Plone Conference, October 15, 2003

Introduction to Python for Plone developers

Jim Roepcke
Tyrell Software Corporation

What we will learn

- Python language basics
- Where you can use Python in Plone
- Examples of using Python in Plone
- Python Resources

What is Python?

- Python is a “strongly but dynamically typed” object-oriented programming language
- Automatic garbage collection
- Everything is an object
- No semi-colons or braces unlike C/C++/Java
  - Blocks are indented
- comments begin with #

Demonstration

- A simple Python program, as it would appear if run in a command-line python interpreter:

```
>>> words = ('Hello', 'World')
>>> sep = ', '
>>> print '%s!' % sep.join(words)
Hello, World!
```

- This program joins two words with a comma and a space, which are printed with an exclamation point following them

Variables

- Do not have to be declared or typed
- Must be assigned to before being used
- Follow standard identifier naming rules
  eg: **`foo3`** is valid, **`3foo`** is not, **`_3foo`** is valid
- Refer to objects, do not "contain" them
  - Objects have a type, variables do not
- **`None`** is the null object, is false in truth tests

Assignment

- Assignment copies the reference, not the value
  - Let **`x`** refer to an object, the number **`3`**
    ```
    x = 3
    ```
  - Let **`x`** and **`y`** refer to what **`z`** refers to
    ```
    x = y = z
    ```
  - Let **`x`** and **`y`** refer to what **`i`** and **`j`** refer to
    ```
    x, y = i, j
    ```

Variables refer to objects

```
>>> x = [5, 10, 15]
>>> y = x
>>> print y
[5, 10, 15]
>>> del x[1]
>>> print y
[5, 15]
```

- Remember, assignment only copies the reference to the object, not the object itself!

Organizing source code

- Python source code is organized into modules
- Modules contain statements, functions, and classes
- Modules have a .py file extension
- Modules are compiled at runtime into .pyc files
- Modules can be organized into packages
- Packages can contain packages

Importing modules

- the **`import`** statement imports a module or attribute from a module so that you can use it in your program

- Importing a module and using it:

```
import rfc822
from = 'Joe <joe@doe.com>'
name, addr = \
   rfc822.parseaddr(from)
```

Importing attributes from modules

- If you want, you can import particular attributes from a module

```
from math import sqrt, pi, sin
rootOfX = sqrt(4)
```

Numbers and Math

- Integer: `0   -1   4`
- Long Integer: `0L   4L   -3L`
- Floating point: `2.5   5.   1.0e100`
- Complex: `2.5j   5j   1e10j`

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Power	`**`
Division	`/`
Floor Division	`//`
Modulo	`%`

Math Example

```
from math import sqrt
x, y = 9, 16
xs, ys = (x, sqrt(x)), (y, sqrt(y))
template = 'The sqrt of %d is %d.'
print template % xs
print template % ys
```

Math Example: output

```
The sqrt of 9 is 3.
The sqrt of 16 is 4.
```

Strings are Sequences

- String can be single quoted or double quoted
```
>>> str = 'this' + " and " + 'that'
```
- Get a single character by *indexing*:
```
>>> str[0]
't'
```
- Get a substring by *slicing*
```
>>> str[1:4]
'his'
```

String methods

http://www.python.org/doc/current/lib/string-methods.html

`capitalize, center, count, encode, endswith, expandtabs, find, index, isalnum, isalpha, isdigit, islower, istitle, isupper, join, ljust, lower, lstrip, replace, rfind, rindex, rjust, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper`

Tuples are Sequences

- Empty tuple: **`()`**
- One item tuple: **`(6,)`**
- Multiple items: **`('a', 'b', 3, (9, ''))`**
- Use indexing and slicing to access contents
- Tuples cannot be modified
- Use **`+`** to concatenate tuples

Lists are Sequences

- Empty List: `[]`
- One item List: `[6]`
- Multiple item List: `[1, 3, [4, 6], '10']`
- Use indexing and slicing to access contents
- **`append()`** to add item, **`del`** to remove item
- Use **`+`** to concatenate lists

Dictionaries are Mappings

- Dictionaries hold key/value pairs

  ```
  emptyDict = {}
  oneKey = {'id':'document_view'}
  car = {'make':'Saab', 'year':1999}
  ```

- Accessing a key: `car['year']`
- Assigning a key: `car['color'] = 'red'`
- Removing a key: `del car['make']`

Getting keys and values out

- You can get lists of keys and values from dictionaries using the **`keys()`** and **`values()`** methods
- You can get a list of tuples containing the keys and values from a dictionary using the **`items()`** method

len is short for length

- To get the number of items in a sequence or mapping such as a string, tuple, list, or dictionary, use the built-in **`len()`** function

```
>>> print len('abc')
3
>>> print len(['foo', 'bar'])
2
>>> print len ({})
0
```

Flow Control

```
if expression is true:
    ...
elif expression2 is true:
    ...
else:
    ...
```

```
while expression is true:
    ...
else:
    ...
```

```
for item in sequence:
    ...
else:
    ...
```

Branching example

```
x = 1
y = 2

if (x == y):
    print 'x equals y'
elif (x > y):
    print 'x is greater than y'
else:
    print "D'oh! I give up!"
```

Branching example: output

```
D'oh! I give up!
```

Looping example

```
numberOfLoops = 3
ct = 0

while (ct < numberOfLoops):
    ct = ct + 1
    print 'Loop # %d' % ct
else:
    print 'Finished!'
```

Looping example: output

```
Loop # 1
Loop # 2
Loop # 3
Finished!
```

Sequence iteration example

```
words = ['green','eggs','and','ham']
sentence = words[0].capitalize()
remainingWords = words[1:]

for word in remainingWords:
    sentence += ' ' + word
else:
    print '%s.' % sentence
```

Sequence iteration example: output

```
Green eggs and ham.
```

Another example

```
car = {'make':'Ford'}
car['model'] = 'Focus'
car['year'] = '2002'

print '--- one way ---'
for key, value in car.items():
    print '%s: %s' % (key, value)

print '--- same thing ---'
for item in car.items():
    print '%s: %s' % item
```

Another example: output

```
--- one way ---
make: Ford
model: Focus
year: 2002
--- same thing ---
make: Ford
model: Focus
year: 2002
```

Truth Tests

- False: `0   None   (len(x) == 0)`
- True: non-zero Numbers, `(len(x) != 0)`
- Comparison: `==   !=   <   <=   >=   >`
- Identity: `is   is not`
- Membership: `in   not in`

Boolean operators

- **`and  or  not`**
- Return one of the operands rather than a true or false value
    - The net effect is the same, however
- Can be used to simulate a ternary operator
    - Java: **`answer = x ? y : z;`**
    - Python: **`answer = (x and y) or z`**

