Subject: Re: Passing variables
Posted by Conor on Thu, 23 Aug 2007 13:34:42 GMT
View Forum Message <> Reply to Message

On Aug 22, 4:07 pm, jeffw...@hotmail.com wrote:
> I am having an issue passing variables from one program to another.
>
> I have a main program with a gigantic for loop. Each time through the
> loop there is a different value from a "B" array. Inside the loop I
> use rk4 to create fieldlines, with the derivatives ("dydx") calculated
> in another program. rk4 only allows you to give it a vector (say
> Xo,Yo) and it gives you back another vector (say X1,Y1). But I need,
> somehow to pass the value from my "B" array to the second program (the
> one that returns the "dydx").
>
> I tried putting a procedure in the middle of my main program along the
> lines of:
>
> pro getb,dummy
> return, [B(n)]
> end
>
> and then in the dydx program had a line with:
>
> myB = getb(0)
>
> But that didn't seem to work. Does anyone know an easy way to pass
> this B variable into this second program?
>
> Thanks.

The easiest way is typically to use a common block. At the start of
everyone program that want to access some variables, put a line like
this:

common, block_name, variable1, variable2, ...

A couple rules about common blocks:

1) The names of the variables don't have to be the same in every
program.
2) The number of variables matter. All common blocks should have the
same number of variables.
3) The positions of the variables matter. Since names don't matter,
the first variable in one program will always be the same as the first
variable in another program, even if they have much different names
4) Common blocks last until you quit IDL. If you declare it once, it
stays declared.

5) Common blocks can't be extended. This means that if you create a common block and later realize you need to add more variables to it, you'll have to quit IDL (or type .reset) in order to "destroy" the common block.

For more on common blocks, see the online_help

Your getb program doesn't work for two reasons. First, you have a bug. I suspect you meant to say:

```
pro getb,n
return, [B(n)]
end
```

or:

```
pro getb,dummy
return, [B(dummy)]
end
```

Also, it doesn't work because of variable scope. A sub-routine automatically runs in it's own 'scope' which means that it can only access variables it has been passed directly, it has created itself, or that it retrieves from a common block. In the case above the 'B' variable meets none of those criteria and is inaccessible to your getb function. You could make this work anyway, using the scope_varfetch function, but you are much better off using a common block, especially since usage of scope_varfetch should be avoided.