Subject: Re: Preserving Variables
Posted by davidf on Tue, 28 Oct 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Matthew Hanson (matt@ktaadn.com) writes:

> Anyone know how to make statis variables? That is, upon returning to a
> function, how can i create a variable that has the same value as when i
> left the function before? I really don't want to use Common Blocks.

I'm not sure how this could be done in a regular IDL
function without common blocks (I say this knowing I
am risking my reputation among students who have heard
my anti-commmon block lectures :-). Perhaps with Save/Restore
functionality, but I have to think, why bother?

In a widget program this kind of thing is easily accomplished
with either pointers or user values. See any of the widget
programs on my web page for examples. I guess you could do
it with pointers with a regular function too, as long as
the pointer was passed into the function as a parameter.

```
   output = My_Function(ptrToStaticVariables)
```

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Preserving Variables
Posted by R. Bauer on Wed, 29 Oct 1997 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

> Matthew Hanson (matt@ktaadn.com) writes:
>
>> Anyone know how to make statis variables? That is, upon returning to a
>> function, how can i create a variable that has the same value as when i
>> left the function before? I really don't want to use Common Blocks.
>
> I'm not sure how this could be done in a regular IDL

> function without common blocks (I say this knowing I
> am risking my reputation among students who have heard
> my anti-commmon block lectures :-). Perhaps with Save/Restore
> functionality, but I have to think, why bother?
>
> In a widget program this kind of thing is easily accomplished
> with either pointers or user values. See any of the widget
> programs on my web page for examples. I guess you could do
> it with pointers with a regular function too, as long as
> the pointer was passed into the function as a parameter.
>
>   output = My_Function(ptrToStaticVariables)
>
> Cheers,
>
> David

An idea may be to use a systemvariable as readonly

Re

--
R.Bauer

Institut fuer Stratosphaerische Chemie (ICG-1)
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de

Subject: Re: Preserving Variables
Posted by thompson on Thu, 30 Oct 1997 08:00:00 GMT

View Forum Message <> Reply to Message

Matthew Hanson <matt@ktaadn.com> writes:

> Hello,
> Anyone know how to make statis variables? That is, upon returning to a
> function, how can i create a variable that has the same value as when i
> left the function before? I really don't want to use Common Blocks.
> -matt

I understand your concern about common blocks, but if a common block is
restricted to a single routine for the sole purpose of retaining information
between calls, then I think most of those concerns disappear. I make such a
"private" common block have the same name as the routine, to avoid running into
a common block of the same name from another routine.

The other way to preserve data, of course, is to use DEFSYSV to define a global system variable, but that can be rather awkward.

It would be nice if IDL had a RETAIN statement, similar to the COMMON
statement, that could be used to retain internal variables between calls.

Bill Thompson

Subject: Re: Preserving Variables
Posted by Stein Vidar Hagfors H on Fri, 31 Oct 1997 08:00:00 GMT
View Forum Message <> Reply to Message

William Thompson wrote:
>
> Matthew Hanson <matt@ktaadn.com> writes:
>
>> Hello,
>> Anyone know how to make statis variables? That is, upon returning to a
>> function, how can i create a variable that has the same value as when i
>> left the function before? I really don't want to use Common Blocks.
>> -matt
>
> I understand your concern about common blocks, but if a common block is
> restricted to a single routine for the sole purpose of retaining information
> between calls, then I think most of those concerns disappear. I make such a
> "private" common block have the same name as the routine, to avoid running into
> a common block of the same name from another routine.

I agree - this is a very good use of common blocks. I often use
similar constructs to make e.g., a cache for stuff that is
computationally expensive, or needs to load data etc. from files.
It's really one of the best things to use common blocks for.

E.g.:

```
FUNCTION pick_entry,id
  common pick_entry_cache,ids,values

  if n_elements(ids) eq 0 then begin
    <read ids, values from a file>
  endif

  return,values(where(ids eq id))
end
```

> The other way to preserve data, of course, is to use DEFSYSV to define a global
> system variable, but that can be rather awkward.

Yep - you basically have to put the defsysvar statement in the startup
file, because IDL refuses to compile any program referring to an
as yet undefinded system variable (unless you package it into an
execute statement - but that's even more cumbersome).

Stein Vidar