zope.catalog Documentation

Release 4.2.2.dev0

Zope Foundation and Contributors

Oct 18, 2018

Contents

1 Using `zope.catalog` 3
- 1.1 Basic Usage . . . 3
- 1.2 Additional Topics . . . 9

2 zope.catalog 11
- 2.1 Interfaces . . . 11
- 2.2 Catalog . . . 13
- 2.3 Index Implementations . . . 14

3 Hacking on `zope.catalog` 19
- 3.1 Getting the Code . . . 19
- 3.2 Working in a `virtualenv` . . . 19
- 3.3 Using `zc.buildout` . . . 21
- 3.4 Using `tox` . . . 22
- 3.5 Contributing to `zope.catalog` . . . 23

4 Indices and tables 25

Python Module Index 27

Contents:

CHAPTER 1

Using zope.catalog

1.1 Basic Usage

Let's look at an example:

```
>>> from zope.catalog.catalog import Catalog
>>> cat = Catalog()
```

We can add catalog indexes to catalogs. A catalog index is, among other things, an attribute index. It indexes attributes of objects. To see how this works, we'll create a demonstration attribute index. Our attribute index will simply keep track of objects that have a given attribute value. The *catalog* package provides an attribute-index mix-in class that is meant to work with a base indexing class. First, we'll write the base index class:

```
>>> import persistent, BTrees.OOBTree, BTrees.IFBTree, BTrees.IOBTree
>>> import zope.interface, zope.index.interfaces

>>> @zope.interface.implementer(
...         zope.index.interfaces.IInjection,
...         zope.index.interfaces.IIndexSearch,
...         zope.index.interfaces.IIndexSort,
...         )
... class BaseIndex(persistent.Persistent):
...
...     def clear(self):
...         self.forward = BTrees.OOBTree.OOBTree()
...         self.backward = BTrees.IOBTree.IOBTree()
...
...     __init__ = clear
...
...     def index_doc(self, docid, value):
...         if docid in self.backward:
...             self.unindex_doc(docid)
...         self.backward[docid] = value
...
```

(continues on next page)

(continued from previous page)

```
...         set = self.forward.get(value)
...         if set is None:
...             set = BTrees.IFBTree.IFTreeSet()
...             self.forward[value] = set
...         set.insert(docid)
...
...     def unindex_doc(self, docid):
...         value = self.backward.get(docid)
...         if value is None:
...             return
...         self.forward[value].remove(docid)
...         del self.backward[docid]
...
...     def apply(self, value):
...         set = self.forward.get(value)
...         if set is None:
...             set = BTrees.IFBTree.IFTreeSet()
...         return set
...
...     def sort(self, docids, limit=None, reverse=False):
...         key_func = lambda x: self.backward.get(x, -1)
...         for i, docid in enumerate(
...                 sorted(docids, key=key_func, reverse=reverse)):
...             yield docid
...             if limit and i >= (limit - 1):
...                 break
```

The class implements *IInjection* to allow values to be indexed and unindexed and *IIndexSearch* to support searching via the *apply* method.

Now, we can use the AttributeIndex mixin to make this an attribute index:

```
>>> import zope.catalog.attribute
>>> import zope.catalog.interfaces
>>> import zope.container.contained

>>> @zope.interface.implementer(zope.catalog.interfaces.ICatalogIndex)
... class Index(zope.catalog.attribute.AttributeIndex,
...             BaseIndex,
...             zope.container.contained.Contained,
...             ):
...    pass
```

Unfortunately, because of the way we currently handle containment constraints, we have to provide *ICatalogIndex*, which extends *IContained*. We subclass *Contained* to get an implementation for *IContained*.

Now let's add some of these indexes to our catalog. Let's create some indexes. First we'll define some interfaces providing data to index:

```
>>> class IFavoriteColor(zope.interface.Interface):
...     color = zope.interface.Attribute("Favorite color")

>>> class IPerson(zope.interface.Interface):
...     def age():
...         """Return the person's age, in years"""
```

We'll create color and age indexes:

```
>>> cat['color'] = Index('color', IFavoriteColor)
>>> cat['age'] = Index('age', IPerson, True)
>>> cat['size'] = Index('sz')
```

The indexes are created with:

- the name of the of the attribute to index
- the interface defining the attribute, and
- a flag indicating whether the attribute should be called, which defaults to false.

If an interface is provided, then we'll only be able to index an object if it can be adapted to the interface, otherwise, we'll simply try to get the attribute from the object. If the attribute isn't present, then we'll ignore the object.

