata is a cache of specific attribute values (from properties and methods) of the cataloged objects, created when they were cataloged. Use the Metadata screen (see Figure 11-7) to manage the metadata for the cataloged objects. When the ZCatalog searches for objects to index, Zope saves all specified attributes of the cataloged objects to a table. The values are saved only if the object's attribute name exactly matches the metadata element name specified in this screen.

Metadata will optimize the performance of ZCatalog queries when many results are returned, because it allows you to work with the cached attribute values of many objects rather than looking up or deriving these values individually when generating a report. Use metadata sparingly, however, because it will greatly

increase the size of your Zope database if you catalog many objects and cache many of their attributes as metadata. Attribute values not stored as metadata can still be looked up from the original objects as well (see section 11.4.2 below). This can also be beneficial if an object's attribute values change often, because metadata values are not automatically updated when an object changes.

Figure 11-6: Example of a Find Objects form

Figure 11-7: Metadata Screen

11.2.4.1 Default ZCatalog Metadata Attributes

When you create a new ZCatalog, several common metadata attributes are specified by default. These elements are

- title
- id
- summary
- meta_type
- bobobase_modification_time

The attributes title, id, meta_type, and bobobase_modification_time already exist for all objects in Zope; you do not need to create them yourself. You do, however, need to create the *summary* attribute for an object.

The summary attribute is provided for CatalogAware objects; it returns the first 200 characters of a DTML document or method instead of the complete text, which saves storage space and can be used as a teaser paragraph. You can use teasers, for example, to display several summaries of many articles on one page.

Metadata elements are automatically included in the results page generated with the Z Search Interface (see section 11.3) for the ZCatalog.

11.2.5 Indexes Screen

Use the Indexes screen (see Figure 11-8) to delete or create new searchable ZCatalog indexes.

Indexes indicate which of the object's attributes (property or method values) are searchable in the ZCatalog. There are three types of indexes:

11.2.5.1 Text index

This index splits text into individual words and is frequently called a “full-text search.” When you search the text index, the results are sorted according to the number of word matches; the more instances of the search term found in the

ZCatalog at /MyZCatalog Help!

This list defines what indexes the Catalog will contain. When objects get cataloged, the values of any attributes they may have which match an index in this list will get indexed. Indexes come in three flavors, Text Indexes, Field Indexes and Keyword Indexes.

Text Indexes break text up into individual words, and are often referred to as full-text indexes. Text indexes sort results by score meaning they return hits in order from the most relevant to the least relevant.

Field Indexes treat the value of an objects attributes atomically, and can be used, for example, to track only a certain subset of object values, such as 'meta_type'.

Keyword Indexes index a sequence of objects that act as 'keywords' for an object. A Keyword Index will return any objects that have one or more keywords specified in a search query.

Index Name	Index Type
<input type="checkbox"/> PrincipiaSearchSource	Text Index
<input type="checkbox"/> bobobase_modification_time	Field Index
<input type="checkbox"/> id	Field Index
<input type="checkbox"/> meta_type	Field Index
<input type="checkbox"/> title	Text Index

Add Index:

Of Type:

Figure 11-8: Index screen for a ZCatalog

object, the higher up in the list the object will be.

Text indexes also support simple boolean search expressions (using “and,” “or,” and “and not”) and partial word matching using wildcards: “?” matches any single character and “*” matches any substring of characters. To use wildcards, the ZCatalog's vocabulary object must have globbing support enabled (see Section 11.1.2 for details). Built-in automatic support matches multiple word forms, so that a search for the word “walk” will also match “walks” and “walking” in the cataloged object.

Text indexes are used on text or string attributes that will contain multiple words or long passages of text. They also can be useful for single-word values if wildcard and multiple-word form matching is important.

NOTE

When text is indexed in a text index, all punctuation, capitalization, and numeric data in the text are ignored.

11.2.5.2 Field index

A field index is a simple index that searches against the entire value of an attribute. With a field index, you can find objects by exactly matching the indexed value or by specifying a range that it must fall between (see section 11.4.3.2 for more on range searching). Field indexes are ideal for indexing numeric or date/time data. They are also useful for indexing short string values that should be matched exactly, such as an object's meta_type attribute.

