Subject: Re: recording macros
Posted by JD Smith on Thu, 19 Jun 2003 23:38:52 GMT
View Forum Message <> Reply to Message

On Thu, 19 Jun 2003 14:44:22 -0700, Ben Tupper wrote:

> Reimar Bauer wrote:
>> Dear Ben and Liam,
>>
>> with journal only the call of the routines could be saved but what is
>> if it is a widget.
>>
>> For widgets I have in principle a different solution.
>>
>> Now you know which things are changed and you have only to setup the
>> makro. I would suggest to write a procedure as I did. Because then this
>> could be used without any additional routine to reproduce the result it
>> could be archived and the users could add idl commands if they like to
>> do it.
>>
>>
>>> If it's run from the command line, could you use journalling, i.e.,
>>>
>>> IDL> journal, 'my_commands.pro' ; start journalling IDL> (user
>>> enters commands)
>>> IDL> journal ; stop journalling
>>>
>>> To replay the session:
>>>
>>> IDL> @my_commands.pro
>>>
>>>
> Thanks Liam and Riemar,
>
> Right now, I don't expect the user to run from the command line since
> the GUI is available (and, for the user, less intimidating.)
>
> Since the data format doesn't change (it's always an image) I have an
> advantage over more generic data handling. I think that I could add a
> RecordMacro method into the event handling to provide. I would have
> to have a RECORD property/flag that tells the handler when to record
> the step and when not to. The behavior I hope to see is similar to that
> of ImageJ (http://rsb.info.nih.gov/ij/) . When recording, a simple text
> editor appears listing the steps taken. Here's an example from
> ImageJ...
>
>
>> run("Threshold...");

>> setThreshold(48, 170);
>> run("Threshold", "thresholded remaining black"); run("Despeckle");
>> run("Add...", "value=25");
>> run("Measure");
>
>
> In the IDL case, each set corresponds to a method which could be called
> in order by CALL_METHOD.   The trick, I think, might be the arguments,
> such as the ADD, value = 25. Perhaps the macro should include the
> object class, the method and any arguments... I don't know what to do
> with keywords.
>
> ClassName Method P1 P2 P3 P4 PN
> HBB_HISTOGRAM SETTHRESHOLD 48 170
> HBB_IMAGE ADD 25
>
> I'll have to think about that.
>

If these are objects, why not encode the macro history within the
object itself? I can imagine keeping, e.g., self.history, as a
pointer to a list of structures like:

```
st={MACRO_ENTRY, $
    method: '', $
    args: ptr_new(), $
    kwds: ptr_new(), $
    undo_data: ptr_new()}
```

with args as a pointer to a list of pointers to individual arguments,
and kwds as a pointer to an _EXTRA style structure, ala {KEYWORD: value,
KEYWORD2: value2}. Re-running a macro is then as easy as (assuming
procedure-methods only):

```
n=ptr_valid(macro.args)?n_elements(*macro.args):0
 extra=ptr_valid(macro.kwds)?*macro.kwds:{____NOTAKEYWORD:0b}
case n of
   0: call_method,macro.method,self,_EXTRA=extra
   1:  call_method,macro.method,self,*(*macro.args)[0],_EXTRA=extra
   2:  call_method,macro.method,self,*(*macro.args)[0],*(*macro.arg s)[1], $
                _EXTRA=extra
   ...
endcase
```

Of course, you have to use an ugly CASE to treat differing numbers of
parameters, and you'll probably run out of steam before reaching the
64000 parameter limit, but in practice, 10 ought to do it. The other
option which avoids this hackery is to use EXECUTE with a hand-built

statement, but this is slow(er), and precludes the use of, e.g., large arrays of input (without lots of temporary variables). A hybrid would be to build an execute statement using the argument data from the pointer heap (though there's also a finite limit on the length of the EXECUTE string):

```
void=execute('self->'+macros.method+','+strjoin('*(*macro.args)['+ $
        strtrim(indgen(n),2)+']',',')+',_EXTRA=extra')
```

If your methods are meaty and take a while to run, the EXECUTE overhead won't be noticeable. Another clever idea to go along with this would be to bundle a snapshot of the data being processed before the macro was completed with the macro entry itself (UNDO_DATA above), for easy UNDO/REDO functionality. For popping macro entries off the history and removing them, HEAP_FREE is your friend.

You can save either just the self.history structure to an IDL savefile, or the entire object itself. I'd vote for the object, since then you get all the data in its present state. If you want to save sans the UNDO data to save space, just "prune" it out before saving by setting to a null pointer, and restore afterwards.

JD