Boolean operator behavior

x or *y*	if *x* is false then *y*, else *x*
x and *y*	if *x* is false then *x*, else *y*
not *x*	if *x* is false, then `1`, else `0`

Truth test examples

```
>>> 's' and 3
3
>>> '' and (1,2,3)
''
>>> 's' and {}
{}
>>> not [] and 's'
's'
>>> not ([] and 's')
1
```

Functions

- the **`def`** statement creates a function

```
def sum(a1, a2):
    if type(a1) == type(a2):
        return a1 + a2
    return None # not very smart
```

- arguments can take default values

```
def concat(s1, s2, s3 = '', s4 = ''):
    return s1 + s2 + s3 + s4
```

  - `concat('a', 'b')` returns `'ab'`

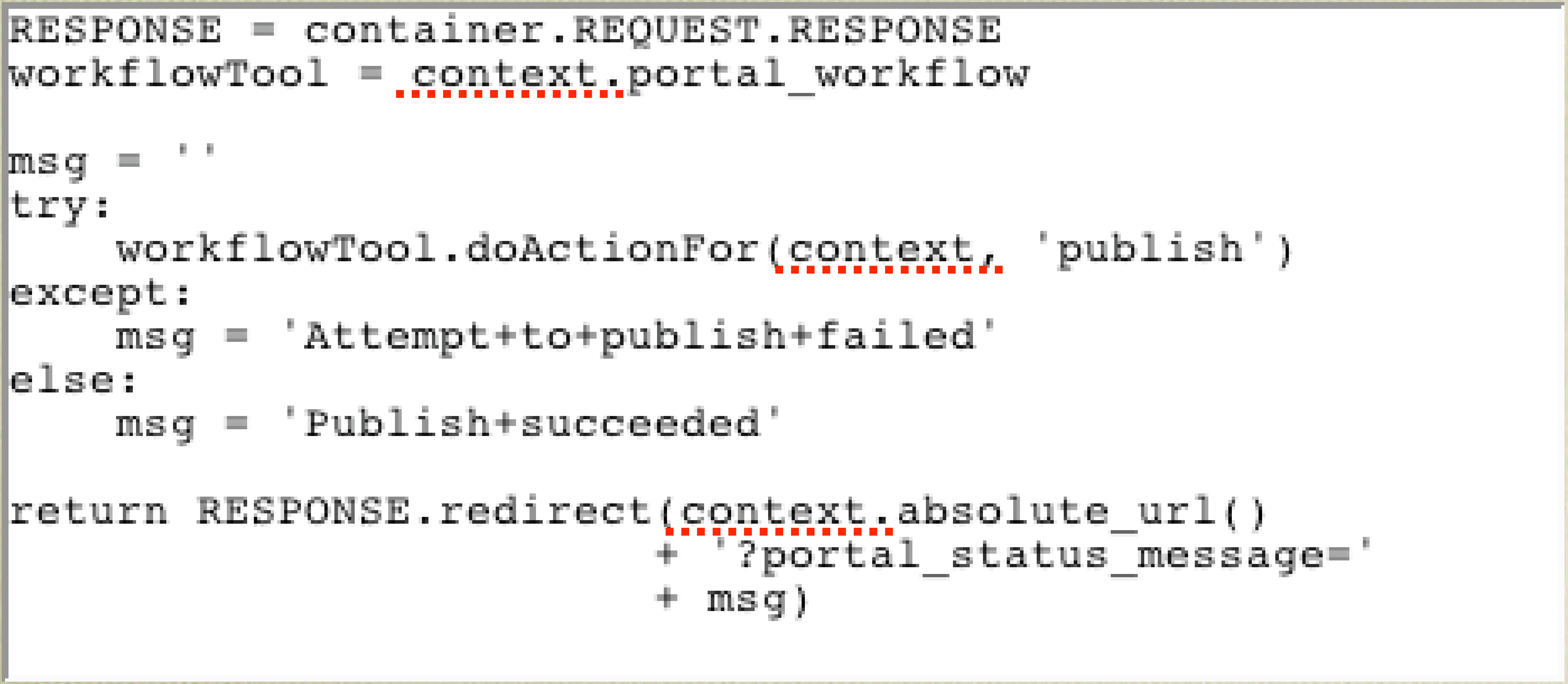
Where you can use Python in Plone, with examples

- Script (Python) objects in Zope
- Zope Page Templates
- CMF Actions
- Expressions in DCWorkflow transitions, variables and worklists
- External Methods
- Custom products

Writing Python scripts in “Script (Python)” objects

- Create a “Script (Python)” object in the *Plone/portal_skins/custom* folder using the ZMI
- The script is a method callable on an object through Python, or through the web
  - **`someObject.someScript(foo='bar')`**
  - *http://site/someObject/someScript?foo=bar*
- In the script, **`context`** refers to **`someObject`**.