Now, let's create some objects and index them:

```
>>> @zope.interface.implementer(IPerson)
... class Person:
...     def __init__(self, age):
...         self._age = age
...     def age(self):
...         return self._age

>>> @zope.interface.implementer(IFavoriteColor)
... class Discriminating:
...     def __init__(self, color):
...         self.color = color

>>> class DiscriminatingPerson(Discriminating, Person):
...     def __init__(self, age, color):
...         Discriminating.__init__(self, color)
...         Person.__init__(self, age)

>>> class Whatever:
...     def __init__(self, **kw): #**
...         self.__dict__.update(kw)

>>> o1 = Person(10)
>>> o2 = DiscriminatingPerson(20, 'red')
>>> o3 = Discriminating('blue')
>>> o4 = Whatever(a=10, c='blue', sz=5)
>>> o5 = Whatever(a=20, c='red', sz=6)
>>> o6 = DiscriminatingPerson(10, 'blue')

>>> cat.index_doc(1, o1)
>>> cat.index_doc(2, o2)
>>> cat.index_doc(3, o3)
>>> cat.index_doc(4, o4)
>>> cat.index_doc(5, o5)
>>> cat.index_doc(6, o6)
```

We search by providing query mapping objects that have a key for every index we want to use:

```
>>> list(cat.apply({'age': 10}))
[1, 6]
>>> list(cat.apply({'age': 10, 'color': 'blue'}))
[6]
>>> list(cat.apply({'age': 10, 'color': 'blue', 'size': 5}))
```

(continues on next page)

(continued from previous page)

```
[]
>>> list(cat.apply({'size': 5}))
[4]
```

We can unindex objects:

```
>>> cat.unindex_doc(4)
>>> list(cat.apply({'size': 5}))
[]
```

and reindex objects:

```
>>> o5.sz = 5
>>> cat.index_doc(5, o5)
>>> list(cat.apply({'size': 5}))
[5]
```

If we clear the catalog, we'll clear all of the indexes:

```
>>> cat.clear()
>>> [len(index.forward) for index in cat.values()]
[0, 0, 0]
```

Note that you don't have to use the catalog's search methods. You can access its indexes directly, since the catalog is a mapping:

```
>>> [(str(name), cat[name].field_name) for name in cat]
[('age', 'age'), ('color', 'color'), ('size', 'sz')]
```

Catalogs work with int-id utilities, which are responsible for maintaining id <-> object mappings. To see how this works, we'll create a utility to work with our catalog:

```
>>> import zope.intid.interfaces
>>> @zope.interface.implementer(zope.intid.interfaces.IIntIds)
... class Ids:
...     def __init__(self, data):
...         self.data = data
...     def getObject(self, id):
...         return self.data[id]
...     def __iter__(self):
...         return iter(self.data)
>>> ids = Ids({1: o1, 2: o2, 3: o3, 4: o4, 5: o5, 6: o6})

>>> from zope.component import provideUtility
>>> provideUtility(ids, zope.intid.interfaces.IIntIds)
```

With this utility in place, catalogs can recompute indexes:

```
>>> cat.updateIndex(cat['size'])
>>> list(cat.apply({'size': 5}))
[4, 5]
```

Of course, that only updates *that* index:

```
>>> list(cat.apply({'age': 10}))
[]
```

We can update all of the indexes:

```
>>> cat.updateIndexes()
>>> list(cat.apply({'age': 10}))
[1, 6]
>>> list(cat.apply({'color': 'red'}))
[2]
```

There's an alternate search interface that returns "result sets". Result sets provide access to objects, rather than object ids:

```
>>> result = cat.searchResults(size=5)
>>> len(result)
2
>>> list(result) == [o4, o5]
True
```

The searchResults method also provides a way to sort, limit and reverse results.

When not using sorting, limiting and reversing are done by simple slicing and list reversing.

```
>>> list(cat.searchResults(size=5, _reverse=True)) == [o5, o4]
True

>>> list(cat.searchResults(size=5, _limit=1)) == [o4]
True

>>> list(cat.searchResults(size=5, _limit=1, _reverse=True)) == [o5]
True
```

However, when using sorting by index, the limit and reverse parameters are passed to the index `sort` method so it can do it efficiently.

