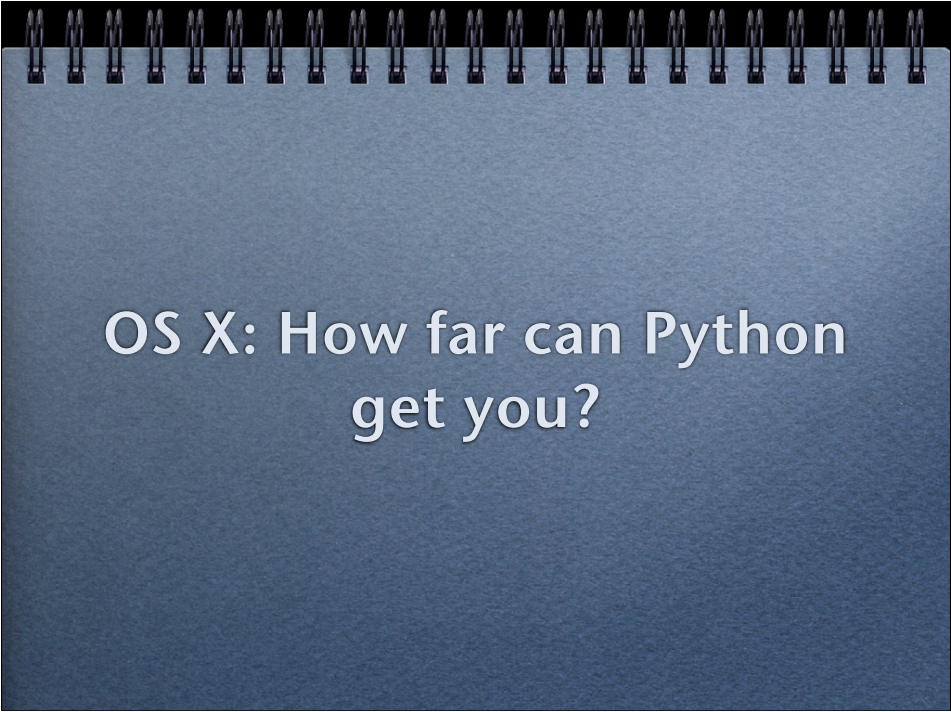


The company I work for, Justsystems, is in parentheses because I'm not representing the company here. The applications and code I'm presenting is what I've written during my "hobby time," which is generally between 9 pm (when my kids go to sleep) and 11 pm (when I try to get to sleep, but end up watching The Daily Show instead).



This photo was taken at the Northern Voices blogging conference when my son was five. He wasn't actually blogging at the time, though. He was playing Bejeweled.



OS X: How far can Python  
get you?

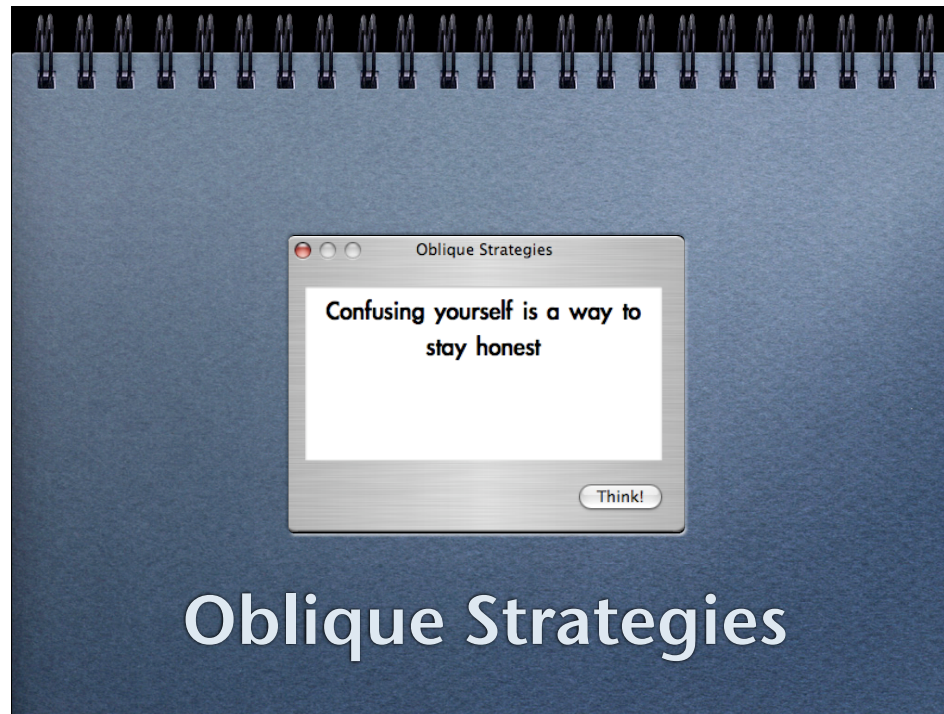
## Pretty far...

