Subject: Re: Global variables and command line
Posted by J.D. Smith on Fri, 17 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Daniel SAGE (daniel.sage@epfl.ch) writes:
>
>> I would like use the variables created in the input command line inside
>> the procedures/functions. But in my application where I automatically
>> generate IDL code, I can't use the common mechanism or passing by
>> parameters.
>
>> How is it possible to make V2 visible inside the test procedure when I
>> can't pass the parameters with the run procedure ?
>
> Well, at the risk of being too obvious here, I would suggest
> putting V2 *in* the common block. :-)
>
>> IDL> common var, v1
>
> to IDL> common var, v1, v2
>
> 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/

At the risk of being too arcane here, I think Daniel's point is quite a valid one, and his problem less easily solved than it might at first glance seem. Sometimes, one wants to access variables from the $MAIN$ level in a routine, and sometimes one wants to put variables created inside a routine into $MAIN$. You might wonder what circumstance could exist that would require this functionality. I would simply then direct you to RSI's own Insight, in which after getting in you have the option of "Select Data to Import"..."IDL Variables"... with a list of $MAIN$ level variables displayed. Now clearly the folks at RSI have convenient access to the names and data locations of the $MAIN$ level variables, and I think they should give us mere ordinary users that access also. As phrased so judiciously in the Reference Guide under the Help procedure:

The OUTPUT keyword is primarily for use in capturing HELP output in order to display it someplace else, such as in a text widget. This keyword is not intended to be used in obtaining programmatic information about the IDL session, and is formatted to be human readable. Research Systems reserves the right to change the format and content of this text at any time, without warning. If you find yourself using OUTPUT for a non-display purpose, you should consider submitting an enhancement request for a function that will provide the information you require in a safe form.

I am henceforth considering submitting an enhancement request for a function that will provide the names and data locations (e.g. through pointers) of $MAIN$ level variables inside routines, and the ability to create $MAIN$ level variables from within routines, in a safe form.

By the by, for creating $MAIN$ level variables, I used to use this mechanism:

Have a common block which contains information on new variables (including their names) created by a program for export into the $MAIN$ level. Have a $MAIN$ level routine as the only interface into the program, and have it call that program. After the program exits, the remainder of the $MAIN$ level routine simply looks in the common block, and creates any exported variables in $MAIN$ (or, e.g just creates new $MAIN$ level pointers to the heap data). This used to work fine. However, with the advent of the non-blocking widget interface, it doesn't work in conjunction with an active command line (the $MAIN$ level program runs through immediately and exits without creating anything).

Of course, you can always hack something, so my present solution for non-blocking applications is to use a little procedure which sets a pointer into a common block, where it can be retrieved from within the program.

E.g. I might say

mark, thisPointer

and have the data *thisPointer available immediately in the program. Indeed, you could take this idea further and have a single $MAIN$ level "import" routine which looks in the relevant common block for any exported pointers, which have their *names* recorded also in the common block, and executes them into $MAIN$ with the same names. But this requires the user to explicitly run the import routine, which not elegant.

As an example, suppose I have some image in a file, call it A. In a fancy widget program, I load A from disk and edit that image. When I exit the program, it would be ideal if A (or a pointer to the heap data which is A) could be created in the $MAIN$ level (as A) if I so choose, for my further analysis and editing. Currently, I would have to run my $MAIN$ level import routine to accomplish this.

Perhaps RSI is trying to protect us from ourselves here, realizing that with the problems people have with variable locality already, breaking it slightly might serve to confuse even more. But they really shouldn't underestimate our ability to harness and control whatever new power comes our way. And besides, if they get to do it, then so should we.

Alright, the rant is over.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-4083
206 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Global variables and command line
Posted by Martin Schultz on Fri, 17 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Daniel SAGE wrote:
>
> Hello,
>
> I would like use the variables created in the input command line inside
> the procedures/functions. But in my application where I automatically
> generate IDL code, I can't use the common mechanism or passing by
> parameters.
>
> pro run
> resolve_routine, "test"
> call_procedure, "test"
> end
>
> pro test
> common var
> print, v1
> print, v2
> end

>
> IDL> common var, v1
> IDL> v1=111
> IDL> v2=222
> % Compiled module: RUN.
> % Compiled module: TEST.
> IDL> run
> % Procedure was compiled while active: RUN. Returning.
> % Compiled module: TEST.
>   111
> % PRINT: Variable is undefined: V2.
> % Execution halted at: TEST   10 HD:Desktop Folder:test.pro
>
> % $MAIN$
>
> How is it possible to make V2 visible inside the test procedure when I
> can't pass the parameters with the run procedure ?
>
> Thanks
>
Your problem is that you define your common block only with v1 - and
once it is defined, you can't change it!!

The above example will work if you type common var,v1,v2 on the command
line. However, a better approach would probably be to call EXECUTE (or
CALL_PROCEDURE, CALL_FUNCTION) to "execute" your procedure or function
that you generate automatically. There should always be some way of
parameter passing. Even if you don't know how many variables you will
need in advance neither their name, you could still construct a
structure and pass that into the function/procedure. I have a CHKSTRU
function in my library (html below) which you can use to see if a
certain structure tag is defined.

Hope this helps,
Martin

--

 -- -------
Dr. Martin Schultz
Department for Earth&Planetary Sciences, Harvard University
186 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
IDL-homepage: http://www-as.harvard.edu/people/staff/mgs/idl/
 -- -------

Subject: Re: Global variables and command line
Posted by davidf on Fri, 17 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Daniel SAGE (daniel.sage@epfl.ch) writes:

> I would like use the variables created in the input command line inside
> the procedures/functions. But in my application where I automatically
> generate IDL code, I can't use the common mechanism or passing by
> parameters.