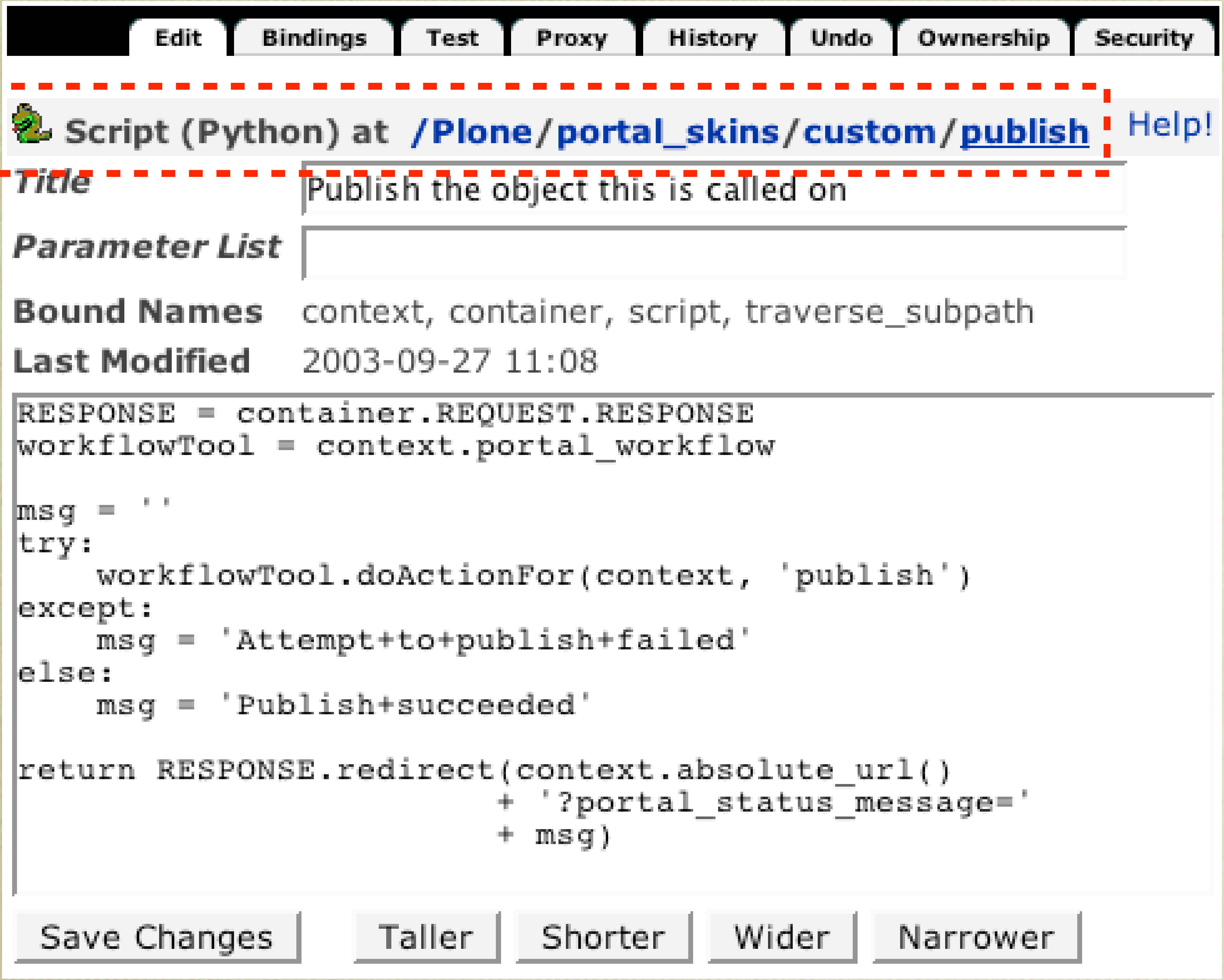
Example Script (Python) that publishes **`context`**



```
RESPONSE = container.REQUEST.RESPONSE
workflowTool = context.portal_workflow

msg = ''
try:
    workflowTool.doActionFor(context, 'publish')
except:
    msg = 'Attempt+to+publish+failed'
else:
    msg = 'Publish+succeeded'

return RESPONSE.redirect(context.absolute_url()
                        + '?portal_status_message='
                        + msg)
```



Script (Python) at /Plone/portal_skins/custom/publish

Title	Publish the object this is called on
Parameter List	
Bound Names	context, container, script, traverse_subpath
Last Modified	2003-09-27 11:08

```
RESPONSE = container.REQUEST.RESPONSE
workflowTool = context.portal_workflow

msg = ''
try:
    workflowTool.doActionFor(context, 'publish')
except:
    msg = 'Attempt+to+publish+failed'
else:
    msg = 'Publish+succeeded'

return RESPONSE.redirect(context.absolute_url()
                        + '?portal_status_message='
                        + msg)
```

Zope Page Templates with TAL & TALES

- Page templates are used to implement all of the web pages in Plone's user interface
- You can use Python expressions in TAL statements in a Zope Page Template object
  - put **`python:`** before the Python expression
- Useful for calling methods that require parameters, using boolean operators, accessing modules like **`Batch`**, **`DateTime`**

What is available to a ZPT's TAL **`python:`** expression?

- **`here`** is the object the template is applied to; the same meaning as Script (Python)'s **`context`**
- **`template`** is the Page template object itself
- **`container`** is the folder the **`template`** is in
- **`request`** is the Zope request object
- **`user`** is the Zope user object for the person requesting the page template
- Refer to Zope Book 2.6 Edition Appendix C

Some **`python:`** expressions

- `<b tal:content="python: here.getId()">`
  `id of the object being rendered`
  `</b>`

- `python: here.getTypeInfo().Title()`
  - Note: `here/getTypeInfo/Title` works too, without the `python:` prefix

- `python: request.get('id', here.getId())`

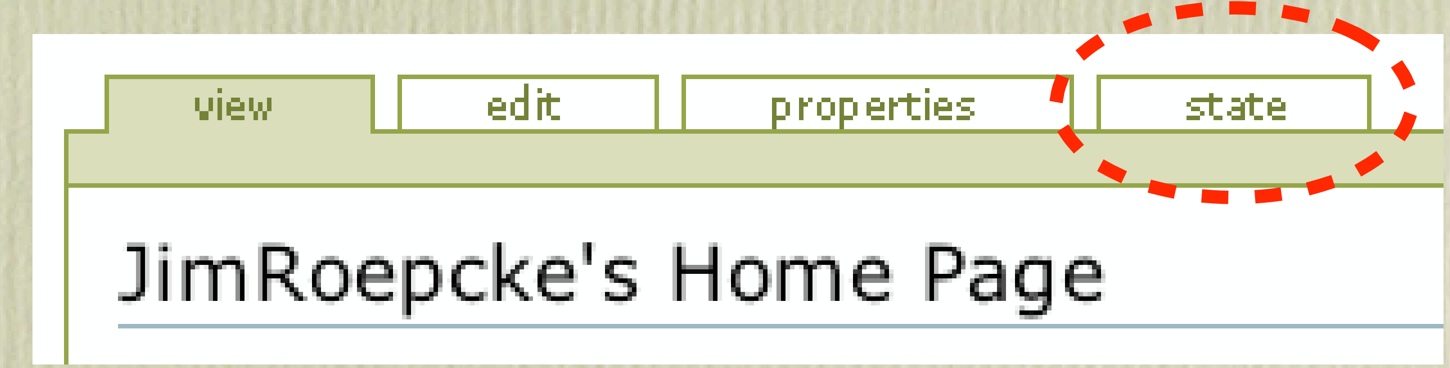
- `python: d and d.Title() or 'Untitled'`