- Standalone applications
- Screensavers
- Utilities
- Games
- Plugins

# Standalone Applications

*A whirlwind tour*

- Oblique Strategies*
- Sandcastle*
- Conversations Network Uploader*
- DrawingBoard*
- Kutia*

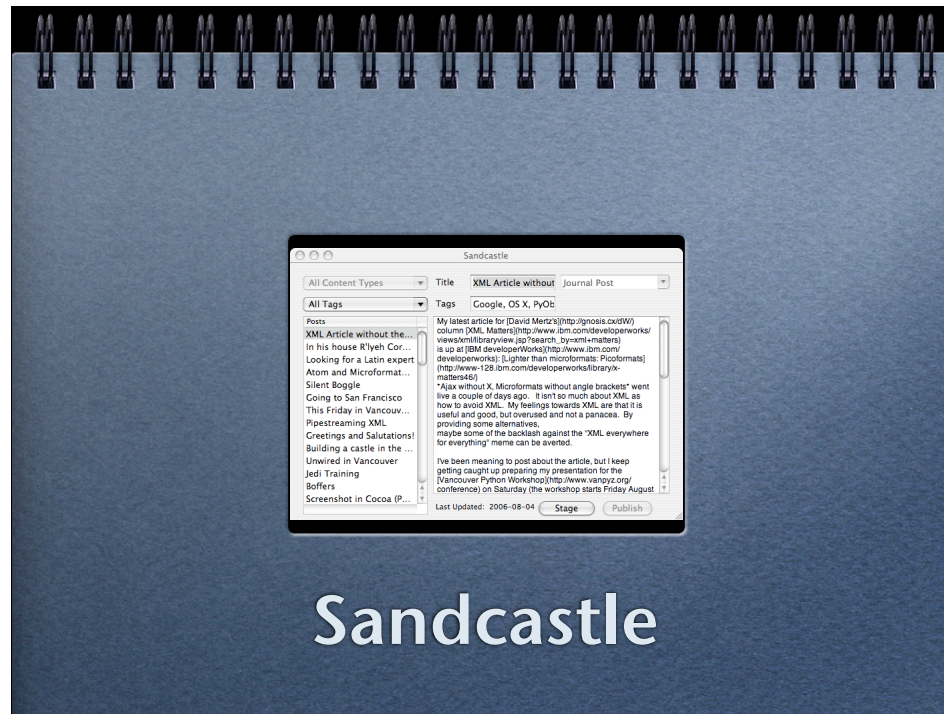


Oblique Strategies are a brainstorming/block-breaking tool from Brian Eno and Peter Schmidt, originally presented as a deck of cards. My first encounter with them was a list of some of the strategies in the Whole Earth Review. There are free versions of Oblique Strategies (web pages, OS X app, and Dashboard widget). What am presenting is based visually on the (free, but not open-source) OS X app from <http://www.curvedspace.org/software/oblique.html>. There is a good online summary here: <http://www.rtqe.net/ObliqueStrategies/>, and of course the ever-lovin' Wikipedia has a page on Oblique Strategies: [http://en.wikipedia.org/wiki/Oblique\\_Strategies](http://en.wikipedia.org/wiki/Oblique_Strategies).

## Why clone a free app?

- As an example for learning Cocoa
- So I could add quotations from other sources
- Became my "hello world" program for learning new things, like Renaissance

I learned a lot about OS X programming by writing this program over and over. I have written it in the "proper" way, using Interface Builder for the GUI, the "don't try this at home way" creating the GUI completely from code (no NIB file needed), and the completely wacky Renaissance way (GUI defined in XML file). More about Renaissance here: <http://www.gnustep.it/Renaissance/>.



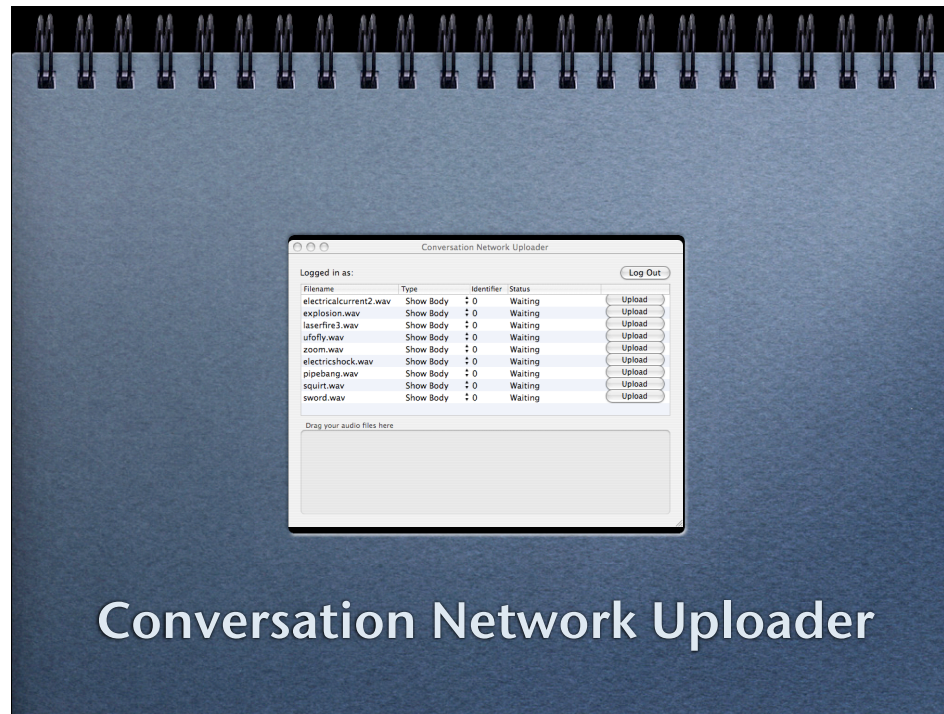
Sandcastle is my blogging client software. My blog doesn't have a server, using Apache, static files and a small amount of SSI includes. Sandcastle has both a GUI and a command-line version. The GUI was built because a) I wanted to add tags to a lot of posts imported from Blogger, and b) it was pretty easy to throw together a GUI. I've since found that I use the command-line more, but that's mainly just a personal preference thing. The good part is both versions work off the same Python code base.



