Subject: Re: Classes and Widget Event Handlers...
Posted by Struan Gray on Sat, 03 Jul 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Morris, dmorris@ball.com writes:

> The event-handler for the widget needs to have access to
> all of the variables of the class, and it must have direct
> access to them. Unfortunately, XMANAGER (as far as I can
> tell) does not allow the use of object method-routines as an
> event handler.

If you make your widget an object that inherits the class whose variables you want to access, it can do so by using the standard self.xxx syntax. If you want the object to have a life of it's own beyond that of the widget you will have to make a way for the widget to swallow and regurgitate the object when it is created and killed. Your only other option is to write a generalised GET/SET_PROPERTY method like those used in the objects graphics classes.

This seems to have come up a lot recently, so here's Mark Rivers' (rivers@cars.uchicago.edu) original example of how to make an widget into an object:

```
>
>     IDL Objects really simplify complex widget programs,
>  because so don't have to create a "state" structure and
>  stick in the uvalue of a top level widget.  Just put the
>  object reference to "self" there instead.  This is not
>  entirely obvious, and the Object manual does not have an
>  example, so here is how I do it.
>
>  ;;  File example__define.pro
>
>  function example::init
>     base = widget_base(uvalue=self)
>     self.widgets.base = base
>     self.widgets.exit = widget_button(base, value='Exit')
>     widget_control, base, /realize
>     xmanager, 'example::init', base, event='example_event', /no_block
>     return, 1
>  end
>
>  pro example_event, event
>     ; Note: The main event handler CANNOT be an object method,
>     , since xmanager won't know how to call it as such.  However,
>     ; it only needs to be 4 lines long: retrieve the object
>     ; reference and call the object method event handler,
>     ; which will know about the objects data structure.
>     widget_control, event.top, get_uvalue=object
>     object->event, event
>  end
>
>  pro example::event, event
>     ; This is the event handler which knows about the object
>     case event.id of
>        self.widgets.exit:  begin
>           widget_control, event.top, /destroy
>        end
>        else: print, 'Unknown widget event'
>     endcase
>  end
>
>  pro example__define
>     widgets={example_widgets, base: 0L, exit: 0L}
>     data = fltarr(30)
>     t = {example, widgets: widgets, data: data}
>  end
>
```

You create the widget with NEWWID = OBJ_NEW(example). IDL scurries off and finds the EXAMPLE__DEFINE file, calls the EXAMPLE::INIT method and away we go. Note that in the event handling method all the widget's info can be accessed with the self.xxx syntax.

When this was originally posted in June '97, Ronn Kling and Bob Mallozzi pointed out you could exploit the way that IDL calls and defines event handlers. Bob's method sneakily just called the event handler EXAMPLE::INIT_EVENT and the XMANAGER then did it's usual thing of appending "_EVENT" to the name of the creation routine to find the default handler. Ronn Kling used the EVENT_PRO keyword when creating the top level base to define EXAMPLE::EVENT as an event procedure.

Both techniques look powerful (and, more importantly, cool :-), but both seem to rely on the ability to invoke an object method as if it were a normal procedure, something that should, formally, be impossible. I've not delved deeply enough into the innards of IDL to know if this 'bug' lies in the EXECUTE function used to convert strings to commands, or if there is general leakage between functions, procedures and methods. I follow Mark's original scheme, and feel that it makes more sense to lobby RSI to make the XMANAGER object-aware than to rely on a quirk of IDL syntax that may change at the next release.

Struan