Let's index more objects to work with:

```
>>> o7 = DiscriminatingPerson(7, 'blue')
>>> o8 = DiscriminatingPerson(3, 'blue')
>>> o9 = DiscriminatingPerson(14, 'blue')
>>> o10 = DiscriminatingPerson(1, 'blue')
>>> ids.data.update({7: o7, 8: o8, 9: o9, 10: o10})
>>> cat.index_doc(7, o7)
>>> cat.index_doc(8, o8)
>>> cat.index_doc(9, o9)
>>> cat.index_doc(10, o10)
```

Now we can search all people who like blue, ordered by age:

```
>>> results = list(cat.searchResults(color='blue', _sort_index='age'))
>>> results == [o3, o10, o8, o7, o6, o9]
True

>>> results = list(cat.searchResults(color='blue', _sort_index='age', _limit=3))
>>> results == [o3, o10, o8]
True

>>> results = list(cat.searchResults(color='blue', _sort_index='age', _reverse=True))
>>> results == [o9, o6, o7, o8, o10, o3]
True
```

(continues on next page)

(continued from previous page)

```
>>> results = list(cat.searchResults(color='blue', _sort_index='age', _reverse=True, _
↪limit=4))
>>> results == [o9, o6, o7, o8]
True
```

The index example we looked at didn't provide document scores. Simple indexes normally don't, but more complex indexes might give results scores, according to how closely a document matches a query. Let's create a new index, a "keyword index" that indexes sequences of values:

```
>>> @zope.interface.implementer(
...         zope.index.interfaces.IInjection,
...         zope.index.interfaces.IIndexSearch,
...         )
... class BaseKeywordIndex(persistent.Persistent):
...
...     def clear(self):
...         self.forward = BTrees.OOBTree.OOBTree()
...         self.backward = BTrees.IOBTree.IOBTree()
...
...     __init__ = clear
...
...     def index_doc(self, docid, values):
...         if docid in self.backward:
...             self.unindex_doc(docid)
...         self.backward[docid] = values
...
...         for value in values:
...             set = self.forward.get(value)
...             if set is None:
...                 set = BTrees.IFBTree.IFTreeSet()
...                 self.forward[value] = set
...             set.insert(docid)
...
...     def unindex_doc(self, docid):
...         values = self.backward.get(docid)
...         if values is None:
...             return
...         for value in values:
...             self.forward[value].remove(docid)
...         del self.backward[docid]
...
...     def apply(self, values):
...         result = BTrees.IFBTree.IFBucket()
...         for value in values:
...             set = self.forward.get(value)
...             if set is not None:
...                 _, result = BTrees.IFBTree.weightedUnion(result, set)
...         return result

>>> @zope.interface.implementer(zope.catalog.interfaces.ICatalogIndex)
... class KeywordIndex(zope.catalog.attribute.AttributeIndex,
...                    BaseKeywordIndex,
...                    zope.container.contained.Contained,
...                    ):
...    pass
```

Now, we'll add a hobbies index:

```
>>> cat['hobbies'] = KeywordIndex('hobbies')
>>> o1.hobbies = 'camping', 'music'
>>> o2.hobbies = 'hacking', 'sailing'
>>> o3.hobbies = 'music', 'camping', 'sailing'
>>> o6.hobbies = 'cooking', 'dancing'
>>> cat.updateIndexes()
```

When we apply the catalog:

```
>>> cat.apply({'hobbies': ['music', 'camping', 'sailing']})
BTrees.IFBTree.IFBucket([(1, 2.0), (2, 1.0), (3, 3.0)])
```

We found objects 1-3, because they each contained at least some of the words in the query. The scores represent the number of words that matched. If we also include age:

```
>>> cat.apply({'hobbies': ['music', 'camping', 'sailing'], 'age': 10})
BTrees.IFBTree.IFBucket([(1, 3.0)])
```

The score increased because we used an additional index. If an index doesn't provide scores, scores of 1.0 are assumed.

1.2 Additional Topics

1.2.1 Automatic indexing with events

In order to automatically keep the catalog up-to-date any objects that are added to a intid utility are indexed automatically. Also when an object gets modified it is reindexed by listening to IObjectModified events.

Let us create a fake catalog to demonstrate this behaviour. We only need to implement the index_doc method for this test.

```
>>> from zope.catalog.interfaces import ICatalog
>>> from zope import interface, component
>>> @interface.implementer(ICatalog)
... class FakeCatalog(object):
...     indexed = []
...     def index_doc(self, docid, obj):
...         self.indexed.append((docid, obj))
>>> cat = FakeCatalog()
>>> component.provideUtility(cat)
```

We also need an intid util and a keyreference adapter.

```
>>> from zope.intid import IntIds
>>> from zope.intid.interfaces import IIntIds
>>> intids = IntIds()
>>> component.provideUtility(intids, IIntIds)
>>> from zope.keyreference.testing import SimpleKeyReference
>>> component.provideAdapter(SimpleKeyReference)

>>> from  zope.container.contained import Contained
>>> class Dummy(Contained):
...     def __init__(self, name):
...         self.__name__ = name
```