### Why write yet another blogging tool?

- Existing tools trying to do too much
- Easier to write a new tool than remove functionality that gets in my way
- Easier to write a new tool than add missing features
- Support Atom and tagging as core features, not hacked in later

Server-side I've tried Manila, Blogger, and WordPress, and none of them felt right. I want a pretty simple, almost stark, site, but with these systems there was too much to turn off, too many options and settings to tweak. With Sandcastle all style and layout is done with CSS, and most of the site navigation is done with tags. Atom 1.0 syndication is used throughout, with each tag getting its own feed. Both tagging and Atom support are additions in other systems, but they are core to Sandcastle.



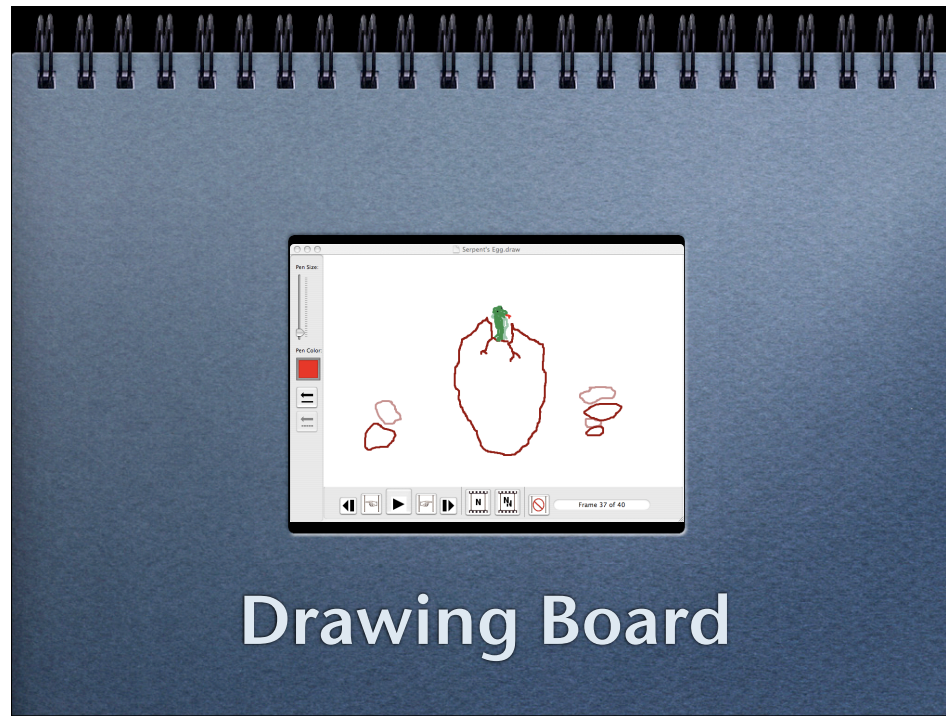
## Conversation Network Uploader

The Conversation Network is a spin-off from IT Conversations, to allow more people to create and upload podcasts on a variety of subjects. The Windows client had already been written, using Python, wx, and some command-line utilities. My job was to port the wxPython code to the Mac. Instead I rewrote it using PyObjC and made it more “Mac-like.” The result was well-received.

## Why write an uploader?

- My boss asked me to
- To give back to the community
- To be involved with a high-visibility project
- To show "The Mac way" to better GUI
- Because I didn't think it would take long

I was wrong about the last part, but I was pleasantly surprised how little code it took to get drag-and-drop working. Using drag-and-drop with lists is a bit more work, but quite doable.