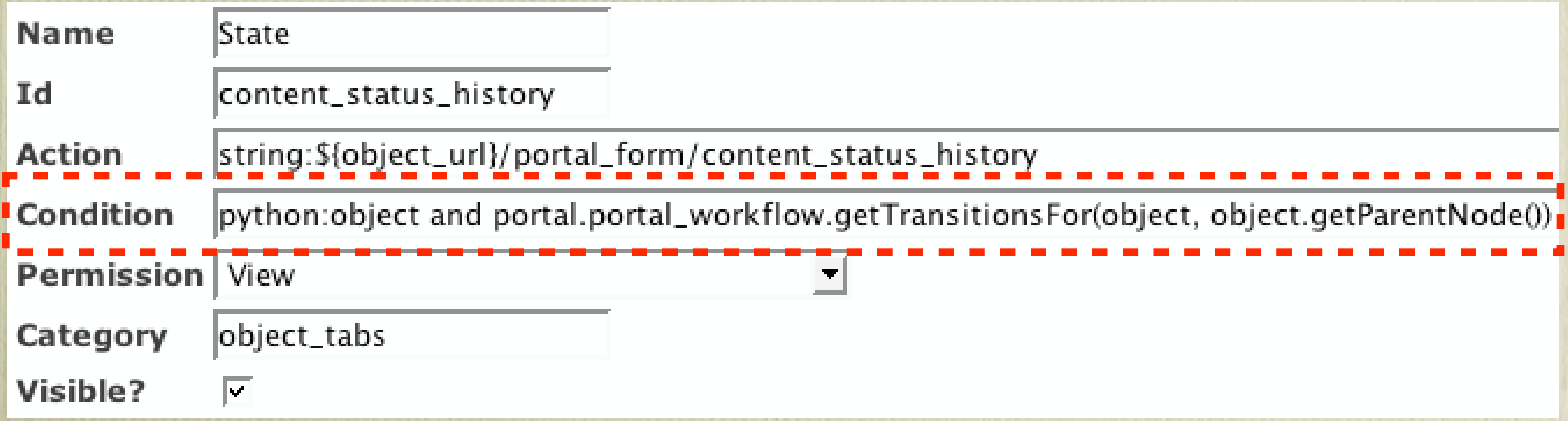
CMF Actions; Action conditions

- Actions configure visibility of URLs linking to functionality in a CMF/Plone site
- Defined in *Plone/portal_actions* and other tools
- Displayed as tabs, personal bar links, and more
- An action may define a condition expression; if false, the action is not shown to the user
- The condition is a TALES expression; you can use **`python:`** just as in Zope Page Templates

Example from Plone 1.0.5

- The State tab should only appear if there are workflow transitions available





Name	State
Id	content_status_history
Action	string:${object_url}/portal_form/content_status_history
Condition	python:object and portal.portal_workflow.getTransitionsFor(object, object.getParentNode())
Permission	View
Category	object_tabs
Visible?	✓

What is available to a CMF Action condition expression?

- **`object`** is the object (a Document or News Item, for example) being rendered, usually the object the action would apply to
- **`folder`** is the folder that **`object`** is in
- **`portal`** is the Plone portal object, which is the root of the Plone site

Transition, Variable and Worklist expressions

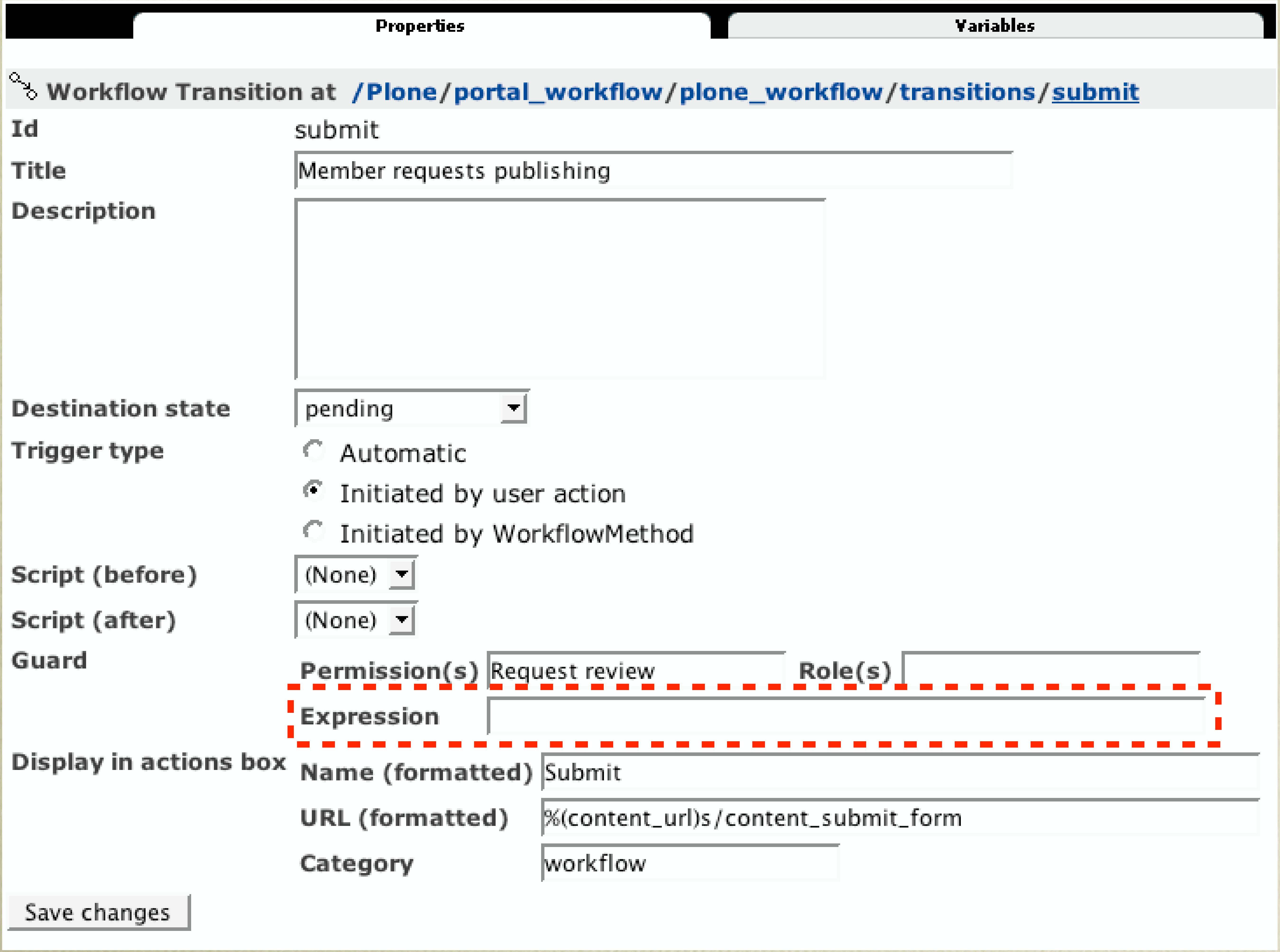
- TALES expressions are used throughout *Plone/portal_workflow* to set variables' values, guard transitions and more
- If a guard expression evaluates to false, the object the guard is protecting will not be available to the current user
- You can use **`python:`** in these expressions
- An easy way to customize Plone's behavior

What is available to a workflow expression?