> How is it possible to make V2 visible inside the test procedure when I
> can't pass the parameters with the run procedure ?

Well, at the risk of being too obvious here, I would suggest
putting V2 *in* the common block. :-)

> IDL> common var, v1

to IDL> common var, v1, v2

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: Global variables and command line
Posted by steinhh on Tue, 21 Apr 1998 07:00:00 GMT
View Forum Message <> Reply to Message

J.D.Smith wrote:

[..snip..]

> At the risk of being too arcane here, I think Daniel's point is quite a
> valid one, and his problem less easily solved than it might at first
> glance seem. Sometimes, one wants to access variables from the $MAIN$
> level in a routine, and sometimes one wants to put variables created
> inside a routine into $MAIN$. You might wonder what circumstance could
> exist that would require this functionality. I would simply then direct
> you to RSI's own Insight, in which after getting in you have the option

> of "Select Data to Import"..."IDL Variables"... with a list of $MAIN$
> level variables displayed. Now clearly the folks at RSI have convenient
> access to the names and data locations of the $MAIN$ level variables,
> and I think they should give us mere ordinary users that access also.

[..snip..]

> I am henceforth considering submitting an enhancement request for a
> function that will provide the names and data locations (e.g. through
> pointers) of $MAIN$ level variables inside routines, and the ability to
> create $MAIN$ level variables from within routines, in a safe form.

[..snip..]

> Perhaps RSI is trying to protect us from ourselves here, realizing that
> with the problems people have with variable locality already, breaking
> it slightly might serve to confuse even more. But they really shouldn't
> underestimate our ability to harness and control whatever new power
> comes our way. And besides, if they get to do it, then so should we.

I agree.

In an attempt to nominate myself for the 'IDL Hacker of the year'
award :-) I dug into something that I'd seen while programming RPC
calls to IDL ... and ended up wasting a days work on this thing that
doesn't seem to work, even though I think it *should* work;

Below is a set of C routines that need to be compiled and linked into
shareable libraries (and remember to include the "libidl" shareable
library in the linking... on my alpha that's done by adding e.g.,
"-L$IDL_DIR/bin/bin.alpha -lidl" to the linker statement). The
easiest way to find the (other) correct compile/link flags is
to modify the $IDL_DIR/external/sharelib/Makefile to include your
file.

The getmain() function is added as a System Routine (very neat - at
least I learned *this* from my little exercise!), but it doesn't
work as expected:

```
IDL> print,CALL_EXTERNAL('gmain.so','addmain')  ;; Install routine
           1
IDL> print,getmain('ADF')
% GETMAIN: Couldn't find variable     ; As expected
% Execution halted at:  $MAIN$
IDL> adf=1
IDL> print,getmain('ADF')
Main name ADF, at 140033a58
Main variable type/flag: 2 0
```

```
Copying:
Returning: 2 2
      1                         ; So far, so good...
IDL> .r
- pro test
- print,getmain('ADF')
- end
% Compiled module: TEST.
IDL> test
Main name ADF, at 140033c98        ; <- address is wrong, but not NULL!
Main variable type/flag: 0 0
Copying:
Returning: 0 2
% PRINT: Variable is undefined: <UNDEFINED>.
% Execution halted at:  TEST               2 /dev/tty
%                     $MAIN$
```

So, does anyone know what's going wrong here...? Does it behave like this for all platforms...?

Regards,
Stein Vidar

C source follows (void for non-nerds.. :-)

```
 ------------------------------------------------------------ --gmain.c
#include <stdio.h>
#include "export.h"
#include <strings.h>

/*
  Initialize with:

  print,CALL_EXTERNAL('gmain.so','addmain')

  then, you *should* be able to get the value of a non-dynamic
  main-level variable with:

  print,getmain('<VARNAME>')

  Alas, this only works from the $MAIN$ scope - which is a bit
  silly....

  */

#define NULL_VPTR ((IDL_VPTR) NULL)

IDL_VPTR getmain(int argc, IDL_VPTR argv[])
{
```

```
  IDL_VPTR main_variable;
  IDL_VPTR retv;
  IDL_VPTR main_name;

  retv = IDL_Gettmp();

  main_name = argv[0];

  IDL_ENSURE_STRING(main_name);
  IDL_ENSURE_SCALAR(main_name);

  main_variable = IDL_GetVarAddr(main_name->value.str.s);

  if (main_variable == NULL_VPTR) {
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_LONGJMP,"Couldn't find variable");
  }

  printf("Main name %s, at %lx\n\r",main_name->value.str.s,main_variable);
  printf("Main variable type/flag: %d %d\n\r",(int)main_variable->type,
    (int)main_variable->flags);

  if ( ! (main_variable->flags & IDL_V_DYNAMIC) ) {
    fprintf(stderr,"Copying: \n\r");
    bcopy(main_variable,retv,sizeof(*retv));
    retv->flags |= IDL_V_TEMP;
  }

  fprintf(stderr,"Returning: %d %d\n\r",retv->type,retv->flags);

  return retv;
}



IDL_SYSFUN_DEF main_def[] = { {(IDL_FUN_RET) getmain, "GETMAIN", 1, 1} };

IDL_LONG addmain(int argc,char *argv[])
{
  int tmp;

  /* I haven't the faintest idea whether or not this statement is required,
     or whether it's completely out of line... */

  tmp=IDL_Init(0,&argc,argv);

  /* This one was neat, though: */

  IDL_AddSystemRoutine(main_def,IDL_TRUE,1); /* Just add getmain... */
```

```
  return tmp;
}
 ------------------------------------------------------------ ----------
```