11.2.5.3 Keyword index

A keyword index allows you to search a list of values, such as a lines or tokens property. To match against a keyword index, you must match only one element of the indexed value list.

As its name implies, keyword indexes are useful for keyword searches. They are also useful as a way to express one-to-many relationships between objects. For example, you can use a lines property in an object to store the ids of related objects, then create an index on the property and use it to find objects related to one another.

11.2.6 Advanced Screen

The Advanced screen is divided into two sections: Catalog Maintenance and Subtransactions, as shown in Figure 11-9.

11.2.6.1 Catalog Maintenance

Two buttons appear in the top section of the Advanced ZCatalog screen: Update Catalog and Clear Catalog.

Update Catalog reindexes all objects that are currently indexed in the ZCatalog and updates the metadata table. It also removes the entries for any indexed objects that have been deleted but not removed from the catalog. Use it when you have added new indexes or metadata elements to your catalog to populate their values from the objects in the catalog.

The Clear Catalog button removes all cataloged objects from the ZCatalog. After using it, you can use the Find Objects tab to repopulate the ZCatalog.

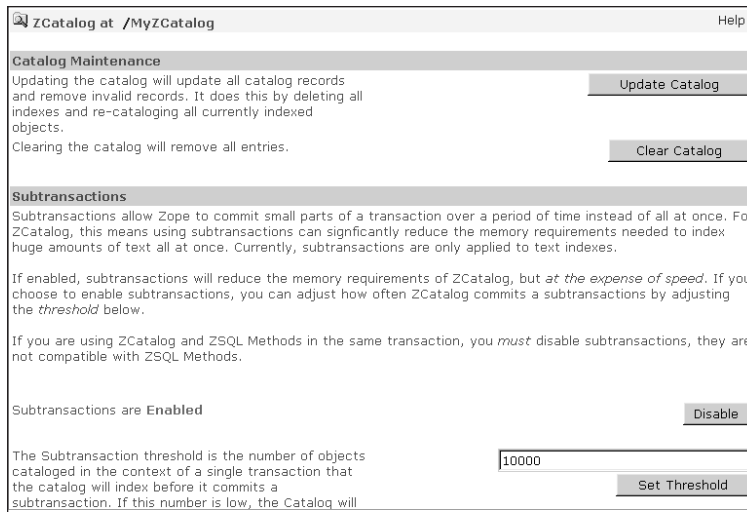


Figure 11-9: Advanced screen for a ZCatalog

11.2.6.2 Subtransactions

The bottom section of the maintenance screen is used for managing the use of subtransactions by the ZCatalog. If subtransactions are enabled in a ZCatalog, the ZCatalog indexes objects incrementally rather than all at once. This value lets you control the memory usage of ZCatalog while objects are being indexed.

The subtransaction intervals used by a ZCatalog to index objects are defined using a *threshold value*. The threshold value also determines how many words are to be indexed when there is only one instance. The higher the threshold value, the faster the indexing will take place. However, if the value is set high, the ZCatalog will require more memory. The lower the value, the less memory will be used, although indexing will take longer.

In general you should leave subtransactions enabled with the default threshold value of 10,000, which means a maximum of 10,000 field values or words of text will be held in memory before being committed to the ZODB. If you find that Zope uses excessive amounts of memory during indexing operations, you can lower the threshold value to decrease the memory required. Or, if memory is plentiful and indexing speed is most important, you can increase the threshold value or disable subtransactions entirely.

NOTE

If you are using ZCatalog together with Z SQL methods, you must disable subtransactions, because they are not compatible with Z SQL methods.

11.3 Z Search Interface

You need two pages to search a ZCatalog: one to receive the search query, and another to display the search results. Using the Z Search Interface option in the folder's add menu, you can easily create standardized search and results pages,

which can then be adapted to your needs. Figure 11-10 shows the form for creating both pages.

NOTE

To save time, create the search interface pages after you have defined all the metadata and indexes in your catalog, because these will be used to create the search and results pages. If you add metadata or indexes later, you will need to modify the pages manually.