- **`here`** is the object being acted on
- **`container`** is the folder that **`here`** is in
- **`nothing`** is a reference to **`None`**
- **`user`** is the current user
- **`state_change`** contains the **`old_state`**, **`new_state`**, and more
- **`transition`** is the transition object being executed

What is available to a workflow expression?

- **`request`** is the request being processed
- **`modules`** contains Python modules you can use
- **`root`** is the root folder of the entire ZODB
- **`status`** contains the most recent entry in the workflow history of **`here`**
- **`workflow`** is the workflow definition being processed for **`here`**
- **`scripts`** is the *scripts* folder in the **`workflow`**

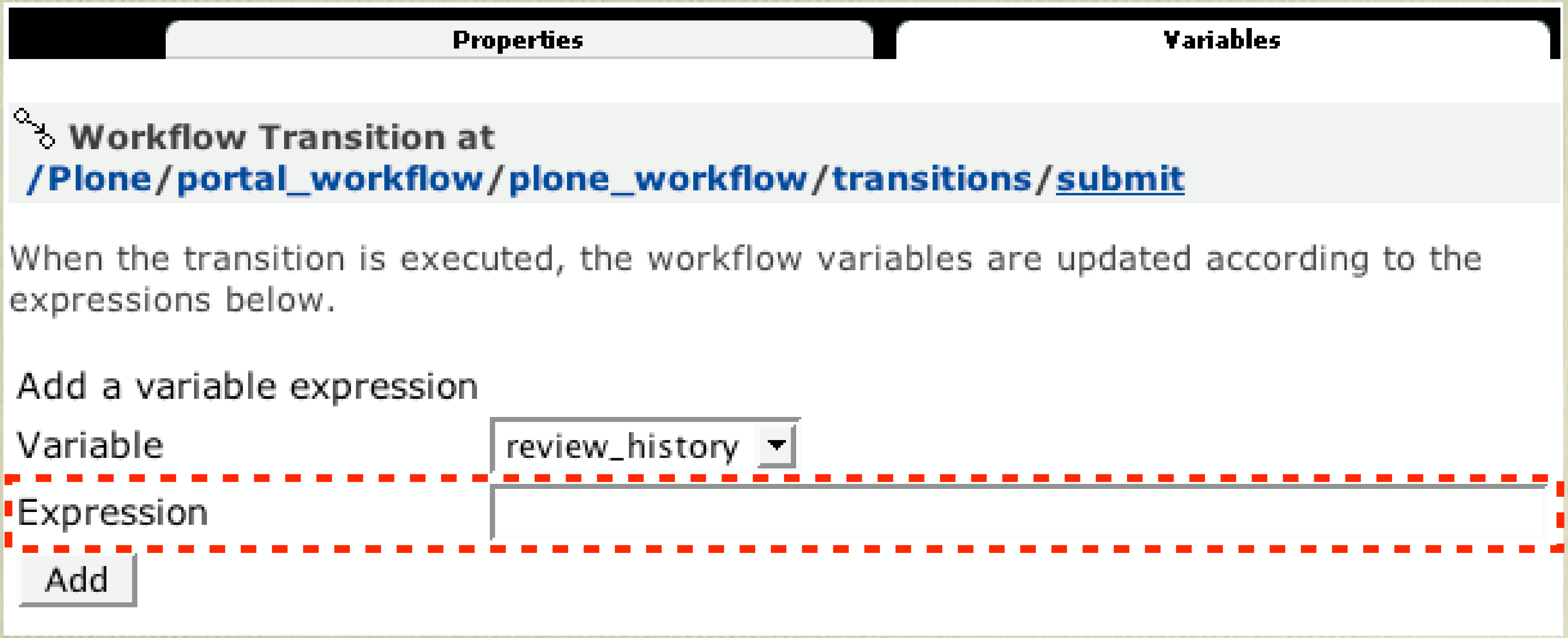


Properties | Variables

Workflow Transition at /Plone/portal_workflow/plone_workflow/transitions/submit

Id	submit
Title	Member requests publishing
Description	
Destination state	pending
Trigger type	Automatic [✓ Initiated by user action] Initiated by WorkflowMethod
Script (before)	(None)
Script (after)	(None)
Guard	Permission(s) Request review Role(s) Expression
Display in actions box	Name (formatted) Submit URL (formatted) %(content_url)s/content_submit_form Category workflow

