Subject: Encapsulating IDL snippets into objects
Posted by Robert Barnett on Wed, 11 Jan 2006 08:11:40 GMT
View Forum Message <> Reply to Message

G'Day,

I'm creating a fairly complicated GUI. Everything in my GUI is written using OO, however, I feel that I could make my GUI more extensible if I could include snippets of IDL code to perform simple arithmetic operations.

I have encapsulated most of my visualisation tasks into objects very sucessfully. These include curve display, table display, image display, drawing rois and curve fitting amongst others. These fit into a framework which allows the user to jump to any particular task and make changes. I've included a notification mechanism for ensuring that changes are propogated across the framework. The precise details of how the framework works is not really important, it's just important to realise that the framework is complicated enough that most things need to be encapsulated in objects.

The problem is, I get a bit concerned on straight forward procedural things. For example, OO is really overkill for simple things like image arithmetic. I can see proof of this in iTools where *every* action must be encapsulated into an IDLitCommand. Whilst this enables certain kinds of high level OO parafanalia (like undo and macros), I'm afraid that I cannot see any benefit in following RSI's footsteps.

My current idea is to encapsulate snippets of code into an object. The sippets of code see object fields as though they are heap variables within the snippets scope. This allows a programmer (or user) to write a small amount of code without actually realising that they are in an object. I call this object a "process" object.

Encapsuating code in this way might seem like splitting hairs. I'm simply just interested in feedback on the idea. Some example code is attached:

```
;+
;
; This is a class for encapsulating aribitrary
;  procedural code into an object
;
;-

pro nmtkgetvars
common nmtkprocess, obj
```

```
obj -> getvars
end

pro nmtksetvars
common nmtkprocess, obj
obj -> setvars
end

function nmtkprocess::complete
common nmtkprocess, obj
obj = self
call_procedure, self.call_pro
end

function nmtkprocess::GetPropertyByIdentifier, identifier, value
inds = where(*self.identifiers eq identifier,count)
if (count gt 0) then begin
   if ( self.ptr_mode) then begin
      value = (*self.values)[inds[0]]
   endif else begin
      value = (*self.values)[inds[0]]
      if (n_elements(*value) gt 0) then value = *value $
      else return, 0
   endelse
   return, 1
endif else begin
   return, self -> $
     IDLitComponent::GetPropertyByIdentifier(identifier,value)
endelse
end

pro nmtkprocess::getvars
self.ptr_mode = 1b
properties = self -> QueryProperty()
for i=0,n_elements(properties)-1 do begin
   if (self -> GetPropertyByIdentifier(properties[i],value) $
      && (n_elements(value) gt 0)) then begin
      type = size(value,/type)
      case (type) of
         10: begin
            if (n_elements(*value) gt 0) then $
              (SCOPE_VARFETCH(properties[i],LEVEL=-2,ENTER=1)) =$
              temporary(*value)
         end
         else: (SCOPE_VARFETCH(properties[i],LEVEL=-2,ENTER=1)) =$
           value
      endcase
   endif
```

```
endfor
self.ptr_mode = 0b
end


pro nmtkprocess::setvars
self.ptr_mode = 1b
properties = self -> QueryProperty()
for i=0,n_elements(properties)-1 do begin
   if (self -> GetPropertyByIdentifier(properties[i],value) $
      and $
      (n_elements(SCOPE_VARFETCH(properties[i],$
                           LEVEL=-2, ENTER=1)) gt 0)) $
   then begin
      type = size(value,/type)
      case (type) of
         10: $
          *value =$
          temporary($
                  SCOPE_VARFETCH(properties[i],LEVEL=-2))
         else: self -> SetPropertyByIdentifier, $
          properties[i],$
          SCOPE_VARFETCH(properties[i],LEVEL=-2)
      endcase
   endif else begin
   endelse
endfor
self.ptr_mode = 0b
end

pro nmtkprocess::cleanup
ptr_free, self.identifiers
for i=0l,n_elements(*self.values)-1 do $
  ptr_free, (*self.values)[i]
ptr_free, self.values
self -> IDLitComponent::Cleanup
end

function  nmtkprocess::init, identifiers, call_pro,_REF_EXTRA=ex
if (~self -> IDLitComponent::init(_EXTRA=ex)) then return, 0
self.identifiers = ptr_new(identifiers)
self.values = $
  ptr_new(ptrarr(n_elements(identifiers),/allocate_heap))
self.call_pro = call_pro
for i=0l,n_elements(identifiers)-1 do begin
   self -> registerproperty, identifiers[i], 0
endfor
return, 1
```

```
end

pro nmtkprocess__define, struct
struct = {nmtkprocess, INHERITS IDLitComponent,  $
         identifiers: ptr_new(), $
         values: ptr_new(), $
         ptr_mode: 0b, $
         call_pro: '' $
        }
end


pro nmtkprocess__testpro
nmtkgetvars ; Get object fields and move them into the current context
if (n_elements(test) eq 0) then test = fltarr(100000l) $
  else test = temporary(test) + 1
harry = "Hypocondriact"
name = 'A'
description = 'B'
nmtksetvars ; Neatly put them back in the original context
end

pro nmtkprocess__test
; Create a process with two user defined fields. Run the given
; procedure when complete() is called.
obj =  obj_new('nmtkprocess',['TEST','HARRY'],'nmtkprocess__testpro ')
t = systime(1)
for a=0l,1000 do $
  result = obj -> complete()
print, systime(1) - t
if (obj -> GetPropertyByIdentifier('TEST',test)) then help, test,$
test[0]
if (obj -> GetPropertyByIdentifier('HARRY',harry)) then help, harry
obj_destroy, obj
help, /heap
end
```