«This document contains excerpts from a discussion between Monty (IBMole author) and Art Isbell. It contains information that might help you a bit when using the IBMole. Unquoted text is Art's, quoted text (lighter gray) is Monty's.Ðdon»

I had about given up on making changes to outlets and actions because everything I tried failed. I then discovered that the string arguments for many IB methods aren't NXStrings, but are instances of a NXString subclass defined only in the IB executable, IPCConcreteCollectedStrings (!). NXString appears to be an

abstract superclass providing no storage of its own (early shades of the new class cluster concept, I guess).

But instantiating IPCConcreteCollectedStrings in IBMole clients was problematic because no implementation is available. Several other private IB classes would be nice to use as well. So I decided to ask for the class object through the IBMole by adding the `-getClass:` method.

Voila!! It works! So now I can instantiate any IB class in my IBMole client. For example, to change the name of the File's Owner's outlet "foo" to "bar":

```
id foo = [[[server
                getClass:"IPCConcreteCollectedString"]
                alloc]
                initFromCString:"foo"];
id bar = [[[server
                getClass:"IPCConcreteCollectedString"]
                alloc]
                initFromCString:"bar"];
id className = [[activeDocument rootObject] className];

[activeDocument renameOutlet:foo of:className to:bar];
```

Connections can be changed or added using IPCConcreteCollectedString arguments as well. So this has been

a really powerful addition to your IBMole capability.

Another problem I had difficulty solving was programmatically setting window options (Hide On Deactivate, Deferred, etc.). Apparently, IB methods access the WindowTemplate directly with no methods available to set the _windowFlags struct members. So I defined WindowTemplate category methods in IBMole.m that can be used to set the various window options.

My IBMole.m (try this stuff in C++ :-):

«Art's changes have already been folded into the IBMole you have on your palette.Đdon»

My guess is that you are asking for the current document too soon after asking the workspace to open it. I would put in a delay or something. Ever notice how IB seems to wait a few seconds to change the filename on a nib you just renamed? It seems to do some stuff with delayed events, and that might be tricking things up because your request for the current document is the first event in line, heck you might even be beating the workspace there!

This is indeed a problem. I have inserted at loop that tests for the existence of an active document in IB and doesn't move on until there is.

The best thing might be to modify the IBMole to do something with the event loop to synchronize things. I don't know exactly what, but it could register a callback to you and put a really low priority event in the queue and when it gets called with the event call you back to let you know it's time to go. Eek, that sounds ugly.

Ugly and unnecessary. I sheepishly report that my problem was in assuming that getObjects: emptied its List argument before filling it, but it apparently prepends new objects to any already in the list :-(So the crash occurred when the first object from the previous (closed) document was sent a message. Leaving previous documents open worked fine, although each was

processed repeatedly :-)

Here's my IBMole client template that opens each nib argument, allows modifications to the active document itself, allows modifications to each object in the active document, saves the changes, and closes the document. It even attempts to recover gracefully should IB die or be terminated by a user :-)

template.m ¬template.m

I've been trying to figure out a way to rename outlets and delete actions using IBMole. So far, I've been unsuccessful. I've

been able to open a nib and open the Class Inspector to the desired class, but I then have to make the changes manually. This is pretty powerful as is, but I would like to be able to do all of this programmatically to minimize the chance for errors. Do you have any ideas?

The magic incantation to open the File's Owner Class Inspector is:

```
[activeDocument editClass:
      [[activeDocument rootObject] className]];
```