Save changes

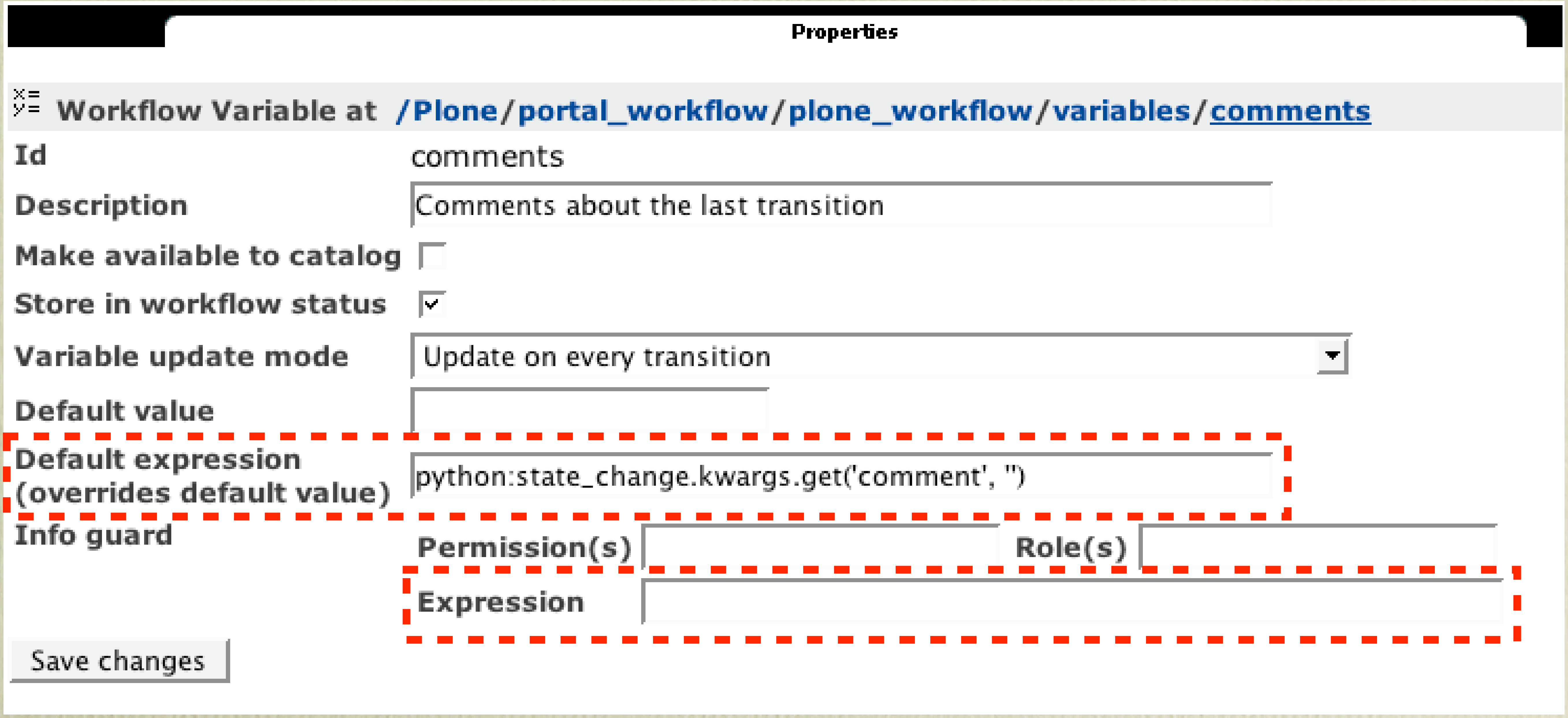


Workflow Transition at /Plone/portal_workflow/plone_workflow/transitions/submit

When the transition is executed, the workflow variables are updated according to the expressions below.

Add a variable expression

Variable	review_history
Expression	

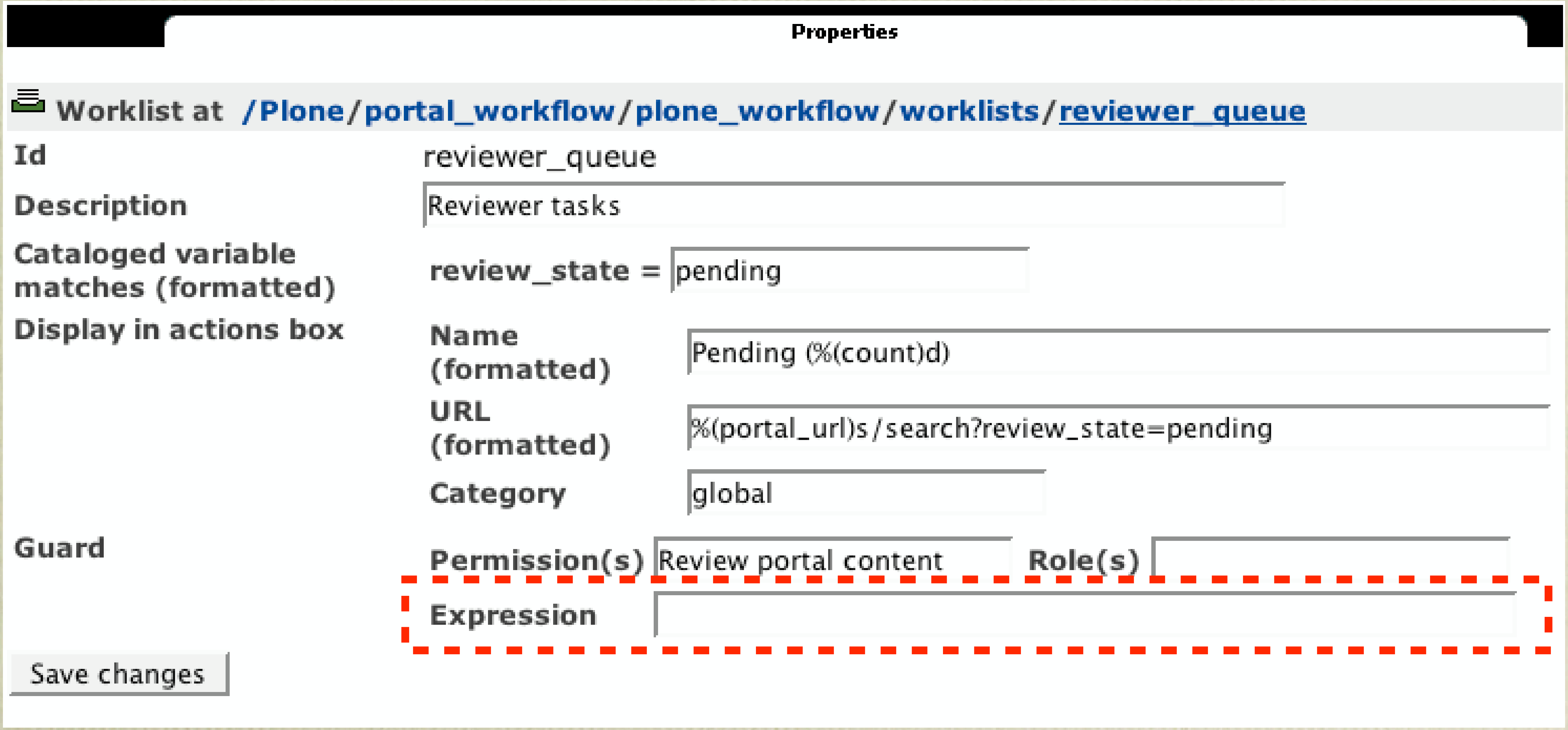


Properties

Workflow Variable at /Plone/portal_workflow/plone_workflow/variables/comments

Id	comments
Description	Comments about the last transition
Make available to catalog	☐
Store in workflow status	✓
Variable update mode	Update on every transition
Default value	
Default expression (overrides default value)	python:state_change.kwargs.get('comment', '')
Info guard	Permission(s) Role(s)
	Expression