Figure 11-10 shows the form used to create a Z Search Interface. Following the figure is an explanation of the fields on this form.

Figure 11-10: Form for creating a search and results page

Select one or more searchable objects:

This list contains all searchable objects (ZCatalogs, Z SQL methods) that can be acquired from the current folder (that is, those in the folder's namespace stack). You can select one or more searchable objects from this list.

Report Id and Report Title:

These are the id and title of the results page to be created.

Report Style:

This offers two options for presenting the results of a search: in tables or as records. The former displays each object found on a row of a table, with each metadata element in its own column. The latter displays the metadata of each object on a line separated by commas.

Search Input Id and Search Input Title:

Use these fields to specify the id and title for the search page, which is a form for users to provide the criteria with which to search the ZCatalog.

11.3.1 The Search Page

The search page consists of a simple HTML form arranged in a table, as shown in Listing 11-1. The individual cells in the table are automatically created when you create the search page, according to the objects to be searched using this form.

Listing 11-1: Standard search page for a simple ZCatalog

```

.....
<dtml-var standard_html_header>
<form action="report_html" method="get">
<h2><dtml-var document_title></h2>
Enter query parameters:<br><table>
<tr><th>Title</th>
  <td><input name="title"
    width=30 value=""></td></tr>
<tr><th>Meta type</th>
  <td><input name="meta_type"
    width=30 value=""></td></tr>
<tr><th>Id</th>
  <td><input name="id"
    width=30 value=""></td></tr>
<tr><th>Summary</th>
  <td><input name="summary"
    width=30 value=""></td></tr>
<tr><th>Bobobase modification time</th>
  <td><input name="bobobase_modification_time"
    width=30 value=""></td></tr>
<tr><td colspan=2 align=center>
<input type="SUBMIT" name="SUBMIT" value="Submit Query">
</td></tr>
</table>
</form>
<dtml-var standard_html_footer>
.....

```

If the search page is created to search ZCatalogs, the form input fields are created based on the indexes defined in the selected ZCatalog. If you specify several ZCatalogs as searchable, the indexes are taken from all the ZCatalogs, arranged according to ZCatalog, and used to create a form. The search form for the various ZCatalogs is listed as shown in Figure 11-11.

Enter query parameters:	
ThirdZCatalog/title	<input type="text"/>
ThirdZCatalog/id	<input type="text"/>
ThirdZCatalog/summary	<input type="text"/>
ThirdZCatalog/meta type	<input type="text"/>
ThirdZCatalog/bobobase modification time	<input type="text"/>
FourthZCatalog/title	<input type="text"/>
FourthZCatalog/meta type	<input type="text"/>
FourthZCatalog/id	<input type="text"/>
FourthZCatalog/summary	<input type="text"/>
FourthZCatalog/bobobase modification time	<input type="text"/>
<input type="button" value="Submit Query"/>	

Figure 11-11: Search page for two ZCatalogs (ThirdZCatalog and FourthZCatalog)

You can modify this form as needed. For example, if to provide an interface to search only some of the indexes, you could simply delete the unnecessary ones from the form. Conversely, if you later add an index to the catalog, you can make it searchable by adding an input field to the search form with the same name as the index.

If you also select one or more Z SQL methods, the arguments for the Z SQL methods are used to construct the table, as shown in Figure 11-12.

Figure 11-12: Example of a search field for a Z SQL method

NOTE *If all search fields on the form are left blank, the results page will display all of the objects indexed in the ZCatalog.*

11.3.2 The Results Page

As mentioned under “Report Style” above, Zope offers two ways to display the results page: as a table or in records. Figure 11-13 shows an example of a table report, and Figure 11-14 shows an example of a records report.