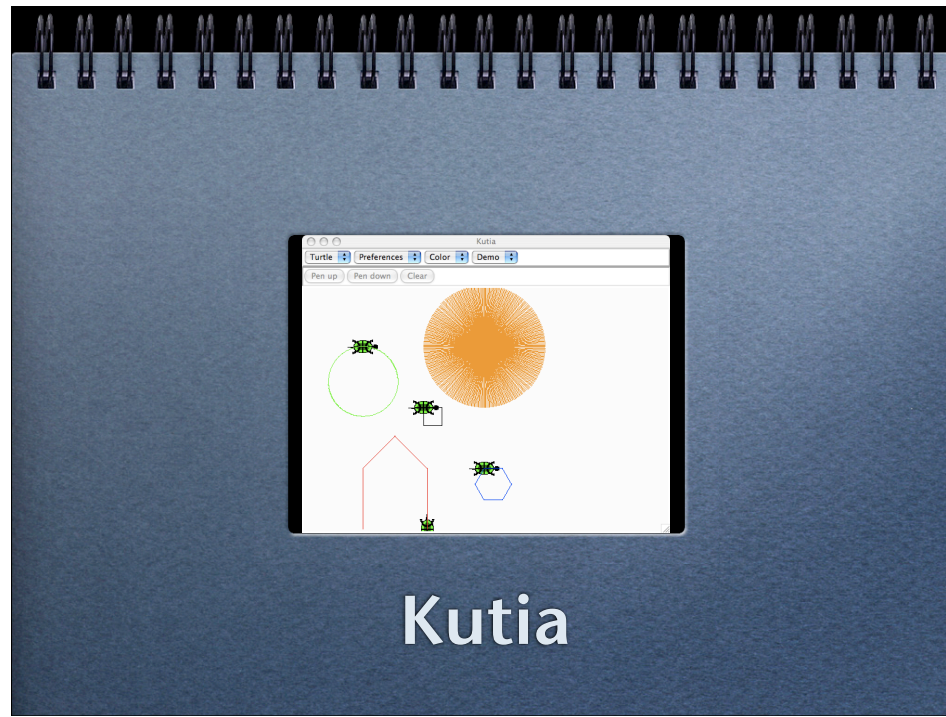


Drawing Board is my lightweight animation tool for kids. I've intentionally kept the feature set small, to focus on simple, sketch-based animation with few distracting bells and whistles. It also can export the resulting animation to SVG for viewing on the web.

## Does the world need another drawing app?

- Corel Dabblar was discontinued
- KidsPix too focused on eye candy
- I want kids to be able to "kick ass"
- Do one thing well and get out of the way
- Simpler is better

Dabblar was a great product, easy to use, with onion-skinning for animation, and natural media tools. If it had been ported to OS X I would just use it (since I'm never likely to create natural media tools myself). KidPix has lots of special effects and my kids mainly draw to blow things up (clear the screen via explosion). I want the kids to be able to a) create art, and b) share their art.



Kutia is my turtle drawing program. This version is using Tkinter, but I'm porting it to PyObjC. The name Kutia came when my wife and I were first dating. She is Bulgarian and I was trying to learn a few words in Bulgarian by asking what everyday objects were. We found a box turtle together and I asked her what the Bulgarian word for turtle was. "Kostenurka," she said (with a rolled "r"). So I asked what the Bulgarian word for box was. "Kutia," she said. So I called the turtle Kutia.

## A different kind of turtle

- Direct manipulation*
- Auto-scripting*
- Learning by doing*

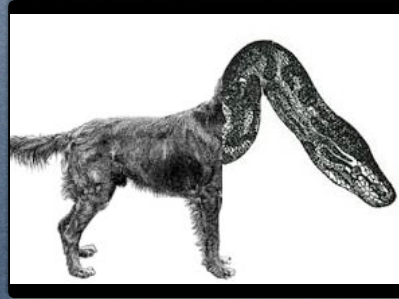
There are two things Kutia does differently (or will, when it's finished). Kids can move the turtle by clicking on the canvas, but clicking buttons, or by dragging the turtle. And the turtle records its movements in a script in the script window. Kids can then edit and re-run the script, playing with different values. Turtle graphics are often used to teach kids programming, but Kutia eases them into it by doing the programming for them at first. This way, it can even be used by pre-literate kids.



## A short segue

Storytime. When I was working on my Master's degree I was also working full time for the university as a networking engineer. We had a lot of Macs and I was the guy people would come to when Apple couldn't help them. At the same time, I was doing a lot of unix scripting, and I've always loved the unix philosophy of small tools that do one thing well and can be wired together to build more complex tools. After using both Mac and unix for so many years, having them both in one package with the introduction of OS X was like having the two halves of my soul joined together. OK, maybe a little exaggerated, but I like OS X a lot, because I like using a Mac, and I like programming on unix. It's a best of both worlds thing.





## PyObjC: A Love Story...

As much as I like OS X, there's another piece to the story. While Objective-C feels to me like what C++ should have been, it's no Python (though it is "pythonic" in many ways). Cocoa is the rich set of libraries for OS X, originating with NextStep, that gives a programmer on OS X so much leverage. The magic is in the PyObjC library which is the glue that brings Cocoa to Python and Python to Cocoa.