Save changes



Properties

Worklist at /Plone/portal_workflow/plone_workflow/worklists/reviewer_queue

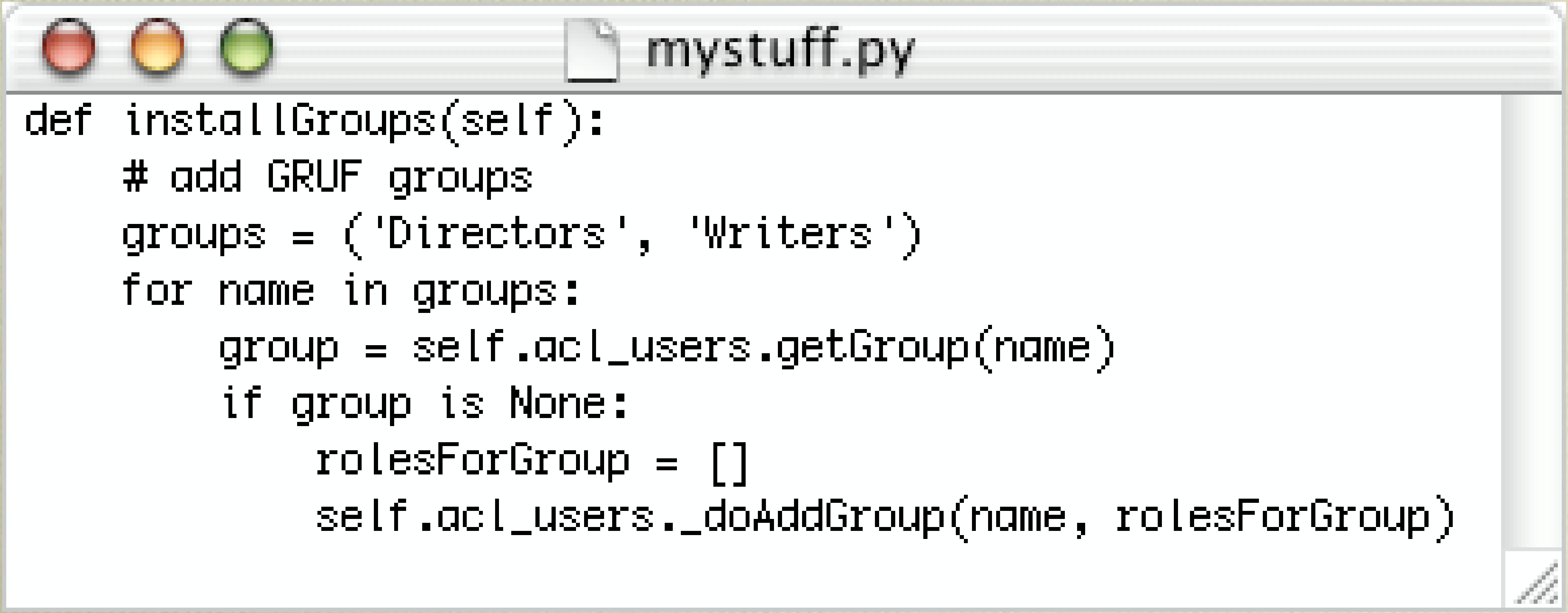
Id	reviewer_queue	
Description	Reviewer tasks	
Cataloged variable matches (formatted)	review_state =	pending
Display in actions box	Name (formatted)	Pending (%(count)d)
	URL (formatted)	%(portal_url)s/search?review_state=pending
	Category	global
Guard	Permission(s)	Review portal content
	Role(s)	
	Expression	

External Methods

- External methods are not subject to restricted execution security rules
- Place a Python module, like **mystuff.py**, in **$(INSTANCE_HOME)/Extensions** on the filesystem
  - The module should contain one or more functions that take a single argument
- Create an External Method object in ZODB using the ZMI

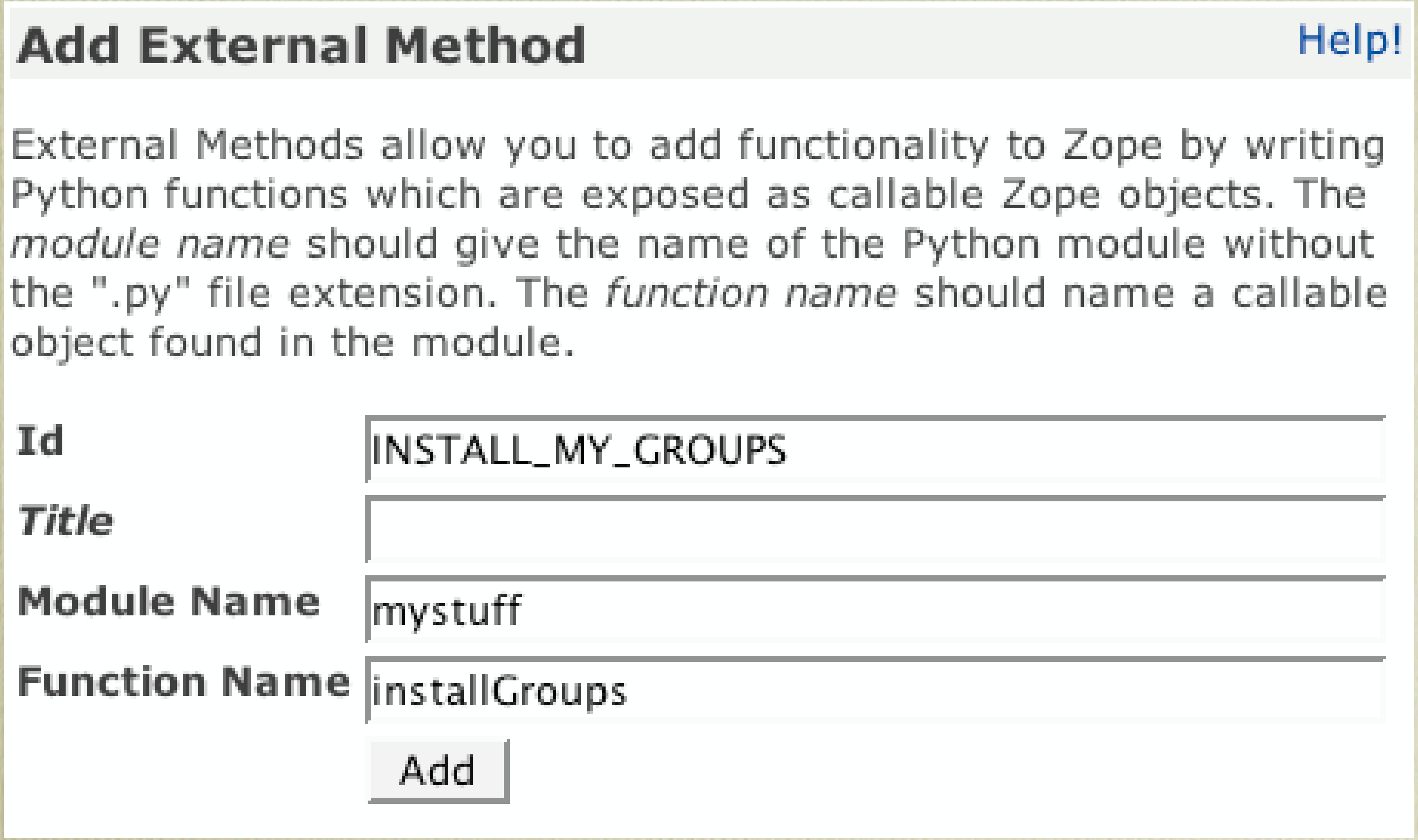
A useful example of an External Method’s module