Title	Id	Summary	Meta type	Bobobase modification time	Data record id
	ThirdZCatalog		ZCatalog	2000/09/11 11:09:24.65 GMT+2	0
Vocabulary	Vocabulary		Vocabulary	2000/09/11 11:09:24.65 GMT+2	1
User Folder	acl_users		User Folder	2000/09/11 11:09:33.21 GMT+2	2
	index_html		DTML Method	2000/09/11 11:09:41.12 GMT+2	3

Figure 11-13: Search results presented as a table

```
, ThirdZCatalog, , ZCatalog, 2000/09/11 11:09:24.65 GMT+2, 0
Vocabulary, Vocabulary, , Vocabulary, 2000/09/11 11:09:24.65 GMT+2, 1
User Folder, acl_users, , User Folder, 2000/09/11 11:09:33.21 GMT+2, 2
, index_html, , DTML Method, 2000/09/11 11:09:41.12 GMT+2, 3
```

Figure 11-14: Search results presented as records

If the search and results pages apply to more than one searchable object, the corresponding results are displayed beneath each other and separated by a line, as shown in Figure 11-15.

Title	Meta type	Id	Summary	Bobbase modification time	Data record id
	ZCatalog	ThirdZCatalog		2000/09/11 11:09:24.65 GMT+2	0
Vocabulary	Vocabulary	Vocabulary		2000/09/11 11:09:24.65 GMT+2	1
User Folder	User Folder	acl_users		2000/09/11 11:09:33.21 GMT+2	2
	DTML Method	index_html		2000/09/11 11:09:41.12 GMT+2	3

ID	First Name	Name	Address
1	John	Smith	234 5th Ave
2	Jane	Connor	501 3rd Street
3	Bob	Doyle	7 Richmore Dr.

Z SQL search results ZCatalog search results

Figure 11-15: Search results for a ZCatalog and a Z SQL method

11.3.2.1 Batch Size

The standard generated results page shows 50 results per selected ZCatalog or Z SQL method. In other words, the standard batch size is 50. If there are more than 50 results, Next and Previous links appear, which allow you to navigate the results (as shown in Figure 11-16).

[\(Previous 2 results\)](#)

Title	Meta type	Id	Summary	Bobbase modification time	Data record id
Installed product StandardCacheManagers (StandardCacheManagers-1-0-0)	Product	StandardCacheManagers		2001/05/20 22:30:33.11 GMT+2	-447984125
Installed product SiteAccess (SiteAccess-2-0-0)	Product	SiteAccess		2001/05/20 22:30:32.01 GMT+2	-447984128

[\(Next 2 results\)](#)




Figure 11-16: Results page with two displayed results and Previous and Next links

To change the number of results displayed, simply replace the batch size number of 50 with the number of results that you want displayed in the following line of the report DTML method (and in other lines if there is more than one searchable object):

```
.....
<dtml-in ZCatalog size=50 start=query_start>
.....
```

If you have several searchable objects, you may need to change the batch navigation logic by renaming the query_start variable for each (for example, to query_start1, query_start2, and so on). The query_start variable determines the starting point for each batch displayed. Creating separate ones will prevent the

next 50 results being displayed for each searchable object when you want to display them for only one object.

NOTE

Remember to change the variables in all relevant locations in the source code. The other locations in addition to size in the in-tag include the lines where the Previous and Next links are created.

Figure 11-17 demonstrates a modified report page with different batch sizes for multiple search results.

(Previous 2 results)

Title	Meta type	Id	Summary	Bobbase modification time	Data record id
DateTime	Help Topic	DateTime.py		2001/05/28 23:44:23.84 GMT+2	-447984247
Cache manager associations	Help Topic	CacheManager-associate.stx		2001/05/28 23:44:23.84 GMT+2	-447984246

(Next 2 results)

(Previous 5 results)

Title	Id	Summary	Meta type	Bobbase modification time	Data record id
index_html			DTML Method	2001/01/20 22:39:14.2876 GMT+1	-1872825419
index_html			DTML Document	2001/05/27 13:20:23 GMT+2	-1872825418
login_html			DTML Method	2001/05/27 13:24:51.75 GMT+2	-1872825417
search_html			DTML Method	2001/05/28 23:44:28.79 GMT+2	-1872825416
report_html			DTML Method	2001/05/29 00:07:53.67 GMT+2	-1872825415

(Next 5 results)




Figure 11-17: Two ZCatalog results with different batch sizes (2 and 5)

11.4 ZCatalog Queries

Although the Z Search interface is useful for quickly creating a simple user interface for searching a Zope site's contents, it is often inadequate for specialized queries. To increase your flexibility, let's look at how to use a ZCatalog to customize a Z Search Interface to better suit your needs.