## Advantages of OS X

- Beautiful UI
- Stable
- Fast
- Rich libraries
- Many services to draw on
- Best of Unix and Mac

Beauty and elegance are themes we'll see again.

## Advantages of Cocoa

- Beautiful widgets
- Drag and drop UI design (or via code)
- Data binding
- Easy to connect to C
- Introspection
- Bundles and Frameworks

Objective-C (the native language of Cocoa) is a thin layer (inspired by the language Smalltalk) over C. Because it is a pure superset of C, it makes it very easy to call into C libraries, and to wrap them to be exposed to Python over the PyObjC bridge. Objective-C can be introspected into at runtime just like Python, which makes it an environment you can learn by exploring.

## Advantages of Python

- Gives Cocoa garbage collection
- Maps consistently to Objective-C
- Great libraries: ElementTree, Twisted
- String manipulation a breeze
- Interactive prompt
- Less verbose than Objective-C

What can I say? I have used, and continue to use, many programming languages, but I keep gravitating toward Python. It's not perfect, but it has the best balance of anything I've found. The fact that you can make applications and plugins on OS X with Python, but no-one has to know you're using Python, is like having a secret superpower.

## Advantages of PyObjC

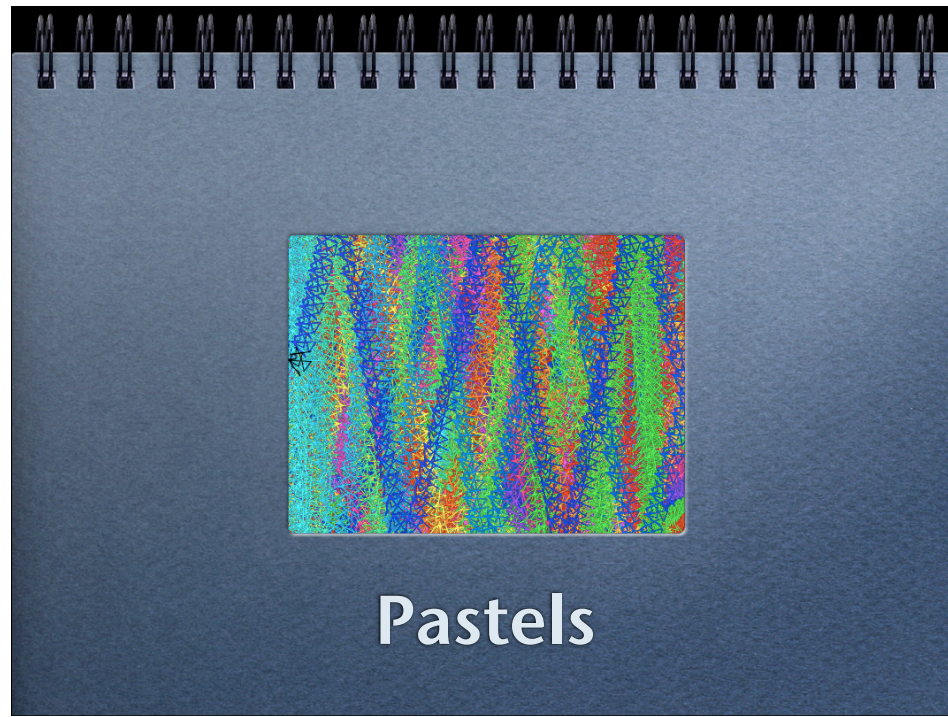
- Subclass Cocoa in Python
- Subclass Python in Cocoa
- Python apps calling OS X libraries
- Objective-C apps embedding Python console or plugins

This is where it all comes together. Beauty and elegance, simplicity and power. Executable pseudocode meets lickable, drag-n-drop UIs.

## Plugins and Utilities

- Screensaver
- Preference Pane (PyObjC example)
- StatusItem

Enough talk, time for more examples.



The idea behind pastels came from simply imagining a random squiggle drawn over and over while cycling through colours and moving the squiggle around. It turned out even better than I'd hoped, becoming my most successful project as far as my kids were concerned.

## Screensavers

- Screensavers are a type of bundle, lessons learned are widely applicable in OS X
- Implementing a screensaver using PyObjC is easy
- Making it configurable was a pain
- The PyObjC mailing list is a lifesaver

