PyObjC Hacking

Author

Bob Ippolito

Conference

PyCon DC, March 2005

Intended Audience

- Python developers using Mac OS X 10.3 or later
- ... that aren't (very) afraid of C
- Who probably know a little about Objective-C
- ... and want to do some crazy stuff on their Mac

Topics

- Objective-C Runtime Tricks
- Wrapping Frameworks
- Writing Plug-Ins
- Code Injection

Objective-C Runtime Tricks

- Classes
- ... at runtime
- Categories
- ... think mix-in
- Protocols
- ... think interface
- Selectors
- ... (not) everything is an object

Classes

- Are first-class objects
- Have a flat namespace
- The runtime is dynamic

Flat class namespace

```
>>> import objc
>>> objc.getClassList()
(<objective-c class NSRecursiveLock at 0xa0a055f8>,
 <objective-c class NSintNumber at 0xa0a06528>,
 <objective-c class NSRandomSpecifier at 0xa0a06d38>,
 ...)
>>> objc.lookUpClass('NSArray')
<objective-c class NSArray at 0xa0a037f8>
```

Dynamic runtime support

```
>>> import objc
>>> objc.lookUpClass('NSApplication')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
objc.nosuchclass_error: NSApplication
>>> import AppKit
>>> objc.lookUpClass('NSApplication')
<objective-c class NSApplication at 0xa2df8358>
```

Categories

- Used to add specific functionality to a class
- ... after it was created
- For example, AppKit adds drawing code to Foundation classes
- ... can be used to replace functionality

NSDate_gmtime.py

```
from Foundation import *
import objc
class NSDate(objc.Category(NSDate)):
    def gmtime(self):
        return time.gmtime(self.timeIntervalSince1970())
```

Loading the Category

```
>>> from Foundation import *
>>> now = NSDate.date()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  AttributeError: 'NSCFDate' object has no attribute 'gmtime'
>>> import NSDate_gmtime
>>> NSDate.date().gmtime()
(2005, 3, 22, 23, 41, 33, 1, 81, 0)
>>> now.gmtime()
(2005, 3, 22, 23, 40, 39, 1, 81, 0)
```

NSString_mangledIntValue.py

```
from Foundation import *
import objc
class NSString(objc.Category(NSString)):
    def intValue(self):
        # "self" is a real NSString here
        # not pretending to be unicode
        try:
            return int(self.UTF8String(), 0)
        except ValueError:
            return 0
```

Don't Try This At Home!

```
>>> import objc
>>> s = NSString.stringWithString_(u'0666')
>>> s.intValue()
666
>>> import NSString_mangledIntValue
>>> s.intValue()
438
```

Protocols

- A way to declare formal interfaces without inheritance
- ... that can be checked at runtime
- Looks like an @interface block
- Not often useful, but some applications use it to verify plugins

Getting a Protocol

```
>>> import objc
>>> objc.protocolNamed('NSObject')
<objc.formal_protocol NSObject at 0x5f160>
```

Checking Protocol conformance

```
>>> import objc
>>> NSCoding = objc.protocolNamed('NSCoding')
>>> o = NSObject.alloc().init()
>>> o.conformsToProtocol_(NSCoding)
0
```

Declaring Protocol conformance

```
import objc
NSLocking = objc.protocolNamed('NSLocking')
class DoesntReallyConformTo(NSObject, NSLocking):
    # if it conformed, there would be
    # an implementation here
    pass
```

Creating new Protocols

```
import objc
MyProtocol = objc.formal_protocol(
    "MyProtocol",
    None,
    [
        objc.selector(
            None,
            selector='mymethod',
            signature='v@:',
        ),
    ],
)
```

Selectors

- Is the "name" of a message that can be sent
- Each colon in the name denotes an argument
- Objective-C message syntax mixes the selector and its arguments
- ... PyObjC does not (can't)
- ... and it uses underscores instead of colons
- Normally the defaults are good for PyObjC
- ... unless the selector is used dynamically by Objective-C code
- Type signature is preserved by the Objective-C compiler (yay!)

Inspecting a Selector

```
>>> from Foundation import *
>>> sel = NSData.dataWithBytes_length_
>>> sel.selector
'dataWithBytes:length:'
>>> sel.signature
'@16@0:4r^v8I12'
```

Implementing non-default Selector

```
from Foundation import *
import objc
import random

class NeedsToReturnInts(NSObject):
    def anInt(self):
        return random.randint(-1000, 1000)
    anInt = objc.selector(anInt, signature='i@:')
```

Type@:{Signatures=i@c}?!

- Look like line noise
- We don't offer a way to explain them
- Or an easy way to compose them
- But our docs point to the relevant Apple docs

Wrapping Frameworks

- There are a bunch of cool third party frameworks you can use
- You can grab useful stuff from C frameworks we don't wrap
- We can't commit Tiger code yet, so you have to wrap those by hand
- Fortunately it's easy enough

DiscRecording.py

```
import objc as _objc
# this can be an absolute path too
_path = _objc.pathForFramework('DiscRecording.framework')
_objc.loadBundle(
    'DiscRecording',
    globals(),
    bundle_path=_path,
)
```

Poking at DiscRecording

```
>>> from DiscRecording import *
>>> print u'\n'.join([
...     device.displayName()
...     for device in DRDevice.devices()
... ])
MATSHITA DVD-R UJ-815
```

Plugins

- Built like a framework, but is runtime loadable code (MH_BUNDLE)
- Python isn't great at this, damned global state!
- ... but it's good enough (that's what I tell myself, anyway)

Where are they used?

- Services (bad idea, every process gets them)
- ... but there is a process-based API too
- Input Managers (bad idea, every process gets them)
- Screen Savers
- Interface Builder palettes
- To extend existing Cocoa applications (QuickSilver, etc.)
- To bootstrap the evil that is objc.inject

Plugin Guidelines

- Usually have to set a custom NSPrincipalClass in the Info.plist
- One and only one Python per process
- ... shared sys.modules, etc.
- Global state = Ugh.

setup.py for SillyBallsSaver

```
from distutils.core import setup
import py2app

plist = dict(
    NSPrincipalClass='SillyBalls',
)

setup(
    plugin=['SillyBalls.py'],
    data_files=['English.lproj'],
    options=dict(py2app=dict(
        extension='.saver',
        plist=plist,
    )),
)
```

objc.inject

- Think "gdb attach"
- Lots of possibilities
- Loads a Python plugin into any app
- A great way to crash
- Module-level code is NOT EXECUTED IN THE MAIN THREAD

objc.inject syntax

```
import objc
objc.inject(<pid>, full_path_to_bundle)
```

Questions?

Go ahead, ask.