11.4.1 Querying ZCatalogs from a Form

The code generated by the Z Search Interface is simply a query of a ZCatalog from values specified on a form. A ZCatalog is a callable object, like a method, and when called it returns results based on the search criteria passed to it. When you call a ZCatalog by name, form values in the REQUEST object are automatically passed as query criteria. By modifying the search form, you can control the search criteria passed to the catalog.

For example, let's create a simple search form, queryForm and a simple report queryReport. In our example, there is a ZCatalog named "Catalog" with the attribute title as both an index and metadata element.

Listing 11-2 shows the DTML code for queryForm, a simple HTML form with a single input field.

Listing 11-2: Code for queryForm DTML document

```

.....
<dtml-var standard_html_header>
<form action="queryReport">
<p>Title: <input type="text" name="title" size="40"></p>
<p><input type="submit"></p>
</form>
<dtml-var standard_html_footer>
.....

```

Listing 11-3 shows the code for queryReport, which lists the titles of the cataloged objects found, based on the title entry made in queryForm.

Listing 11-3: Code for queryReport DTML Document

```

.....
<dtml-var standard_html_header>
<dtml-in name="Catalog">
    <dtml-var name="title"><br>
</dtml-in>
<dtml-var standard_html_footer>
.....

```

The most important line of code in queryReport is the dtml-in statement on the second line. This statement calls the Catalog by name, passing it any values submitted from the form. If the user types words into the title field of the form and submits it, these words are used as search criteria against the title text index in the Catalog.

11.4.2 Catalog Brains

When a ZCatalog is called, as in our simple example queryForm, it does not return the actual objects matched by the query criteria. It instead creates a sequence of *brain* objects that contain the cached metadata values of the matched objects along with several methods that can be put to various uses. It is very important to understand that when you are working with Catalog results, you are not working with the actual objects found, only their metadata. You can, however, explicitly access the actual objects if you need to.

Table 11-1 lists the methods available on the brain objects (that is, result items) returned from a ZCatalog query. Each of these methods is directly available inside of a dtml-in loop that iterates over the ZCatalog results (such as in queryForm above), because dtml-in puts each result item at the top of the namespace in turn as it steps through the result sequence.

The two most useful methods listed in Table 11-1 are getURL and getObject. The former can be used to easily create links in your results page to each object found; the latter to access any additional object attributes—such as properties and methods—that you may need in addition to the metadata cached in the ZCatalog.

NOTE

Calling getObject on many result objects can slow your ZCatalog searches. Use it sparingly and with small batch sizes when using dtml-in.

Table 11-1: ZCatalog Brain Methods

ZCatalog Brain Method	Description
has_key(key)	Returns true if the result item has a metadata element named "key."
getPath()	Returns the physical path of the result item in your Zope hierarchy.
getURL()	Returns the absolute URL of the result item (which may differ from getPath() if you are using virtual hosting).
getObject()	Returns the actual Zope object corresponding to the ZCatalog result.
getRID()	Returns the ZCatalog's internal record id for the result item, which is the same as the data_record_id_ variable returned by ZCatalog. This can be used to uniquely identify results more easily.

Let's enhance our queryReport page (Listing 11-4) using the getURL method above, which will allow us to construct a link to each object returned by the query.

Listing 11-4: Enhanced queryReport DTML document

```

.....
<dtml-var standard_html_header>
<dtml-in name="Catalog">
  <a href="<dtml-var name="getURL">">
    <dtml-var name="title">
  </a><br>
</dtml-in>
<dtml-var standard_html_footer>
.....

```

By adding the anchor tag to the queryReport code and using the getURL method of the ZCatalog brain to insert the object's URL in the href attribute, each search result now links to the object found by the query.

11.4.3 Querying ZCatalogs Directly

To query a ZCatalog directly, we call the ZCatalog using an expression and explicitly pass it the query criteria as arguments. The argument names match the indexes you wish to query against, and the values contain the query criteria for each named index. You can supply criteria for as few or as many indexes as you wish. (You can also use other arguments to control the sort order of the result items returned by the ZCatalog.)

