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

john Slavich 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.
>
> Thanks to anyone who's willing to help.
>
> John
>
> jcslavic@ouray.cudenver.edu

John,

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
```

or

```
  pro widget_making_routine,...
 info={item1:0L,item2:1,...}

 answer2=func2(arg1,arg2,...,info)
  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.

Note that if all you need to do is set a single element of info with a function call, it's easier to say, e.g.:

```
 info.item1=func1(arg1,arg2,...)
```

in either of the two places mentioned above.

Another helpful tip for you concerns returning "things" (as you put it) from functions. You can actually return as many items as you like from both functions and procedures. Simply pass them as an argument. This basically also hinges on the fact that almost everything in IDL is passed by reference. So, I can say, e.g.

```
 answer1=func1(input1,answer2,answer3,..)
```

and if I put a variable into answer2, etc, I can set that variable in the function and its value will persist even after that function exits (the very definition of passing by reference!). A few caveats: explicit expressions (e.g. numbers), structure *items* and indexed variables are passed by value (NOT by reference), and cannot be modified outside of their called environment. I *cannot* therefore, say:

```
 a=func(b,c.item1,d[5],5.1)
```

and expect to set the 'item' field of my stucture c, or the 5th element of my array d, inside func, and have those values persist once func exits (the very definition of

passing by value). It's also obvious setting the calling environment constant 5.1 has no meaning.

Good Luck,

JD