(continues on next page)

(continued from previous page)

```
...     def __repr__(self):
...         return '<Dummy %r>' % str(self.__name__)
```

We have a subscriber to IIntidAddedEvent.

```
>>> from zope.catalog import catalog
>>> from zope.intid.interfaces import IntIdAddedEvent
>>> d1 = Dummy(u'one')
>>> id1 = intids.register(d1)
>>> catalog.indexDocSubscriber(IntIdAddedEvent(d1, None))
```

Now we have indexed the object.

```
>>> cat.indexed.pop()
(..., <Dummy 'one'>)
```

When an object is modified an objectmodified event should be fired by the application. Here is the handler for such an event.

```
>>> from zope.lifecycleevent import ObjectModifiedEvent
>>> catalog.reindexDocSubscriber(ObjectModifiedEvent(d1))
>>> len(cat.indexed)
1
>>> cat.indexed.pop()
(..., <Dummy 'one'>)
```

Preventing automatic indexing

Sometimes it is not accurate to automatically index an object. For example when a lot of indexes are in the catalog and only specific indexes needs to be updated. There are marker interfaces to achieve this.

```
>>> from zope.catalog.interfaces import INoAutoIndex
```

If an object provides this interface it is not automatically indexed.

```
>>> interface.alsoProvides(d1, INoAutoIndex)
>>> catalog.indexDocSubscriber(IntIdAddedEvent(d1, None))
>>> len(cat.indexed)
0

>>> from zope.catalog.interfaces import INoAutoReindex
```

If an object provides this interface it is not automatically reindexed.

```
>>> interface.alsoProvides(d1, INoAutoReindex)
>>> catalog.reindexDocSubscriber(ObjectModifiedEvent(d1))
>>> len(cat.indexed)
0
```

CHAPTER 2

zope.catalog

2.1 Interfaces

Catalog Interfaces

interface `zope.catalog.interfaces.`**`ICatalogQuery`**

Provides Catalog Queries.

`searchResults``(`*`**kw`*`)`

Search on the given indexes.

Keyword arguments dictionary keys are index names and values are queries for these indexes.

Keyword arguments has some special names, used by the catalog itself:

- _sort_index - The name of index to sort results with. This index must implement zope.index.interfaces.IIndexSort.
- _limit - Limit result set by this number, useful when used with sorting.
- _reverse - Reverse result set, also useful with sorting.

interface `zope.catalog.interfaces.`**`ICatalogEdit`**

Extends: `zope.index.interfaces.IInjection`

Allows one to manipulate the Catalog information.

`updateIndexes``()`

Reindex all objects.

interface `zope.catalog.interfaces.`**`ICatalogIndex`**

Extends: `zope.index.interfaces.IInjection`, `zope.index.interfaces.IIndexSearch`

An index to be used in a catalog

`__parent__`

Implementation `zope.schema.Field`

Read Only False

Required True

Default Value None

interface `zope.catalog.interfaces.`**`ICatalog`**

Extends: *`zope.catalog.interfaces.ICatalogQuery`*, *`zope.catalog.interfaces.ICatalogEdit`*, `zope.container.interfaces.IContainer`

Marker to describe a catalog in content space.

`__setitem__`(*key*, *value*)

Add the given *object* to the container under the given name.

Raises a `TypeError` if the key is not a unicode or ascii string.

Raises a `ValueError` if the key is empty, or if the key contains a character which is not allowed in an object name.

Raises a `KeyError` if the key violates a uniqueness constraint.

The container might choose to add a different object than the one passed to this method.

If the object doesn't implement *IContained*, then one of two things must be done:

1. If the object implements *ILocation*, then the *IContained* interface must be declared for the object.
2. Otherwise, a *ContainedProxy* is created for the object and stored.

The object's *__parent__* and *__name__* attributes are set to the container and the given name.

If the old parent was `None`, then an *IObjectAddedEvent* is generated, otherwise, an *IObjectMovedEvent* is generated. An *IContainerModifiedEvent* is generated for the container.

If the object replaces another object, then the old object is deleted before the new object is added, unless the container vetos the replacement by raising an exception.

If the object's *__parent__* and *__name__* were already set to the container and the name, then no events are generated and no hooks. This allows advanced clients to take over event generation.

interface `zope.catalog.interfaces.`**`IAttributeIndex`**

I index objects by first adapting them to an interface, then retrieving a field on the adapted object.

