

Simplistix

Zope User Folders

An epic journey, hopefully with a happy ending!

Chris Withers

Who am I?

- Chris Withers
- Independent Zope and Python Consultant
- Using Zope and Python since 1999

- What do I use Zope and Python for?
 - Content Management
 - Systems Integration
 - XML manipulation

Pre-requisites

- Knowledge of Python
- Working knowledge of Zope

In The Beginning...

...there was HTTP

- HTTP is stateless!
 - Great!
 - restart servers
 - easily cluster
 - very scalable

The Deepest Pits of Hell

- ...aka HTTP Authentication
- HTTP is stateless!
 - Argh!
 - have to send authentication credentials with every request
 - client HAS to be trusted to do the right thing
 - clients DON'T do the right thing
 - cookies are a HACK!

The Options

– “Basic” Authentication

- insecure
 - unencrypted credentials sent in every request
- no logout

– Cookies

- insecure
 - cookie theft
 - session theft

More Options

- Basic Authentication over SSL
 - the “joys” of https://
 - certificates
 - compression
 - debugging
 - no logout
- Secure Cookies
 - Do clients respect the “secure” bit?
 - Where do clients store the cookie?

Taming the Dragon

- The default UserFolder
 - Users stored in ZODB
 - HTTP Basic Authentication
 - Users managed through ZMI

- No need for use in scripts
 - everyone uses ZMI
 - or writes a Python Product to do things

The Dark Times...

- ZODB isn't always great to store users
 - Relational Database
 - LDAP Repository
 - Operating System Users
- Alternatives to Basic Authentication
 - Certificates
 - Microsoft Passport
- The need to manage users from scripts
 - Not everyone has ZMI access

Coping with History

- The Appearance of Custom User Folders
 - Default UserFolder becomes API
- SMBUserFolder
- MySQLUserFolder
- etc

The UserFolder "API"

- replace methods by subclassing UserFolder
- imitating BasicUsers

```
def getUser(self, name):
    """
    Returns the user object specified by name.  If there is no
    user named 'name' in the user folder, return None.
    """

def getUsers(self):
    """
    Returns a sequence of all user objects which reside in the user
    folder.
    """

def getUserNames(self):
    """
    Returns a sequence of names of the users which reside in the user
    folder.
    """
```

The User "API"

- User folder "users" needs to support these methods:

```
def getUsername(self):  
    """Return the username of a user"""  
  
def getId(self):  
    """Get the ID of the user. The ID can be used, at least from  
    Python, to get the user from the user's  
    UserDatabase"""  
  
def _getPassword(self):  
    """Return the password of the user."""  
  
def getRoles(self):  
    """Return the list of roles assigned to a user."""  
    raise NotImplementedError  
  
def getRolesInContext(self, object):  
    """Return the list of roles assigned to the user,  
    including local roles assigned in context of  
    the passed in object."""  
  
def getDomains(self):  
    """Return the list of domain restrictions for a user"""
```

The UserFolder "API"

- replace methods by subclassing UserFolder
- changing UserFolder behaviour

```
def identify(self, auth):
    """returns a name and password from a HTTP authentication header"""

def authenticate(self, name, password, request):
    "Does name and password authenticate as a user in this userfolder?"

def authorize(self, user, accessed, container, name, value, roles):
    """
    Is user allowed to access name and value from container,
    accessed through accessed? ...Defer to security policy
    """

def validate(self, request, auth):
    """
    this method performs identification, authentication, and
    authorization but calling the identify, authenticate and
    authorize methods of the userfolder.
    """
```

The UserFolder "API"

- management from scripts is frustrating
 - unscriptable from protected code

```
def _doAddUser(self, name, password, roles, domains, **kw):  
    """Create a new user. The 'password' will be the  
    original input password, unencrypted. The implementation of this  
    method is responsible for performing any needed encryption."""  
  
def _doChangeUser(self, name, password, roles, domains, **kw):  
    """Modify an existing user."""  
  
def _doDelUsers(self, names):  
    """Delete one or more users."""
```

- ...or hell on earth:

```
def manage_users(self, submit=None, REQUEST=None, RESPONSE=None):  
    """This method handles operations on users for the web based forms  
    of the ZMI. Application code is encouraged to use manage_std_addUser"""  
    if submit=='Add...':  
        return self._add_User(self, REQUEST)
```

What happened next?

Abstractions

- LoginManager
 - Wacky ZPatterns underpinnings
 - “Plugins” WAY ahead of its time...