Project page: <http://livingcode.org/project/pastels/>

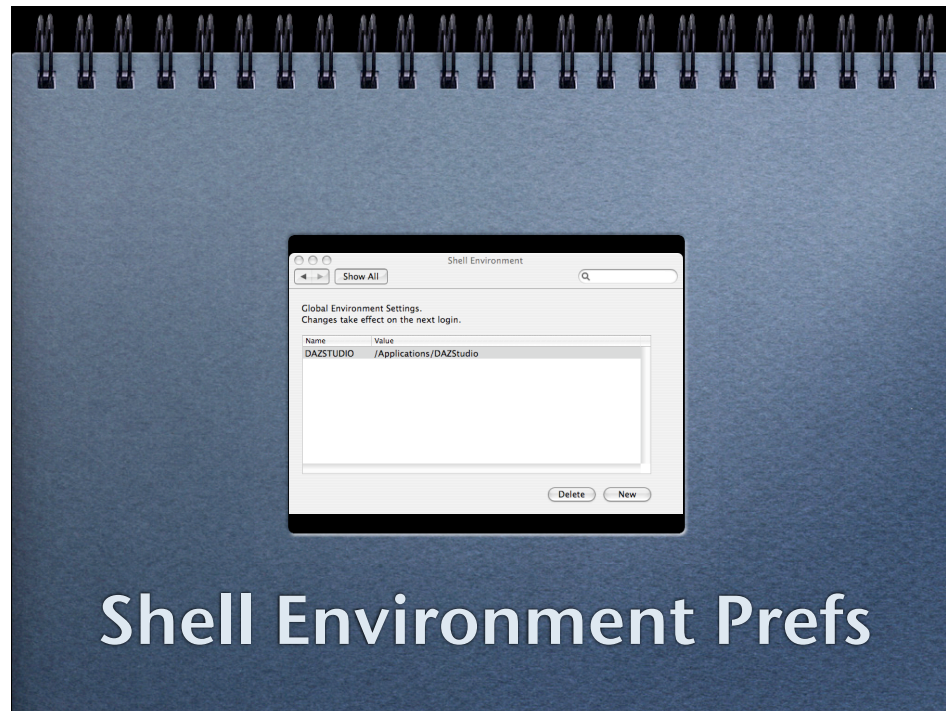
Source code: <http://code.google.com/p/pastels/>



The other half of  
my QA  
Department



My daughter liked Pastels so much that I added a secret feature just for her: If you press the “p” key while it is running, instead of exiting the screensaver, it will save a snapshot of it to the desktop. My kids are tough critics, so the fact that they enjoyed watching this program run was very rewarding for me.



## Shell Environment Prefs

This is the only program shown here that I didn't write (the Conversation Network Uploader contains a lot of code I didn't write, but I built the UI) an example program from the PyObjC distribution to show how easy it is to build plugin bundles for other programs. The system preferences are just a bunch of these plugins and you can easily create your own.

## Plugins are easy

- Inherit most behaviour from Cocoa
- Define interface in Interface Builder
- Usually a couple of required methods
- Several optional methods if needed
- My examples tend to be longer to avoid the Interface Builder step...

Since a lot of what I write is intended to be example code, it ends up being longer than needed because I often create the UI in code rather than in Interface Builder. This makes more work for me, but is easier to write about than “ctrl-drag from the icon of the window in the object palette to the instance of the controller, then in the inspection window choose the window outlet and click OK,” which is how articles about Interface Builder projects tend to read (accompanied by copious screenshots). Even with the verbose descriptions and the illustrations, I have trouble following that, but not much problem with “self.window = Window(size=(640,480), title=”My Application”).

## NSStatusItem

```
def applicationDidFinishLaunching_(self, notification):
    print 'timer launched'
    # Make the statusbar item
    statusbar = NSStatusBar.systemStatusBar()
    # if you use an icon, the length can be NSSquareStatusItemLength
    statusitem = statusbar.statusItemWithLength_(NSVariableStatusItemLength)
    self.statusitem = statusitem # Need to retain this for later
    # statusitem.setImage_(some_image)
    #statusitem.setMenu_(some_menu)
    statusitem.setToolTip_('Seconds since startup')
    statusitem.setAction_('terminate:') # must have some way to exit
    self.timer = NSTimer.alloc().initWithFireDate_interval_target_selector_userInfo_repeats_(
        start_time,
        1.0,
        self,
        'display:',
        None,
        True
    )
    NSRunLoop.currentRunLoop().addTimer_forMode_(self.timer, NSDefaultRunLoopMode)
    self.timer.fire()

def display_(self, notification):
    print 'display:'
    self.statusitem.setTitle_(elapsed())
```

No screenshot for this one, it just puts the text of a number in your menu bar, and increments the number once per second. There's no NIB file and this is the bulk of the code (there's a little bit of UI initialization code too). If you add a NIB file it's easy to put an image and a menu in there too. Status items have some limitations (you can't control their position in the menu bar, for instance), but it can be really handy to stick a small utility where it can be accessed at any time, no matter what application is running.



We've seen how PyObjC can help you build:

- Standalone applications
- Plugins
- Screensavers
- Wrappers for command-line utilities
- Any questions so far?