`interface`

Interface

Objects will be adapted to this interface

Implementation `zope.schema.Choice`

Read Only False

Required False

Default Value None

`field_name`

Field Name

Name of the field to index

Implementation `zope.schema.NativeStringLine`

Read Only False

Required True

Default Value None

Allowed Type `str`

`field_callable`
Field Callable

If true, then the field should be called to get the value to be indexed

Implementation `zope.schema.Bool`

Read Only False

Required True

Default Value None

Allowed Type `bool`

`interface` `zope.catalog.interfaces.`**`INoAutoIndex`**
Marker for objects that should not be automatically indexed

`interface` `zope.catalog.interfaces.`**`INoAutoReindex`**
Marker for objects that should not be automatically reindexed

2.2 Catalog

Catalog

`class` `zope.catalog.catalog.`**`ResultSet`**(*uids*, *uidutil*)
Lazily accessed set of objects.

`class` `zope.catalog.catalog.`**`Catalog`**(*family=None*)
Bases: `zope.container.btree.BTreeContainer`

`index_doc`(*docid*, *texts*)
Register the data in indexes of this catalog.

`unindex_doc`(*docid*)
Unregister the data from indexes of this catalog.

`zope.catalog.catalog.`**`indexAdded`**(*index*, *event*)
When an index is added to a catalog, we have to index existing objects

When an index is added, we tell it's parent to index it:

```
>>> class FauxCatalog:
...     updated = None
...     def updateIndex(self, index):
...         self.updated = index
```

```
>>> class FauxIndex:
...     pass
```

```
>>> index = FauxIndex()
>>> index.__parent__ = FauxCatalog()
```

```
>>> from zope.catalog.catalog import indexAdded
>>> indexAdded(index, None)
>>> index.__parent__.updated is index
True
```

zope.catalog.catalog.**indexDocSubscriber**(*event*)
A subscriber to IntIdAddedEvent

zope.catalog.catalog.**reindexDocSubscriber**(*event*)
A subscriber to ObjectModifiedEvent

zope.catalog.catalog.**unindexDocSubscriber**(*event*)
A subscriber to IntIdRemovedEvent

2.3 Index Implementations

The implementations of various kinds of indexes are spread across a few modules.

2.3.1 Attribute Indexes

Index interface-defined attributes

class zope.catalog.attribute.**AttributeIndex**(*field_name=None*, *interface=None*, *field_callable=False*, **args*, ***kwargs*)

Bases: object

Index interface-defined attributes

Mixin for indexing a particular attribute of an object after first adapting the object to be indexed to an interface.

The class is meant to be mixed with a base class that defines an `index_doc` method and an `unindex_doc` method:

```
>>> class BaseIndex(object):
...     def __init__(self):
...         self.data = []
...     def index_doc(self, id, value):
...         self.data.append((id, value))
...     def unindex_doc(self, id):
...         for n, (iid, _) in enumerate(self.data):
...             if id == iid:
...                 del self.data[n]
...                 break
```

The class does two things. The first is to get a named field from an object:

```
>>> class Data(object):
...     def __init__(self, v):
...         self.x = v
```

```
>>> from zope.catalog.attribute import AttributeIndex
>>> class Index(AttributeIndex, BaseIndex):
...     pass
```

```
>>> index = Index('x')
>>> index.index_doc(11, Data(1))
>>> index.index_doc(22, Data(2))
>>> index.data
[(11, 1), (22, 2)]
```

If the field value is `None`, indexing it removes it from the index:

```
>>> index.index_doc(11, Data(None))
>>> index.data
[(22, 2)]
```

If the field names a method (any callable object), the results of calling that field can be indexed:

```
>>> def z(self): return self.x + 20
>>> Data.z = z
>>> index = Index('z', field_callable=True)
>>> index.index_doc(11, Data(1))
>>> index.index_doc(22, Data(2))
>>> index.data
[(11, 21), (22, 22)]
```

Of course, if you neglect to set `field_callable` when you index a method, it's likely that most concrete index implementations will raise an exception, but this class will happily pass that callable on:

```
>>> index = Index('z')
>>> data = Data(1)
>>> index.index_doc(11, data)
>>> index.data
[(11, <bound method ...>>)]
```

The class can also adapt an object to an interface before getting the field:

```
>>> from zope.interface import Interface
>>> class I(Interface):
...     pass
```

```
>>> class Data(object):
...     def __init__(self, v):
...         self.x = v
...     def __conform__(self, iface):
...         if iface is I:
...             return Data2(self.x)
```

```
>>> class Data2(object):
...     def __init__(self, v):
...         self.y = v * v
```

```
>>> index = Index('y', I)
>>> index.index_doc(11, Data(3))
>>> index.index_doc(22, Data(2))
>>> index.data
[(11, 9), (22, 4)]
```