Let's create some sample queries on a ZCatalog containing the indexes shown in Table 11-2 (which are the defaults created by Zope).

Table 11-2: ZCatalog Indexes

ZCatalog Indexes	Index Type
PrincipiaSearchSource	TextIndex
bobobase_modification_time	FieldIndex
id	FieldIndex
meta_type	FieldIndex
title	TextIndex

11.4.3.1 Basic Field Index Queries

Field Indexes can be used to find objects whose attributes match the values you specify. To match objects with a field index, we simply pass arguments with the name of each index we want to match, followed by the value desired. For example, to find all cataloged objects with the meta_type value 'DTML Document' and the id value 'index_html', you would use the following ZCatalog query expression:

```
expr="Catalog(id='index_html', meta_type='DTML Document')"
```

When you pass multiple indexes, they are “anded” together to form the search query, meaning that all criteria must be matched for the object to be returned in a result. To expand the search to match multiple values of an indexed attribute simultaneously, pass the desired values in a list to the index you are matching against. For example, to expand the above search to include DTML Document objects, use this expression:

```
expr="Catalog(id='index_html', meta_type=['DTML Document', 'DTML Method'])"
```

When you use the above expression, a meta_type value of DTML Document or DTML Method, along with an id value of index_html, would satisfy the search criteria for an object.

NOTE

Keyword indexes work the same way, except that the search value needs to match only one item of the keyword indexed list attribute (such as a lines or tokens property).

11.4.3.2 Range Queries on Field Indexes

It's a more complex process to match a field index against a range than it is to match it against an exact value. To match a range, we must specify two arguments for the index, the desired range value(s) and range usage. The range value(s) are passed in an argument matching the name of the field index. The usage argument uses the field index name followed by `_usage`. The three usage values that control how the range matching is performed against the index are shown in Table 11-3.

Table 11-3: Usage values for range matching

Range usage value	Description
range:min	Matches indexed values equal to or greater than the range value.
range:max	Matches indexed values less than or equal to the range value.
range:min:max	Matches indexed values between a specified minimum and maximum value.

For example, let's consider how we would query our ZCatalog by date, using the `bobobase_modification_time` index to find objects based on their modified date and time.

For our first example, let's find all cataloged objects modified in the last 30 days. This could be used in our bookstore application to find recently added or modified Book instances.

```
.....
expr="Catalog(bobobase_modification_time=ZopeTime()-30,
              bobobase_modification_time_usage='range:min')"
.....
```

In the first argument, which matches the index name, we pass the value of the current Zope system date and time minus 30, which calculates the date 30 days prior to today as our search value. The second usage argument instructs the ZCatalog to return objects whose date is equal to or greater than the search value.

By simply changing the usage value, we could find all object that have not been modified for at least 30 days, as follows:

```
.....
expr="Catalog(bobobase_modification_time=ZopeTime()-30,
              bobobase_modification_time_usage='range:max')"
.....
```

To match values that fall between two values (a minimum and maximum), you must pass a list containing at least two items as the search value. The item of the list with the smallest value is treated as the minimum search value and the item with the largest value as the maximum search value. For example, to find all cataloged objects modified yesterday, we could use the following expression:

```
.....
expr="Catalog(bobobase_modification_time=[(ZopeTime()-1).earliestTime(),
                                          (ZopeTime()-1).latestTime()],
              bobobase_modification_time_usage='range:min:max')"
.....
```

`(ZopeTime()-1)` returns the date and time exactly 24 hours ago as a `Date-Time` object. To determine the starting and ending time of the previous day for use as our maximum and minimum values, we use the `earliestTime()` and `latestTime()` method respectively of the `Date-Time` object. These are passed in a two-item list as the search value. We then use the `range:min:max` usage value to instruct the ZCatalog to perform the bounded query.

11.4.3.3 Sorting the Results Using Field Indexes

Field indexes are also useful for sorting resulting objects efficiently. Although the `dtml-in` tag allows you to sort a sequence using the `sort="..."` attribute, this sorting occurs in RAM, which requires all of the ZCatalog search results to be loaded into memory regardless of whether they are displayed in the current batch. Normally only those results displayed are actually loaded into memory.