- The **`installGroups`** function creates groups in a GroupUserFolder



```
def installGroups(self):
    # add GRUF groups
    groups = ('Directors', 'Writers')
    for name in groups:
        group = self.acl_users.getGroup(name)
        if group is None:
            rolesForGroup = []
            self.acl_users._doAddGroup(name, rolesForGroup)
```

Creating an External Method

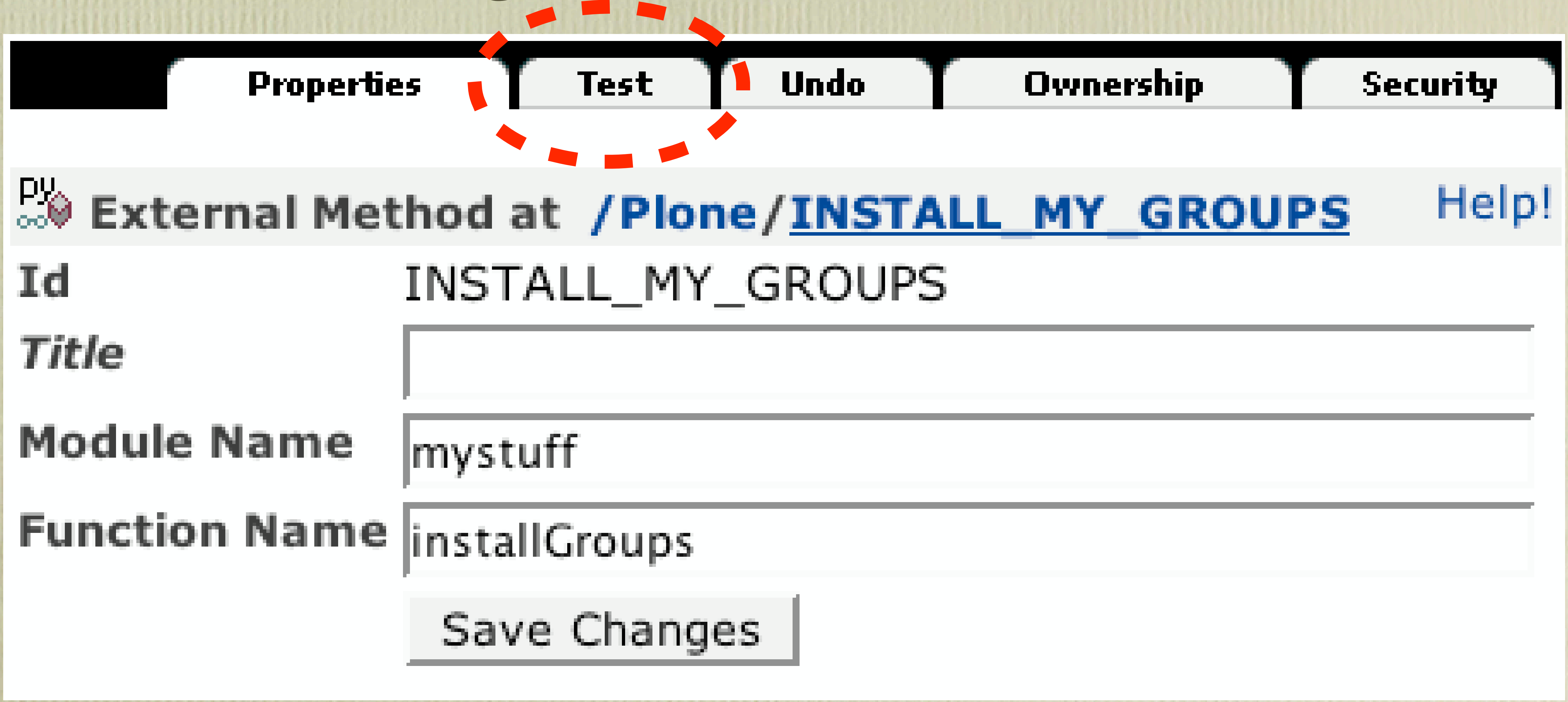


Add External Method Help!

External Methods allow you to add functionality to Zope by writing Python functions which are exposed as callable Zope objects. The *module name* should give the name of the Python module without the ".py" file extension. The *function name* should name a callable object found in the module.

Id	INSTALL_MY_GROUPS
Title	
Module Name	mystuff
Function Name	installGroups

Running an External Method



External Method at /Plone/INSTALL_MY_GROUPS

Id	INSTALL_MY_GROUPS
Title	
Module Name	mystuff
Function Name	installGroups

Custom Products

- Reusable packages of functionality
- Installed in **$(INSTANCE_HOME)/Products**
- Initialized in the **__init__.py** module, can contain other modules, packages, resources
- Much of Zope's functionality is packaged in products
- CMF and Plone are collections of products

Python Resources

- http://python.org/
- http://python.org/doc/2.1.3/
- http://python.org/doc/Intros.html
- http://diveintopython.org/
- http://directory.google.com/Top/Computers/Programming/Languages/Python/
- http://plone.org/documentation/python

Script (Python) Resources

- http://zope.org/Documentation/Books/ZopeBook/2_6Edition/ScriptingZope.stx
- Zope's Help System from the ZMI

Zope Page Templates Resources

- http://zope.org/Documentation/Books/ZopeBook/2_6Edition/ZPT.stx
- http://zope.org/Documentation/Books/ZopeBook/2_6Edition/AdvZPT.stx
- http://zope.org/Documentation/Books/ZopeBook/2_6Edition/AppendixC.stx
- Zope's Help System from the ZMI

CMF Action Resources

- http://plone.org/documentation/book/5

External Method Resources

- http://www.zope.org/Documentation/How-To/ExternalMethods
- http://www.zope.org/Members/pbrunet/ExternalMethodsBasicSummary
- Zope’s Help System from the ZMI

DCWorkflow Resources

- http://www.zope.org/Members/hathawsh/DCWorkflow_docs
- http://plone.org/documentation/book/4
- Zope's Help System from the ZMI

Product Resources

- http://www.zope.org/Documentation/Books/ZDG/current/Products.stx
- http://sf.net/projects/collective

Help from the community

- irc.freenode.net
  - #plone
  - #zope
  - #python
- http://plone.org/development/lists
- http://www.zope.org/Resources/MailingLists

Any Questions?

Thank you for coming!

Please send your feedback to:

Jim Roepcke <jimr@tyrell.com>