If adapting to the interface fails, the object is not indexed:

```
>>> class I2(Interface): pass
>>> I2(Data(3), None) is None
True
>>> index = Index('y', I2)
>>> index.index_doc(11, Data(3))
>>> index.data
[]
```

When you define an index class, you can define a default interface and/or a default interface:

```
>>> class Index(AttributeIndex, BaseIndex):
...     default_interface = I
...     default_field_name = 'y'
```

```
>>> index = Index()
>>> index.index_doc(11, Data(3))
>>> index.index_doc(22, Data(2))
>>> index.data
[(11, 9), (22, 4)]
```

default_field_name = None

Subclasses can set this to a string if they want to allow construction without passing the `field_name`.

default_interface = None

Subclasses can set this to an interface (a callable taking the object do index and the default value to return) if they want to allow construction that doesn't provide an `interface`.

index_doc(*docid*, *object*)

Derives the value to index for *object*.

Uses the interface passed to the constructor to adapt the object, and then gets the field (calling it if `field_callable` was set). If the value thus found is `None`, calls `unindex_doc`. Otherwise, passes the *docid* and the value to the superclass's implementation of `index_doc`.

2.3.2 Field Indexes

Field catalog indexes

interface zope.catalog.field.**IFieldIndex**

Extends: *zope.catalog.interfaces.IAttributeIndex*, *zope.catalog.interfaces.ICatalogIndex*

Interface-based catalog field index

class zope.catalog.field.**FieldIndex**(*field_name=None*, *interface=None*, *field_callable=False*, **args*, ***kwargs*)

Bases: *zope.catalog.attribute.AttributeIndex*, `zope.index.field.index.FieldIndex`, `zope.container.contained.Contained`

Default implementation of a *IFieldIndex*.

2.3.3 Keyword Indexes

Keyword catalog index

interface zope.catalog.keyword.**IKeywordIndex**

Extends: *zope.catalog.interfaces.IAttributeIndex*, *zope.catalog.interfaces.ICatalogIndex*

Interface-based catalog keyword index

class zope.catalog.keyword.**KeywordIndex**(*field_name=None*, *interface=None*, *field_callable=False*, **args*, ***kwargs*)

Bases: *zope.catalog.attribute.AttributeIndex*, `zope.index.keyword.index.KeywordIndex`, `zope.container.contained.Contained`

Default implementation of *IKeywordIndex*.

class zope.catalog.keyword.**CaseInsensitiveKeywordIndex**(*field_name=None*, *interface=None*, *field_callable=False*, **args*, ***kwargs*)

Bases: *zope.catalog.attribute.AttributeIndex*, zope.index.keyword.index.CaseInsensitiveKeywordIndex, zope.container.contained.Contained

A kind of *IKeywordIndex* that is not sensitive to case.

2.3.4 Text Indexes

Text catalog indexes

interface zope.catalog.text.**ITextIndex**

Extends: *zope.catalog.interfaces.IAttributeIndex*, *zope.catalog.interfaces.ICatalogIndex*

Interface-based catalog text index.

We redefine the fields that *zope.catalog.interfaces.IAttributeIndex* defines in order to change their defaults.

interface

Interface

Objects will be adapted to this interface. The default is `zope.index.text.interfaces.ISearchableText`

Implementation `zope.schema.Choice`

Read Only False

Required False

Default Value <InterfaceClass zope.index.text.interfaces.ISearchableText>

field_name

Field Name

Name of the field to index. Defaults to `getSearchableText`.

Implementation `zope.schema.NativeStringLine`

Read Only False

Required True

Default Value ‘getSearchableText’

Allowed Type `str`

field_callable

Field Callable

If true (the default), then the field should be called to get the value to be indexed

Implementation `zope.schema.Bool`

Read Only False

Required True

Default Value True

Allowed Type `bool`

class zope.catalog.text.**TextIndex**(*field_name=None*, *interface=None*, *field_callable=False*, **args*, ***kwargs*)

Bases: *zope.catalog.attribute.AttributeIndex*, zope.index.text.textindex.TextIndex, zope.container.contained.Contained

Default implementation of *ITextIndex*.

