Subject: Sharing Variables within a Widget
Posted by john Slavich on Mon, 17 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

To anyone who can help me,

I'm in the process of building my first widget program and now I'm at an impass. This is how my widget is presently srtuctured:

-> Function 1
-> Function 2
-> PRO, Event
-> PRO (main)

Function 1 and 2 have no widgets, they are called upon to make calculations on data set. Function 1 requires additional information calculated in Function 2 (i.e. the data Func. 1 requires is not part of the Return from Function 2). Since Functions can only return 1 "thing," I would like to store information, calculated in Function 1 and 2, into the top event, so that I may call upon the information in other functions or procedures which are not widgets. For example, in the PRO event, I can call up information stored in a structure from the PRO main, by typing:
Widget_control, event.top, Get_UValue=info
All the info I need can be accessed in the structured called "info," but when I do the same thing in Function 1 and 2, it doesn't work.
I hope this is making sense, I'm still the learning process.

Thanks to anyone who's willing to help.

John

jcslavic@ouray.cudenver.edu

Subject: Re: Sharing Variables within a Widget
Posted by davidf on Mon, 17 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

JD Smith writes an extremely lucid and helpful reply to John Slavich's widget programming question. But I would like to further clarify one of his points, which I *know* will be misinterpreted by someone.

JD writes:

> I think your difficulty is with understanding event procedures. Your PRO,Event is the
> *only* procedure which is automatically passed an event (i.e., is the

only procedure
> which can say 'Widget_control, event.top, Get_UValue=info'). To let a function have
> access to your info structure, simply pass it as an argument. You can do this from
> one of two places: in your widget setup routine (where you define the structure info),
> or in your PRO, Event routine (where your Widget_control call retrieves info). Here's
> an example:
>
> pro my_event_handler, event
> Widget_control, event.top, Get_UValue=info
>
> answer1=func1(arg1,arg2,...,info) ; info passed on to answer1
> end
>
> and an example function ;
>
> function func1, arg1,arg2,...,info
> info.item1=some_crazy_calculation
> end
>
>
> This works because structures are passed by reference, and so setting a field of info
> inside func1,or func2 (or both) changes the 'info' in the calling routine.

As a matter of fact, this example of JD's *won't* work exactly as he claims. It is true that calling the function in the event handler and passing the info structure to the function so that it can be changed in the function will work. At least it will work IN THE EVENT HANDLER. In other words, if you print the new info structure in the event handler after you return from the function call, you will see the field has changed.

But if you depend on that new field value in subsequent program operations, you will be sorely disappointed. The reason is that you have made the change to a LOCAL VARIABLE! That is, the info structure in the event handler is a LOCAL variable, copied into the event handler from a global storage location (the user value of the top-level base widget). In order to make the change "permanent", you must copy the LOCAL info structure back to the global storage location before you leave the event handler. This is what is missing from JD's example code, but not, I feel confident, from his real code.

The correct sequence of events will look like this:

```
pro my_event_handler, event
```

```
      Widget_control, event.top, Get_UValue=info
      answer1=func1(arg1,arg2,...,info) ; info passed on to answer1
      WIDGET_CONTROL, event.top, SET_UVALUE=info
  end

 and an example function ;

  function func1, arg1,arg2,...,info
      info.item1=some_crazy_calculation
  end
```

And (just to nitpick for the cognosenti) I would do the getting and setting of the user value with the NO_COPY keyword. No sense making multiple copies of the same data. :-)

Cheers!

David

David Fanning, Ph.D.
Fanning Software Consulting
2642 Bradbury Court, Fort Collins, CO 80521
Phone: 970-221-0438 Fax: 970-221-4762
E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com

Subject: Re: Sharing Variables within a Widget
Posted by J.D. Smith on Tue, 18 Mar 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David,

Thanks for correcting my oversight... Quite right. Local variables remain local until exported back to the user value (or handle) from which they were obtained.

Sorry for any confusion.

JD
> ---

Subject: Re: Sharing Variables within a Widget
Posted by peter on Tue, 18 Mar 1997 08:00:00 GMT

john Slavich (jcslavic@ouray.cudenver.edu) wrote:
: To anyone who can help me,

: I'm in the process of building my first widget program and now I'm at
: an impass. This is how my widget is presently srtuctured:

: -> Function 1
: -> Function 2
: -> PRO, Event
: -> PRO (main)

: Function 1 and 2 have no widgets, they are called upon to make
: calculations on data set. Function 1 requires additional information
: calculated in Function 2 (i.e. the data Func. 1 requires is not part of
: the Return from Function 2). Since Functions can only return 1 "thing,"
: I would like to store information, calculated in Function 1 and 2, into
: the top event, so that I may call upon the information in other
: functions or procedures which are not widgets. For example, in the PRO
: event, I can call up information stored in a structure from the PRO
: main, by typing:
: Widget_control, event.top, Get_UValue=info
: All the info I need can be accessed in the structured called "info,"
: but when I do the same thing in Function 1 and 2, it doesn't work.
: I hope this is making sense, I'm still the learning process.

Presumably, function1 and function2 are called from the event handler.
So, retrieve the uvalue in the event handler (as you are doing) then
pass that info to f1 and f2. After f1 and f2 are done with the
structure, you'll return to the event handler, which can then put the
modified structure back into the uvalue of the widget.

This makes an appropriate clean separation between routines that build
and manage the widget, and routines that process data.

A more general comment: functions can return only one thing, but
procedures (and functions) can change as many of their arguments as you
want, since IDL used call-by-reference argument passing (most of the
time!). So it is possible to "return" as many things as you want.

Hope this helps,

Peter