MAC OS X HACKS

100 Industrial-Strength Tips & Tools

O'REILLY®

Rael Dornfest & Kevin Hemenway

MAC OS X HACKS

MAC OS X HACKS

Rael Dornfest and Kevin Hemenway

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

HACK #45 Speakable Web Services

Explore Mac OS X's speech recognition and its suitability for building useful, voice-driven commands that invoke external as well as local web services.

When Scotty tried to talk to a Macintosh through its mouse in *Star Trek IV* (1986), the joke was on Apple. Why couldn't this famously easy-to-use computer accept the most natural form of input? Over the years, I dabbled now and then with voice command systems, but they never seemed worth the trouble—until now. I've been exploring the speech technologies in Mac OS X on an 800MHz TiBook, and I'm really impressed. Apple has done a marvelous job with the recognition and control systems, and now that you can script the Internet so easily in OS X, it's straightforward to build useful voice-driven commands that invoke external as well as local services. Consider this dialog:

> Me: "Temperature"
>
> Computer: "36 degrees"

There are, of course, a million ways to look up the temperature on the Web. Most of them start with the browser. You fire it up and go to a bookmark, which in my case is *http://www.weather.com/weather/local/03431*. There are at least two problems with this scenario. First, you have to translate the request into an application context (the browser) and a procedure (go to bookmarks, select Local Weather). Second, you destroy your original context. For example, I'm typing these words in the Emacs Terminal-based text editor. I'd like to keep on typing, and reading what I am writing, even as I ask for and receive the temperature. Speaking the request and hearing the response is an ideal solution. Here are a few ways to implement it.

Perl and AppleScript Working Together

I started with a Perl script that uses SOAP::Lite to hit a web service at XMethods (*http://www.xmethods.com*), like so:

```
#! /usr/bin/perl -w
use strict;
use SOAP::Lite;
my $temp = SOAP::Lite
  -> service('http://www.xmethods.net/sd/2001/TemperatureService.wsdl')
  -> getTemp('03431') . " degrees";
`osascript -e 'say "$temp"'`;
```

Here, we're using Perl's backtick evaluation to run a command-line tool, `osascript`, which runs AppleScript code—in this case, to speak the result of the SOAP call. Use of the text-to-speech engine introduces some fascinating

subtleties. For example, if you omit the leading space in degrees, the answer will sound like:

> three six period zero dee eee gee are eee eee ess

It would be handy if you could just save this as a file called *Temperature* in the *Speakable Items* folder (for example, */Users/john/Library/Speech/SpeakableItems*) and launch it by speaking the name "temperature." But so far as I've been able to determine, scripted speakable items (as opposed to those that invoke key-driven commands) have to be written in AppleScript and, further, saved from the script editor as type *application* (not *text* or *compiled script*). Fortunately, AppleScript can invoke the Unix shell, which can invoke the Perl script. Let's refactor slightly, and have the Perl script simply return a bare value, suitable for downstream use in any kind of application, whether voice-enabled or not:

```
#! /usr/bin/perl -w
use strict;
use SOAP::Lite;
print SOAP::Lite
-> service('http://www.xmethods.net/sd/2001/TemperatureService.wsdl')
-> getTemp('03431');
```

I saved that script as */Users/jon/Temperature* and then saved the following AppleScript application as */Users/jon/Library/Speech/SpeakableItems/Temperature*:

```
set theResult to do shell script "/Users/jon/Temperature"
say theResult & " degrees" as string
```

Now, the textual result of the `Temperature` script is spoken by AppleScript. You can, alternatively, do the whole thing in AppleScript, like so:

```
tell application "http://services.xmethods.net:80/soap/servlet/rpcrouter"
  set theResult to call soap {method name:"getTemp", \
    parameters:{zipcode:"03431"}, method namespace \
    uri:"urn:xmethods-Temperature", SOAPAction:"/TemperatureService"}
end tell

say theResult & " degrees" as string
```

This is easier in one way, harder in another. It's easier if you're not a Perl programmer or if you haven't added SOAP::Lite and its required substrate (`expat`, XML::Parser) to the Perl kit that comes with Mac OS X. But when a web service is described by a Web Services Description Language (WSDL) file, it's easier to use SOAP::Lite than AppleScript, since the former can use the WSDL file to simplify access.

It's ideal when there's a web service that will give you the answer you're looking for, but when that's not the case, there's always good old HTML screen-scraping. In that case, a language like Perl or Python will run rings

around AppleScript. Here's a script that speaks my weblog's current rank and page-view count for today:

```
! /usr/bin/perl -w
use strict;
use LWP::Simple;
my $res = get "http://www.weblogs.com/rankingsByPageReads.html";
$res =~ m#(.+)Jon's Radio</a></td><td><td align="right">(†+) #;
my $preface = $1;
my $count = $2;
$preface =~m#">(†+)˙ </td><td>#;
my $rank = $1;
'osascript -e 'say "Rank $rank, count $count"'';
```

In this case, it's more trouble than it's worth to return raw results from Perl and format them for speech output in AppleScript.

