

PRINCIPAL ANNOTATIONS

Difficulty

Sprinter

Skills

- You should understand the concept of annotations.
- Be familiar with adapters and how to register them.

Problem/Task

A common task is to append meta-data to principals. However, principals are often imported from external data sources, so that they are not attribute annotatable. Therefore a different solution is desirable. The Principal Annotation service was developed to always allow annotations for a principal. This chapter will show you how to use the Principal Annotation service to store additional data.

Solution

We now know that we want to store additional meta-data for the principal, but what do we want to store? To make it short, let's provide an E-mail address and an IRC nickname. Since we do not want to hand-code the HTML forms, we will describe the two meta-data elements by an interface as usual.

But before we can write the interface, create a new package named `principalinfo` in the `book` package. Do not forget to add the `__init__.py` file.

27.1 Step I: The Principal Information Interface

Add file called `interfaces.py` in the newly created package. Then place the following interface in it.

```

1 from zope.i18n import MessageIDFactory
2 from zope.interface import Interface
3 from zope.schema import TextLine
4
5 _ = MessageIDFactory('principalinfo')
6
7
8 class IPrincipalInformation(Interface):
9     """This interface additional information about a principal."""
10
11     email = TextLine(
12         title=_("E-mail"),
13         description=_("E-mail Address"),
14         default=u"",
15         required=False)
16
17     ircNickname = TextLine(
18         title=_("IRC Nickname"),
19         description=_("IRC Nickname"),
20         default=u"",
21         required=False)

```

The interface is straight forward. the two data elements are simply two text lines. If you wish, you could write a special `E-mail` field that also checks for valid E-mail addresses.

27.2 Step II: The Information Adapter

The next task is to provide an adapter that is able to adapt from `IPrincipal` to `IPrincipalInformation` using the principal annotation service to store the data. In a new module named `info.py` add the following adapter code.

```

1 from persistent.dict import PersistentDict
2 from zope.interface import implements
3 from zope.app import zapi
4
5 from interfaces import IPrincipalInformation
6
7 key = 'book.principalinfo.Information'
8
9 class PrincipalInformation(object):
10     r"""Principal Information Adapter"""
11     implements(IPrincipalInformation)
12
13     def __init__(self, principal):
14         annotationsvc = zapi.getService('PrincipalAnnotation')
15         annotations = annotationsvc.getAnnotations(principal)
16         if annotations.get(key) is None:

```

27.2. THE INFORMATION ADAPTER

```
17         annotations[key] = PersistentDict()
18         self.info = annotations[key]
19
20     def __getattr__(self, name):
21         if name in IPrincipalInformation:
22             return self.info.get(name, None)
23             raise AttributeError, "%s' not in interface." %name
24
25     def __setattr__(self, name, value):
26         if name in IPrincipalInformation:
27             self.info[name] = value
28         else:
29             super(PrincipalInformation, self).__setattr__(name, value)
```

- ▷ Line 7: The key is used to uniquely identify the annotation that is used by this adapter.
- ▷ Line 8: Get the principal annotation service. Note that this code assumes that such a service exists. If not, a `ComponentLookupError` is raised and the initialization of the adapter fails. Luckily, when the ZODB is first generated it adds a principal annotation service to the root site manager.
- ▷ Line 9: Retrieve the set of annotations for the principal that was passed in. Internally, the annotation service uses the principal's id to store the annotations. Therefore it is important that a principal always keep its id or when it is changed, the annotation must be moved.
- ▷ Line 10–11: If the key was not yet registered for the principal, then initialize a persistent dictionary, which we will use to store the values of the fields.
- ▷ Line 12: The persistent data dictionary is set to be available as `info`.
- ▷ Line 14–17: If the name of the attribute we are trying to get is in the `IPrincipalInformation` interface, then retrieve the value from the `info` dictionary. If the name does not corresponds to a field in the interface, then raise an attribute error. Note that `__getattr__` is only called after the normal attribute lookup fails.
- ▷ Line 19–23: Similar to the previous method, if the name corresponds to a field in the `IPrincipalInformation` interface, then store the value in the data dictionary. If not, then use the original `__getattr__()` method to store the value.

This was not that hard, was it?

27.3 Step III: Registering the Components

Now that we have an adapter, we need to register it as such. Also, we want to create an edit form that allows us to edit the values.

```

1 <configure
2     xmlns="http://namespaces.zope.org/zope"
3     xmlns:browser="http://namespaces.zope.org/browser"
4     i18n_domain="principalinfo">
5
6     <adapter
7         factory=".info.PrincipalInformation"
8         provides=".interfaces.IPrincipalInformation"
9         for="zope.app.security.interfaces.IPrincipal"
10        permission="zope.ManageServices"
11    />
12
13    <browser:editform
14        name="userInfo.html"
15        schema=".interfaces.IPrincipalInformation"
16        for="zope.app.security.interfaces.IPrincipal"
17        label="Change User Information"
18        permission="zope.ManageServices"
19        menu="zmi_views" title="User Info" />
20
21 </configure>

```

- ▷ Line 6–11: The adapter is registered for all objects implementing `IPrincipal`. The entire `IPrincipalInformation` schema is available under the `zope.ManageServices` permission, which might not be desirable, but is sufficient for this example. For a real project, you would probably give the accessor a less strict permission than the attribute mutator. This can be done with a `zope:class` directive containing `zope:require` directives.
- ▷ Line 13–19: This edit form is registered for `IPrincipal` components, so that it will be available as a view for all principals. However, the schema that is being edited in `IPrincipalInformation`. The edit form will automatically lookup the adapter from `IPrincipal` to `IPrincipalInformation`.

