

---

## CHAPTER 33

---

# WRITING NEW ZCML DIRECTIVES

### *Difficulty*

Sprinter

### *Skills*

- Be familiar using ZCML. If necessary, you should read the “Introduction to ZCML” chapter.
- Have a purpose in mind for creating or extending a namespace.

### *Problem/Task*

As you know by now, we use ZCML to configure the Zope 3 framework, especially for globally-available components. When developing complex applications, it is sometimes very useful to develop new and custom ZCML directives to reduce repetitive tasks or simply make something configurable that would otherwise require Python code manipulation. This chapter will implement a small `browser:redirect` directive that defines a view that simply redirects to another URL.

### *Solution*

## 33.1 Introduction

One of the major design goals of ZCML was to make it very easy for a developer to create a new ZCML directives. We differentiate between simple and complex

directives. Simple directives consist of one XML element that causes a set of actions. Complex directives are wrapper-like directives that can contain other sub-directives. They usually do not cause actions themselves, but provide data that is applicable to most of the sub-directives. In this chapter, however, we will just create a simple directive; the complex ones are not much more difficult.

There are three simple steps to implementing a directive. First, you develop the directive's schema, which describes the attributes the XML element can and must have. Like with any schema, you can specify, whether an attribute is required or not. When the XML is parsed, the unicode values that are returned from the parser are automatically converted to Python values as described by the field. Besides the common fields, such as `TextLine` or `Int`, you also have special configuration fields, such as `GlobalObject`, which automatically converts a Python reference to a Python object. A full list of additional fields is provided in the "Introduction to ZCML" chapter. Directive schemas are commonly placed in a file called `metadirectives.py`.

The second step is to develop a handler for the directive, which is for simple directives a function taking the attributes as arguments. The first attribute of the handler is the `context` of the configuration. If you have a complex directive, the handler is usually a class, where the constructor takes the attributes of the directive as arguments. Each sub-directive is then a method on the class. The class must also be callable, so that it can be called when the complex directive is closed. It is very important to note, that the directives should *not perform any actions*, but only declare the actions as we will see later. This way the configuration mechanism can detect configuration conflict. By convention the handlers are stored in `metaconfigure.zcml`.

Once the directive schema and handler are written, we can now register the ZCML directive using the ZCML `meta` namespace, which is usually done in a configuration file named `meta.zcml`. The meta-configuration file is then registered in `packages-includes` using a filename like `<package>-meta.zcml`.

Now that you have an overview over the necessary tasks, let's get our hands dirty. As mentioned before the goal is to provide a directive that creates a view that makes a simple redirect. A view must always be defined for a particular object and needs a name to be accessible under. We should also optionally allow a layer to be specified. Usually, we also want to specify a permission, but since this view just redirects to another we simply make the view public. The final attribute we need for the directive is the `url`, which specifies the URL we want to direct to. so, first create a package named `redirect` in `ZOPE3/src/book/` (and don't forget about `__init__.py`).

## 33.2 Step I: Developing the Directive Schema

In a new file named `metadirectives.py` add the following schema.

```
1 from zope.interface import Interface
2 from zope.configuration.fields import GlobalObject
3 from zope.schema import TextLine
4
5 class IRedirectDirective(Interface):
6     """Redirects clients to a specified URL."""
7
8     name = TextLine(
9         title=u"Name",
10        description=u"The name of the requested view.")
11
12    for_ = GlobalObject(
13        title=u"For Interface",
14        description=u"The interface the directive is used for.",
15        required=False)
16
17    url = TextLine(
18        title=u"URL",
19        description=u"The URL the client should be redirected to.")
20
21    layer = TextLine(
22        title=u"Layer",
23        description=u"The layer the redirect is defined in.",
24        required=False)
```

- ▷ Line 2: As you can see, all configuration-specific fields, like `GlobalObject` are defined in `zope.configuration.fields`.
- ▷ Line 3: However, you can also use any of the conventional fields as well.
- ▷ Line 5: The directive schemas are just schemas like any other ones. There is no special base-class required.
- ▷ Line 12: Whenever you have an attribute whose name is a Python keyword, then simply add an underscore behind it; the underscore will be safely ignored during runtime.