- eXtensible User Folder
 - Lots of code / options
 - Questionable Quality

Evolution

- Password Encryption
 - clunky due to backward compatibility
 - AccessControl.AuthEncoding
 - Zope 2.7 supports SHA, SSHA, Crypt, MySQL

```
class PasswordEncryptionScheme: # An Interface

    def encrypt(pw):
        """Encrypt the provided plain text password."""

    def validate(reference, attempt):
        """Validate the provided password string. Reference is the
        correct password, which may be encrypted; attempt is clear text
        password attempt."""

    def registerScheme(prefix, schemeObject):
        "Registers an password encoding scheme for '{prefix}xxx'"
```

Evolution

- Scriptable Aliases
 - at last!

```
def userFolderAddUser(self, name, password, roles, domains, **kw):
    """API method for creating a new user object. Note that not all
    user folder implementations support dynamic creation of user
    objects."""

def userFolderEditUser(self, name, password, roles, domains, **kw):
    """API method for changing user object attributes. Note that not
    all user folder implementations support changing of user object
    attributes."""

def userFolderDelUsers(self, names):
    """API method for deleting one or more user objects. Note that not
    all user folder implementations support deletion of user objects."""
```

Add-on Products

- CookieCrumbler
 - addition of form based challenges
 - addition of cookie based authentication
- Groups User Folder
 - addition of groups

Where does all this leave us?

- With a helluva lot of complexity!

Making Things Simpler!

- SimpleUserFolder
- Simple API
- Documented API
- Scriptable with Python Scripts
- Scriptable with ZSQL Methods
- Easy to Subclass

Add a User

- `def addUser(name, password, roles)`
 - This adds a user.
- `name`
 - a string specifying the user's name
- `password`
 - a string containing the user's password.
- `roles`
 - a list of strings identifying the roles to be stored for this user.

Edit a User

- `def editUser(name, password, roles)`
 - This edits a user. It should raise an exception if the user does not exist.
- `name`
 - a string specifying the user's name. This identifies the user so this method cannot be used to rename users.
- `password`
 - a string containing the user's password or `None`.
- `roles`
 - a list of strings identifying the roles to be stored for this user.

Delete a User

- `def deleteUser(name)`
 - This deletes a user.
- `name`
 - The name of the user to delete

Listing Users

- `def getUserIds()`
 - returns a list of user names for this user folder
- You only have to implement the methods you want to!

Getting Users

- def getUserDetails(name)
 - returns something like:

```
{  
  'password': 'p4ssw0rd',  
  'roles': ['Manager'],  
  'someProperty': 1  
}
```

- Extra properties can be accessed as follows

```
from AccessControl import getSecurityManager  
print getSecurityManager().getUser()['someProperty']
```

- Return None if user is not found

Easy to Script

- Just call the methods you've implemented!

```
# add a user
context.acl_users.addUser('test_user','own3d',['Manager'])

# edit a user
context.acl_users.editUser('test_user',None,[])

# delete a user
context.acl_users.delUser('test_user')
```

Some Examples

- Python Script Implementation
- ZSQL Method Implementation

What's not to like about SUF?

- Can't override auth method
 - use cookie crumbler
 - subclass and override “authenticate”
- No computed local roles
- No groups
- Can't do complex things
 - getUserDetails called often
 - back to subclassing

A New Beginning?

- Pluggable Auth Service
 - requires PluginRegistry
- I haven't had long to play with this!
- Maintained by Zope Corporation
- Documented Plugin API
- Everything is pluggable

Plugin Architecture

- Each plugin can provide several types of functionality
- Each type of functionality is defined by an Interface
- Each type of functionality is “activated” once added

Types of Functionality

- Extraction
 - get credentials from request
- Authentication
 - validation credentials
- Challenge
 - challenge user to provide credentials
- Update (Credentials)
 - when the user credentials are edited

Types of Functionality

- Reset Credentials
 - do logging out
- UserFactory
 - create users
- Properties
 - add properties
- Groups
 - what groups does a user belong to?

Types of Functionality

- Roles
 - what global roles does a user have?
- Validation
 - check properties
- User Enumeration
 - search for users
- User Adder
 - creating users

Types of Functionality

- Group Enumeration
 - querying groups by id
- Role Enumeration
 - querying roles by id
- Role Assigner
 - assign roles to Principals

What's in the box?

- Scriptable Plugin
- ZODB user management
- Session Authentication
- Basic Authentication
- Cookie Authentication
- Delegation to Existing User Folders

What's not to like about PAS?

- Scriptable?
 - has search methods
 - back to the old way of add/edit users
- Complexity
 - remember LoginManager & eXUserFolder?
- Who writes plugins?
- Feels Alpha Quality
 - and not packaged
 - will it go into the core?

The Return of the Jedi?

- SimpleUserFolder 2.0?
- Single method to implement?

```
def validate(context, user, request):  
  
    # challenge  
    if not request.form.has_key('password'):  
        raise request.RESPONSE.redirect('login_page')  
  
    # authenticate  
    if request.password != 'own3d':  
        raise Unauthorized  
  
    # authorize  
    if '/'.join(context.getPhysicalPath()) == '/Members/test':  
        user.addRole('Manager')
```

- What about user management?

The Return of the Jedi?

- There will be gotchas
- Should I implement it?
- Should I “Package” PAS?

Evil still lurks...

- Can't eradicate old cruft without major suffering
- How does Zope 3 do all this?

The End

- Any questions?

Thankyou!

- Chris Withers
- chris@simplistix.co.uk
- <http://www.simplistix.co.uk>
- Do people want these slides to be available?

Zope.org

- Who wants to help?
- How do people who want to help go about helping?
- What are the problems?