To sort results efficiently using a field index, pass a `sort_on` argument to the ZCatalog, with its value set to the name of the field index to sort by. You can also specify an optional second `sort_order` argument to determine whether the sort order is ascending or descending. This sorting occurs when the query is performed, thus avoiding the memory and performance penalties of the `dtml-in` sort attribute.

Let's extend the earlier example that returned objects modified in the last 30 days. To return the results sorted by modified date with the most recently modified object appearing first, use this expression:

```
.....
expr="Catalog(bobobase_modification_time=ZopeTime()-30,
              bobobase_modification_time_usage='range:min',
              sort_on='bobobase_modification_time',
              sort_order='descending')"
.....
```

NOTE

Because `sort_on` can be used only with field indexes, you will still need to use the `dtml-in` sort option to sort by other attribute values, like an object's title. To sort using the `dtml-in` sort option, the desired attribute must be a metadata element of the ZCatalog.

11.4.3.4 Text Index Queries

Text index query arguments are coded like exact field index queries, except that the value represents a query string rather than an exact value to match. This query string allows simple boolean searching and partial word-match functionality on large quantities of indexed text (see section 11.2.5.1 for more details).

Our example ZCatalog has two text indexes: `title` and `PrincipiaSearchSource`. The former is the title property of the object, the latter is the body text of the document or object. To find cataloged objects whose titles contain the words "event" and "calendar," you could use the following expression.

```
.....
expr="Catalog(title='event and calendar')"
.....
```

As we discussed with field indexes, if multiple indexes are included, all the search criteria must be satisfied on an object for it to be returned.

NOTE

You can simultaneously search field, text, and keyword indexes in a single expression by combining the other techniques described above in section 11.4.3.

11.4.3.5 ZCatalog Query Limitations

Although it is easy to create “and” queries that match criteria from multiple indexes and return objects satisfying all of them, it is difficult to perform “or” queries across multiple indexes. For example, you cannot use a simple query to return objects where the search criteria match either the title or PrincipiaSearchSource indexes.

There are two ways to overcome this limitation. The first is to perform each query separately and combine the results into a single result set. The second is to create a new method attribute of the object or a script that programmatically combines the contents of the attributes, then index against this combined value.

Let’s explore the first option: combining search result sets, which you can do simply by adding the results of ZCatalog queries using the plus (+) operator. For example, to return all objects containing the word “victory” in either their title or body text, do the following:

```
.....
expr="Catalog(title='victory') + Catalog(PrincipiaSearchSource='victory')
.....
```

Although this approach will get us what we want, it has one slight problem: Objects containing the search word in both attributes will be returned twice by our expression, which is not usually the desired result.

The second approach, to create method or script that combines the attribute values, is somewhat more complex and involves writing Python code, but it does solve the duplicate results problem. The sample Python script shown in Listing 11-5, `text_content`, can be created in Zope’s root folder, so that it can be acquired by all of the cataloged objects.

Listing 11-5: `text_content` Python Script

```
.....
return context.title + '\n' + context.PrincipiaSearchSource()
.....
```

This script returns the object’s title, followed by a line break (`\n`) and the body text returned by the `PrincipiaSearchSource` method of the object. If, after creating the script, you create a `TextIndex` called “`text_content`” in your `ZCatalog` and update it (using the advanced tab); the `ZCatalog` will call this method for all cataloged objects and index the result. After doing this, you can perform the desired query using the following expression:

```
.....
expr="Catalog(text_content='victory')
.....
```

(We will discuss Python in more detail in Chapters 13 through 15.)

NOTE

You cannot use a DTML method as the source for an index, because the ZCatalog expects the attributes to be simple values, properties, or Python methods/scripts that do not require arguments.

11.5 The Vocabulary

When you create a new ZCatalog, you are asked whether you also want to create a new vocabulary or use an existing one. A vocabulary contains words (normal text, DTML, and HTML tags) found in the objects cataloged in the corresponding ZCatalog.