## 33.3 Step II: Implementing the Directive Handler

The handler is added to `metaconfigure.zcml`.

```
1 from zope.app.publisher.browser.viewmeta import page
2
3 class Redirect(object):
4     """Redirects to a specified URL."""
5     url = None
6
7     def __call__(self):
```

```

8         self.request.response.redirect(self.url)
9
10
11 def redirect(_context, name, url, for_=None, layer='default'):
12     # define the class that performs the redirect
13     redirectClass = type(str("Redirect %s for %s to '%s'" %(name, for_, url)),
14                          (Redirect,), {'url' : url})
15
16     page(_context, name, 'zope.Public', for_, layer, class_=redirectClass)

```

- ▷ Line 1: Since we are just defining a new page, why not reuse the page-directive handler? This makes the implementation of our handler much simpler.
- ▷ Line 3–8: This is the base view class. We simply allow a URL to be set on it. When the view is called, we simply redirect the HTTP request. There is no need to implement `IBrowserPublisher` or `IBrowserView` here, since the `page()` function will mix-in all of these APIs plus implementation.
- ▷ Line 11–16: This is the actual handler of the directive. The first step is to create a customized version of the view class by merging in the URL (line 13). Then we simply call the page-directive handler, where we use the public permission. The page-directive handler hides a lot of the glory details of defining a full-blown view, including creating configuration actions.

An action is created by calling `_context.action()`). This function supports the following arguments:

- **discriminator** – This is a unique identifier that is used to recognize a particular action. It is very important that no two actions have the same discriminator when starting Zope. This allows us to use the discriminator for conflict resolution and spotting duplicate actions. It is usually a tuple.
- **callable** – Here we specify the callable (usually a method or function) that is called when the action is executed.
- **args & kw** – Arguments and keywords that are passed to the callable as arguments on execution time.

## 33.4 Step III: Writing the Meta-Configuration

Now that we have all pieces of the directive, let's register it in `meta.zcml`.

```

1 <configure
2     xmlns="http://namespaces.zope.org/meta">
3
4     <directives namespace="http://namespaces.zope.org/browser">
5         <directive
6             name="redirect"

```

## 33.5. TESTING THE DIRECTIVE

```
7     schema=".metadirectives.IRedirectDirective"
8     handler=".metaconfigure.redirect"
9     />
10 </directives>
11
12 </configure>
```

- ▷ Line 2: The `meta` namespace is used to define new ZCML directives.
- ▷ Line 4 & 10: The `meta:directives` directive is used to specify the namespace under which the directives will be available. In our case it is the `browser` namespace.
- ▷ Line 5–9: The `meta:directive` directive is used to register a simple directive. The `name` is the name as which the directive will be known/accessible. The `schema` specified the directive schema and the `handler` the directive handler, both of which we developed before.

If you develop a complex directive, you would use the `meta:complexDirective` directive, which supports the same attributes. Inside a complex directive you can then place `meta:subdirective` directives, which define the sub-directives of the complex directive. You might want to look into the “Registries with Global Utilities” chapter for an example of a relatively simple complex directive.

You now have to register the new directive with the Zope 3 system by placing a file named `redirect-meta.zcml` in `package-includes`. It should have the following content:

```
1 <include package="book.redirect" file="meta.zcml" />
```

The next time you restart Zope, the directive should be available.

## 33.5 Step IV: Testing the Directive

The best way to test the directive is to use it. Let’s have the view “manage.html” be redirected to “manage” for all folders. In a new configuration file, `configure.zcml`, add the following directives:

```
1 <configure
2     xmlns="http://namespaces.zope.org/browser">
3
4     <redirect
5         name="manage.html"
6         for="zope.app.folder.interfaces.IFolder"
7         url="manage" />
8
9 </configure>
```

- ▷ Line 2: As specified, the `redirect` directive is available via the `browser` namespace.
- ▷ Line 5: The name is the view that must be called to initiate the redirection.
- ▷ Line 6: The redirection will only be available for folders.
- ▷ Line 7: The target URL is the relative name “manage”.

After adding `redirect-configure.zcml` containing

```
1 <include package="book.redirect" />
```

to `package-includes`, restart Zope 3. You should now be able to call the URL `http://localhost:8080/@manage.html`, which should bring you to `http://localhost:8080/@contents.html`, since “manage” just redirects to “contents.html”.

This functionality can be easily duplicated in a functional test. Put the following test case into a file named `ftests.py`:

```
1 import unittest
2
3 from zope.app.tests.functional import BrowserTestCase
4
5
6 class Test(BrowserTestCase):
7
8     def test_RedirectManageHtml(self):
9         response = self.publish("/manage.html")
10
11         self.assertEqual(response.getStatus(), 302)
12         self.assertEqual(response.getHeader('Location'), 'manage')
13
14
15 def test_suite():
16     return unittest.makeSuite(Test)
17
18 if __name__=='__main__':
19     unittest.main(defaultTest='test_suite')
```

If you are not familiar with the `BrowserTestCase` API, I suggest you read the “Writing Functional Tests” chapter. Otherwise the test is straightforward and you can execute in the usual manner.