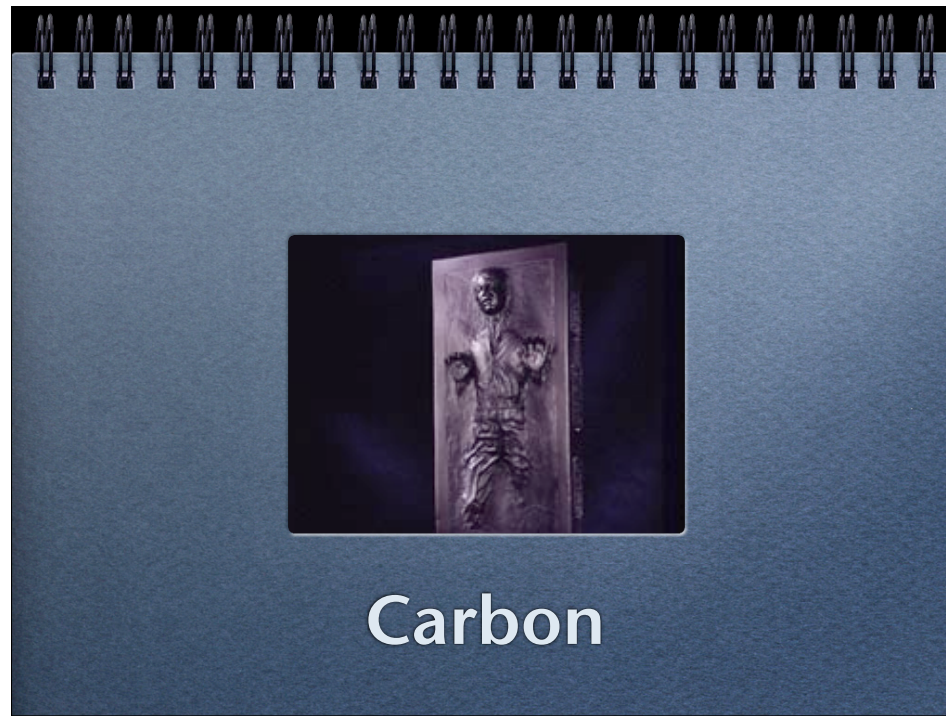
Strictly speaking I haven't shown how to wrap command-line utilities. The Conversation Network Uploader is basically such a wrapper, with some threading to handle several processing pipelines going on at the same time. But really, Python makes this easy: just use the "subprocess" module.

Holy Flying  
Spaghetti  
Monster!

Is there anything  
PyObjC can't do?



Arr, I just be gettin' to that, me hearties.



## Carbon

Carbon is the name for the C-based libraries for OS X. This includes a lot of legacy stuff from OS 9 days, but also a lot of newer functionality. The newer stuff is all available from Cocoa (and thus Python), but there's still a lot that you have to drop down to C to do. The good news is, Python allows you to drop down to C when you need to. In fact, it gives you more choices than Starbucks...

## Many ways to wrap C

- SWIG
- Pyrex
- Psyco
- Boost (C++)
- Ctypes
- Objective-C framework

My opinions: Friends don't let friends use Swig, Pyrex is easiest, Ctypes has the benefit of being part of the standard library (as of Python 2.5). If you don't mind adding a framework to your project, wrapping the C in Objective-C and building it as a framework is pretty easy, and as I'll show in a few slide, wrapping an Objective-C framework to be imported by PyObjC is basically two lines of code. The PyObjC tools also include a Ctypes-like API, but Ctypes is: a) more standard, and b) better documented.



## Itches that are hard to scratch

- Creating an editable Quicktime movie
- Grabbing the framebuffer (iSight)
- Recording from the microphone
- Checking CPU temperature
- Taking a screenshot
- Etc.

I've almost got the Quicktime one working, after struggling with it off-and-on for months. I'll show the iSight and screen shot solutions further on. Checking the CPU (or CPUs) is mainly a matter of sifting through a bunch of data to extract what you need (tedious, but possible). I haven't yet tried to record from the microphone, that's coming once I get the Quicktime puzzler solved.

## Taking a screenshot

```
def screenShot(self):
    rect = NSScreen.mainScreen().frame()
    image = NSImage.alloc().initWithSize_((rect.size.width,
    rect.size.height))
    window = NSWindow.alloc
    ().initWithContentRect_styleMask_backing_defer_(
        rect,
        NSBorderlessWindowMask,
        NSBackingStoreNonretained,
        False)
    view = NSView.alloc().initWithFrame_(rect)
    window.setLevel_(NSScreenSaverWindowLevel + 100)
    window.setHasShadow_(False)
    window.setAlphaValue_(0.0)
    window.setContentView_(view)
    window.orderFront_(self)
    view.lockFocus()
    screenRep= NSBitmapImageRep.alloc().initWithFocusedViewRect_
    (rect)
    image.addRepresentation_(screenRep)
    view.unlockFocus()
    window.orderOut_(self)
    window.close()
    return image
```

This is one way to take a screenshot, completely within Python/PyObjC/Cocoa. There ought to be a way to do this with one API call. Heck, Python on a Nokia phone can do it.

Or...

```
def screenShot2(self):  
    "Puts screenshot on the pasteboard"  
    subprocess(['screencapture', '-c', '-C'])
```

Here's the alternative. Use the subprocess module to call a built-in command-line utility. While the previous example returns an NSImage, this version puts the screenshot on the system pasteboard, so if you need it as an image you have to extract it just as you would doing cut-and-paste.