CHAPTER 3

Hacking on `zope.catalog`

3.1 Getting the Code

The main repository for `zope.catalog` is in the Zope Foundation Github repository:

> https://github.com/zopefoundation/zope.catalog

You can get a read-only checkout from there:

```
$ git clone https://github.com/zopefoundation/zope.catalog.git
```

or fork it and get a writeable checkout of your fork:

```
$ git clone git@github.com/jrandom/zope.catalog.git
```

The project also mirrors the trunk from the Github repository as a Bazaar branch on Launchpad:

https://code.launchpad.net/zope.catalog

You can branch the trunk from there using Bazaar:

```
$ bzr branch lp:zope.catalog
```

3.2 Working in a `virtualenv`

3.2.1 Installing

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a virtualenv. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/hack-zope.catalog
```

Next, get this package registered as a “development egg” in the environment:

```
$ /tmp/hack-zope.catalog/bin/python setup.py develop
```

3.2.2 Running the tests

Run the tests using the build-in `setuptools` testrunner:

```
$ /tmp/hack-zope.catalog/bin/python setup.py test
running test
.................
----------------------------------------------------------------------
Ran 17 tests in 0.000s

OK
```

If you have the `nose` package installed in the virtualenv, you can use its testrunner too:

```
$ /tmp/hack-zope.catalog/bin/easy_install nose
...
$ /tmp/hack-zope.catalog/bin/nosetests
.................
----------------------------------------------------------------------
Ran 17 tests in 0.000s

OK
```

If you have the `coverage` pacakge installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/hack-zope.catalog/bin/easy_install nose coverage
...
$ /tmp/hack-zope.catalog/bin/nosetests --with coverage
running nosetests
...................
Name                         Stmts   Miss  Cover   Missing
----------------------------------------------------------
zope/catalog.py                  0      0   100%
zope/catalog/attribute.py       31      0   100%
zope/catalog/catalog.py        125      0   100%
zope/catalog/field.py           10      0   100%
zope/catalog/interfaces.py      22      0   100%
zope/catalog/keyword.py         13      0   100%
zope/catalog/text.py            15      0   100%
----------------------------------------------------------
TOTAL                          216     16   100%
----------------------------------------------------------------------
Ran 19 tests in 0.554s

OK
```

3.2.3 Building the documentation

`zope.catalog` uses the nifty `Sphinx` documentation system for building its docs. Using the same virtualenv you set up to run the tests, you can build the docs:

```
$ /tmp/hack-zope.catalog/bin/easy_install Sphinx
...
$ bin/sphinx-build -b html -d docs/_build/doctrees docs docs/_build/html
...
build succeeded.
```

You can also test the code snippets in the documentation:

```
$ bin/sphinx-build -b doctest -d docs/_build/doctrees docs docs/_build/doctest
...

Doctest summary
===============
  117 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the  \
    results in _build/doctest/output.txt.
```

3.3 Using zc.buildout

3.3.1 Setting up the buildout

`zope.catalog` ships with its own `buildout.cfg` file and `bootstrap.py` for setting up a development buildout:

```
$ /path/to/python2.6 bootstrap.py
...
Generated script '.../bin/buildout'
$ bin/buildout
Develop: '/home/jrandom/projects/Zope/zope.catalog/.'
...
Generated script '.../bin/sphinx-quickstart'.
Generated script '.../bin/sphinx-build'.
```

3.3.2 Running the tests

Run the tests:

```
$ bin/test --all
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Ran 400 tests with 0 failures and 0 errors in 0.366 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
```

3.4 Using `tox`

3.4.1 Running Tests on Multiple Python Versions

tox is a Python-based test automation tool designed to run tests against multiple Python versions. It creates a `virtualenv` for each configured version, installs the current package and configured dependencies into each `virtualenv`, and then runs the configured commands.

`zope.catalog` configures the following `tox` environments via its `tox.ini` file:

- The `py26`, `py27`, `py33`, `py34`, and `pypy` environments builds a `virtualenv` with the appropriate interpreter, installs `zope.catalog` and dependencies, and runs the tests via `python setup.py test -q`.
- The `coverage` environment builds a `virtualenv` with `python2.6`, installs `zope.catalog`, installs `nose` and `coverage`, and runs `nosetests` with statement coverage.
- The `docs` environment builds a virtualenv with `python2.6`, installs `zope.catalog`, installs `Sphinx` and dependencies, and then builds the docs and exercises the doctest snippets.