You need to register the configuration with the Zope 3 framework by adding a file named `principalinfo-configure.zcml` to `package-includes` having the following one line directive.

```

1 <include package="book.principalinfo" />

```

You can now restart Zope 3, and the view should be available.

27.4 Step IV: Testing the Adapter

Before I show you how to use the Web interface to test the view, let's first write a test for the adapter to ensure the correct functioning. The most difficult part about the unit tests here is actually setting up the environment, such as defining and registering a principal annotation service. We will implement the test as a doctest in the `PrincipalInformation`'s doc string.

Let's first setup the environment.

```
1 >>> from zope.app.tests import setup
2 >>> from zope.app.principalannotation.interfaces import \
3 ...     IPrincipalAnnotationService
4 >>> from zope.app.principalannotation import PrincipalAnnotationService
5
6 >>> site = setup.placefulSetUp(site=True)
7 >>> sm = zapi.getGlobalServices()
8 >>> sm.defineService('PrincipalAnnotation',
9 ...                 IPrincipalAnnotationService)
10 >>> svc = setup.addService(site.getSiteManager(), 'PrincipalAnnotation',
11 ...                       PrincipalAnnotationService())
```

- ▷ Line 1: The `setup` module contains some extremely helpful convenience functions.
- ▷ Line 2–3: Import the interface that a principal annotation service must provide.
- ▷ Line 4: Import the implementation of the service.
- ▷ Line 6: Create a placeful setup, making the root folder a site, which is returned.
- ▷ Line 7–9: A new service type can only be defined via the global service manager, so get it first. Then define the service type by name and interface.
- ▷ Line 10–11: Add a principal annotation service to the site of the root folder.

Now that the service is setup, we need a principal to use the adapter on. We could use an existing principal implementation, but every that the principal annotation service needs from the principal is the `id`, which we can easily provide via a stub implementation.

```
1 >>> class Principal(object):
2 ...     id = 'user1'
3 >>> principal = Principal()
```

Now create the principal information adapter:

```
1 >>> info = PrincipalInformation(principal)
```

Before we give the fields any values, they should default to `None`. Any field not listed in the information interface should cause an `AttributeError`.

```

1 >>> info.email is None
2 True
3 >>> info.ircNickname is None
4 True
5 >>> info.phone
6 Traceback (most recent call last):
7 ...
8 AttributeError: 'phone' not in interface.

```

Next we try to set a value for the email and make sure that it is even available if we re instantiate the adapter.

```

1 >>> info.email = 'foo@bar.com'
2 >>> info.email
3 'foo@bar.com'
4
5 >>> info = PrincipalInformation(principal)
6 >>> info.email
7 'foo@bar.com'

```

Finally, let's make sure that the data is really stored in the service.

```

1 >>> svc.annotations['user1']['book.principalinfo.Information']['email']
2 'foo@bar.com'

```

Be careful to clean up after yourself.

```

1 >>> setup.placefulTearDown()

```

To make the tests runnable via the test runner, add the following test setup code to `tests.py`.

```

1 import unittest
2 from zope.testing.doctestunit import DocTestSuite
3
4 def test_suite():
5     return DocTestSuite('book.principalinfo.info')
6
7 if __name__ == '__main__':
8     unittest.main(defaultTest='test_suite')

```

Make sure that the test passes, before you proceed.

27.5 Step V: Playing with the new Feature

Now that the tests pass and the components are configured let's see the edit form. Restart Zope 3. Once restarted, go to `http://localhost:8080/++etc++site/default/manage` and click on "Authentication Service" in the "Add:" box. Once the authentication service is added, go to its "Contents" tab. Click on "Add Principal Source" in the "Add:" box and call it "btree", since it is a b-tree based persistent source. Enter the source's management screen and add a principal with any values. Once you enter the principal, you will see that a tab named "User Info" is available,



The screenshot shows a web application interface with a navigation bar at the top containing 'Edit', 'User Info', and 'Introspector'. Below this is a secondary bar with 'Errors', 'Undo', 'Undo more', 'Undo all', and 'Help'. The main content area is titled 'Change User Information' and shows a timestamp: 'Updated on Aug 25, 2004 10:13:52 PM'. There are two input fields: 'E-mail' with the value 'foo@bar.com' and 'IRC Nickname' with the value 'foobar'. At the bottom are two buttons: 'Refresh' and 'Change'.

Figure 27.1: The principal's "User Info" screen.

which will provide you with the edit form created in this chapter. You can now go there and add the E-mail and IRC nickname of the principal.

Exercises

1. Currently the interface that is used to provide the additional user information is hard-coded. It would be nice, if the user could choose the interface s/he wishes to append as user information. Generalize the implementation, so that the user is asked to input the desired data interface as well.