## PySight: iSight for Python

- Tim Omernick's `CocoaSequenceGrabber` framework does the hard part
- I only wrap it using `PyObjC`
- The next slide shows the complete code to wrap the framework

There are a couple of open-source frameworks out there for working with the framebuffer. Again, this is harder than it needs to be (but supposed to get better with next year's OS 10.5). I chose Tim's code because it doesn't try to do too much, just what I needed.

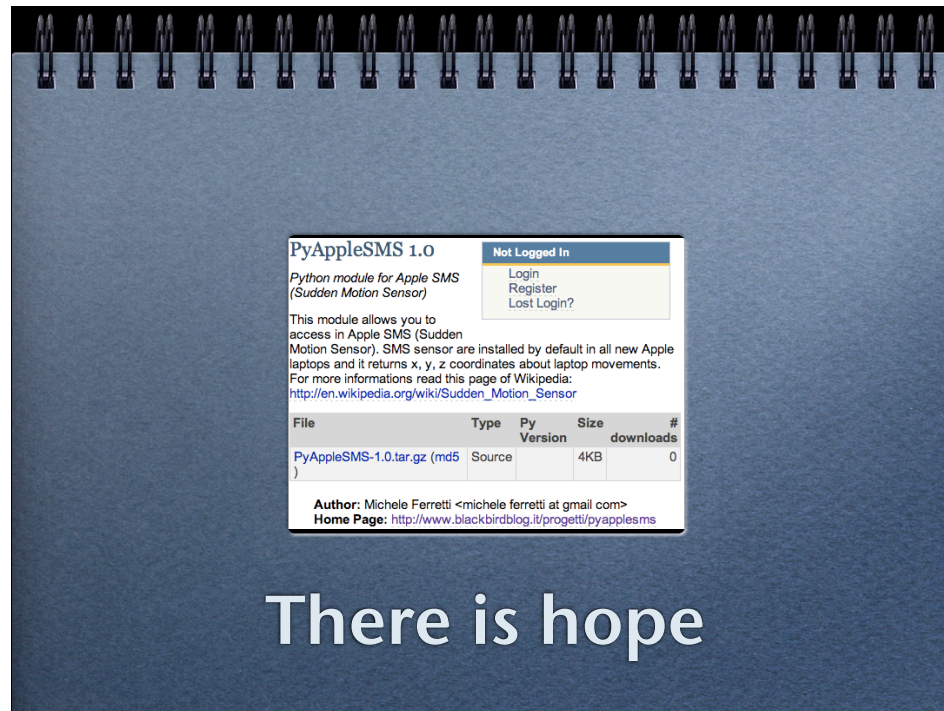
## pysight.\_\_init\_\_.py

```
import objc, AppKit, Foundation, os
if 'site-packages.zip' in __file__:
    base_path = os.path.join(os.path.dirname(os.getcwd()),
                              'Frameworks')
else:
    base_path = '/Library/Frameworks'
bundle_path = os.path.abspath(os.path.join(base_path,
                                              'CocoaSequenceGrabber.framework'))
objc.loadBundle('CocoaSequenceGrabber', globals(),
               bundle_path=bundle_path)
del objc, AppKit, Foundation, os, base_path, bundle_path
```

OK, a bit more than two lines of code. There's some path testing to see if we're running from the command-line or within a distutils-built program (or bundle), and some namespace cleaning at the end, but the bulk of the work is : import objc, call objc.loadBundle(). Two lines to import an arbitrary Cocoa-based framework.



When I rewrote Tim's example app using the Python wrapper it was about half the lines of code and ran at the same speed.



There is hope

This is a recent post to the Python Cheeseshop. Newer Macs have lots of things that are not accessible from Cocoa: Remote controls, sudden motion sensors (SMS), light sensors and control for the keyboard backlight. This is a project to access the sudden motion sensor in newer laptops from Python. I wish I could read the author's blog, but it is in Italian. Between Apple exposing more functionality to Cocoa with each OS release, and independent authors like Michele and Tim wrapping the C APIs to make them more accessible, there is hope for the desperate Python hacker.



## Programming for the Daniela of it

My wife is 100% non technical, but she reads my technical articles to ensure they make sense at least grammatically. My goal is to make writing Python programs so accessible and contagious that not only my kids take it up, but that Daniela can dive in and create her own tools. Computers can be so much more if users can actually take control of them.





AppKiDo is a great interface for reading the Cocoa documentation neatly hyperlinked and searchable (and it is open source if you want to learn from its code). Print statements in PyObjC applications go to the console, so keep it handy. The console is an application that lives in `/Applications/Utilities/`.

A dark blue, spiral-bound notebook cover is shown. The spiral binding is at the top. The text "Any questions?" is centered on the cover in a white, sans-serif font. Below it, the URL "http://livingcode.org/" is written in a smaller white font.

Any questions?

<http://livingcode.org/>