I have to confess I'm still tempted to dismiss this speech stuff as an amusing parlor trick. But it may finally be reaching a tipping point. Look, Dad's talking to the computer, my kids snickered. When I showed my son he could play GnuChess using voice commands, though, he was riveted. It's a case-by-case thing, but when an application has a limited control vocabulary ("pawn a2 to a4"), the Mac's speaker-independent speech recognition can give you hands-free control that's accurate and more effective than mouse control. Well, to be honest, mostly accurate. I'm having a little trouble getting GnuChess to distinguish between "d" and "e"—a problem that could be solved by also supporting "delta" and "echo."

Not many of the XMethods services are likely candidates for voice treatment. Complex inputs and outputs don't make much sense. You can build IVR-style (interactive voice response) menus, like so:

```
tell application "SpeechRecognitionServer"
  local choices
  set choices to {"Temperature" "BlogStats"}
  set thePrompt to "What do you need to know?"
  try
    set theResult to listen for choices with prompt thePrompt giving ↵
    up after 10
    say (do shell script "/Users/Jon/" & theResult)
  end try
end tell
```

Unless you really want to inflict voice trees on yourself, though, you'll probably soon tire of this approach, once the novelty wears off. Complex output is a nonstarter as well. It's faster to read than to hear more than a word or short phrase, the Mac's synthesized voices work best on short snippets, and there's no way for the computer to usefully speak structured output.

Namespace Management

The namespace mode of the files in the *Speakable Items* folder is active system wide. There are separate per-application namespaces. For example, the *Speakable Items/Internet Explorer* subfolder defines voice commands just for MSIE. You can, in fact, extend that namespace in a hands-free manner, using the "make this page speakable" voice command. If the current page is *http://news.google.com*, for example, then "make this page speakable" prompts with the page's HTML doctitle, Google News. When the prompt is active, the valid speech commands are "save" and "cancel." If you say "save," you will create a voice-activated bookmark triggered by the phrase "Google News." Pretty darned slick! It's IE-specific, though, and that's a shame because I prefer Mozilla on the Mac to the IE version (5.2) that came with the TiBook.

The per-application namespaces are segregated from one another, but as you extend the main namespace, you'll start to run into conflicts. New commands that sound too much like existing ones will cause misrecognition. The problem is easily solved, though. Just open the *Speakable Items* folder and rename files—either preexisting items or your new items—in order to step around these conflicts.

As you build up vocabularies, it's easy to forget that the recognition engine is speaker-independent, not language-dependent. For example, I've been enjoying Brent Simmons' Huevos **[Hack #85]**, a nifty little tool that can float in a small window and send a search term to any of a user-defined set of web sites. The voice command to launch it—"switch to Huevos"—works best when I anglicize the name as "Hoo-eee-vos." Apple's site says that a Spanish recognizer is available but, for now, I'm still trying to decide whether to mangle the pronunciation of "Huevos" or rename it for speech purposes.

Speech control of computers is mainly considered to be an assistive technology. In my case, there's certainly an element of that. After too many years of typing and mousing, my wrists are chronically sore, and I'm happy to avoid all keystrokes and mouse clicks that I can. Most of that wear and tear is from writing and programming, though, so until I can come to terms with dictation (as, I'm told, the prolific author David Pogue has done), voice control won't help much. But Apple's implementation has made me rethink the mixed-mode user interface. Consider, for example, the mechanism for picking one of the 50 U.S. states in a web form. Some sites ask you to type the two-letter abbreviation, but most offer a picklist. Scanning a list of 50 items is unproductive. I can use completion to skip to the N section of the list, but adding an H takes me to Hawaii, not New Hampshire. Here's a well-defined namespace that could probably be accessed using speech more quickly and

naturally than by any other method. I suspect the same holds true for many multiple-choice situations in data entry forms and elsewhere.

Consider another Brent Simmons application, the popular RSS newsreader NetNewsWire [Hack #87]. It's already more usefully speakable then most OS X apps I've tried. Along with menu navigation, you can speak the crucial commands "next unread," "mark all as unread," and "open in browser." These are more mnemonic than their keyboard equivalents (⌘-G, ⌘-Shift-K, and ⌘-B) and, especially in the case of ⌘-Shift-K, more accessible too. An interesting refinement would be to voice-enable random access to feeds, just as MSIE allows spoken random access to items on the Go and Favorites menus. I've got 128 subscriptions, for example. It would be cool to say "Sam Ruby" and jump straight to Sam's blog. Or to say "Jeremy" and jump to a completion list showing Allaire and Zawodny, and speak one of those surnames to finalize the selection.

As software services multiply, so do their control vocabularies. XML manages this proliferation using namespaces. Per-application or per-service speech-enabling can use the same strategy to reduce the hard problem of open-ended speech recognition to an easier one that can be solved in useful and practical ways.

—Jon Udell