This example requires that you have a working `python2.6` on your path, as well as installing `tox`:

```
$ tox -e py26
GLOB sdist-make: .../zope.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
..........
----------------------------------------------------------------------
Ran 17 tests in 0.152s

OK
___________________________________ summary ____________________________________
py26: commands succeeded
congratulations :)
```

Running `tox` with no arguments runs all the configured environments, including building the docs and testing their snippets:

```
$ tox
GLOB sdist-make: .../zope.interface/setup.py
py26 sdist-reinst: .../zope.interface/.tox/dist/zope.interface-4.0.2dev.zip
py26 runtests: commands[0]
...
Doctest summary
===============
 117 tests
   0 failures in tests
   0 failures in setup code
   0 failures in cleanup code
build succeeded.
___________________________________ summary ____________________________________
py26: commands succeeded
py27: commands succeeded
py33: commands succeeded
py34: commands succeeded
pypy: commands succeeded
coverage: commands succeeded
docs: commands succeeded
congratulations :)
```

3.5 Contributing to `zope.catalog`

3.5.1 Submitting a Bug Report

`zope.catalog` tracks its bugs on Github:

> https://github.com/zopefoundation/zope.catalog/issues

Please submit bug reports and feature requests there.

3.5.2 Sharing Your Changes

Note: Please ensure that all tests are passing before you submit your code. If possible, your submission should include new tests for new features or bug fixes, although it is possible that you may have tested your new code by updating existing tests.

If have made a change you would like to share, the best route is to fork the Githb repository, check out your fork, make your changes on a branch in your fork, and push it. You can then submit a pull request from your branch:

> https://github.com/zopefoundation/zope.catalog/pulls

If you branched the code from Launchpad using Bazaar, you have another option: you can "push" your branch to Launchpad:

```
$ bzr push lp:~jrandom/zope.catalog/cool_feature
```

After pushing your branch, you can link it to a bug report on Launchpad, or request that the maintainers merge your branch using the Launchpad "merge request" feature.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

z

zope.catalog.attribute, 14
zope.catalog.catalog, 13
zope.catalog.field, 16
zope.catalog.interfaces, 11
zope.catalog.keyword, 16
zope.catalog.text, 17

Index

Symbols

__parent__ (zope.catalog.interfaces.ICatalogIndex attribute), 11
__setitem__() (zope.catalog.interfaces.ICatalog method), 12

A

AttributeIndex (class in zope.catalog.attribute), 14

C

CaseInsensitiveKeywordIndex (class in zope.catalog.keyword), 16
Catalog (class in zope.catalog.catalog), 13

D

default_field_name (zope.catalog.attribute.AttributeIndex attribute), 16
default_interface (zope.catalog.attribute.AttributeIndex attribute), 16

F

field_callable (zope.catalog.interfaces.IAttributeIndex attribute), 13
field_callable (zope.catalog.text.ITextIndex attribute), 17
field_name (zope.catalog.interfaces.IAttributeIndex attribute), 12
field_name (zope.catalog.text.ITextIndex attribute), 17
FieldIndex (class in zope.catalog.field), 16

I

IAttributeIndex (interface in zope.catalog.interfaces), 12
ICatalog (interface in zope.catalog.interfaces), 12
ICatalogEdit (interface in zope.catalog.interfaces), 11
ICatalogIndex (interface in zope.catalog.interfaces), 11
ICatalogQuery (interface in zope.catalog.interfaces), 11
IFieldIndex (interface in zope.catalog.field), 16
IKeywordIndex (interface in zope.catalog.keyword), 16
index_doc() (zope.catalog.attribute.AttributeIndex method), 16
index_doc() (zope.catalog.catalog.Catalog method), 13
indexAdded() (in module zope.catalog.catalog), 13
indexDocSubscriber() (in module zope.catalog.catalog), 13
INoAutoIndex (interface in zope.catalog.interfaces), 13
INoAutoReindex (interface in zope.catalog.interfaces), 13
interface (zope.catalog.interfaces.IAttributeIndex attribute), 12
interface (zope.catalog.text.ITextIndex attribute), 17
ITextIndex (interface in zope.catalog.text), 17

K

KeywordIndex (class in zope.catalog.keyword), 16

R

reindexDocSubscriber() (in module zope.catalog.catalog), 14
ResultSet (class in zope.catalog.catalog), 13

S

searchResults() (zope.catalog.interfaces.ICatalogQuery method), 11

T

TextIndex (class in zope.catalog.text), 17

U

unindex_doc() (zope.catalog.catalog.Catalog method), 13
unindexDocSubscriber() (in module zope.catalog.catalog), 14
updateIndexes() (zope.catalog.interfaces.ICatalogEdit method), 11

Z

zope.catalog.attribute (module), 14
zope.catalog.catalog (module), 13
zope.catalog.field (module), 16
zope.catalog.interfaces (module), 11
zope.catalog.keyword (module), 16
zope.catalog.text (module), 17