11.5.1 The Vocabulary Screen

The Vocabulary screen, the Vocabulary object's main screen, contains a list of all words found in the attributes of cataloged objects that are text indexed. The batch size of this list, that is, the number of words displayed, is 20 words per page.

DTML and HTML tags are stored in the vocabulary without angle brackets, and DTML tags are stored as single words without hyphens. For example, say you have a ZCatalog named "MyZCatalog" that contains only one object: the DTML document `index_html` with the source text shown in Listing 11-6.

Listing 11-6: `index_html` source text

```
.....
<dtml-var id>
<hr>
<a href="../index_html">Back</a>
hey, you!
hello
hi there
many hellos!
.....
```

Figure 11-18 shows the words generated in the vocabulary from this source text.

11.5.2 Searching the Vocabulary

You can use the Query screen's short form, consisting of an input field and a Submit button, to search the vocabulary. If a vocabulary is created using globbing (see section 11.1.2 for details), you can search it using wildcards (such as `*` or `?`). For example, if you were to search the vocabulary shown in Figure 11-18, the results of the following queries would be as shown:

```
he*           ['hey', 'hello']
hellos       ['hello']
hello        ['hello']
h?l*        ['hello']
*y           ['hey', 'many']
```

NOTE

The globbing option is chosen only when the vocabulary object is created. You cannot change the globbing option after the fact. To change the globbing option, you would need to delete the original vocabulary, create a new one with the same id, and update all the ZCatalogs that share the vocabulary (using the Advanced screen of the ZCatalogs).

Vocabulary		Query	Undo	Ownership	Security
▼ Vocabulary at /Catalog/Vocabulary					Help
Vocabulary contains 14 word(s).					
Word	Word ID				
/a	134076537				
/index_html	1272425366				
back	-80650770				
dtmlvar	-1435263603				
hello	-45658602				
hellos	1795908136				
hey	-275390192				
hi	246503988				
hr	181604265				
href	439883501				
id	1531093162				
many	1082139467				
there	394981047				
you	1851629454				

Figure 11-18: Vocabulary example

11.5.3 Expanding the Vocabulary: The `insert()` and `manage_insert()` Methods

You can expand a vocabulary using the `insert()` and `manage_insert()` methods.

11.5.3.1 The `insert()` Method

You can enter new words into a vocabulary yourself using the vocabulary object's `insert()` method. Using this method, you can, for example, insert the word “catalog” using DTML (assuming that the vocabulary has the id `Vocabulary`):

```
<dtml-call "Vocabulary.insert('Catalog')">
```

11.5.3.2 The `manage_insert()` Method

You can also enter new words in a vocabulary with the `manage_insert()` method, using the URL of the vocabulary, as follows:

```
http://localhost:8080/Vocabulary/manage_insert?word=Catalog
```

As you can see, the parameter shown for this method—that is, the word to be added—is named “word.” If your addition is actually several words and you want to include it in the vocabulary using the URL, enter it like this:

```
http://localhost:8080/Vocabulary/manage_insert?word=My+Catalog
```

To insert new words using a form, insert the input value using the `url_quote_plus` option of `dtml-var` (see Appendix A).

11.5.4 ZCatalog with Special Characters

The ZCatalog supplied by with Zope cannot index words with foreign characters correctly—for example, German umlauts or Japanese Kanji characters—to enable searches for them. However, there are several replacement ZCatalog splitters to handle indexing foreign character sets.

One such splitter patch is the Voodoo Kludge Splitter, which lets you map special characters so that they are indexed properly in ZCatalog vocabulary. You can download this product at

.....
<http://www.zope.org/Members/kslee/VoodooKludgeSplitter>
.....

NOTE

Unfortunately this Splitter is slower than the one delivered with Zope and, as a result, updates take longer. You should, therefore, consider whether you want improved speed or special characters.

Summary

This chapter has explored how to use ZCatalogs to index and search Zope objects and examined the strengths and weaknesses of ZCatalog and how to use it effectively and efficiently. Armed with this knowledge, you should be able to construct search engines for your Zope website to facilitate a site search or other more specific object query functionality.

Chapter 12 will explore how to connect Zope to external relational database systems.