Subject: Re: IDL excels in debugging??? Do you know something I dont?
Posted by chase on Fri, 19 May 1995 07:00:00 GMT
View Forum Message <> Reply to Message

>>>> > "Mark" == Mark Rivers <rivers@cars3.uchicago.edu> writes:
In article <D8sBMD.20o@midway.uchicago.edu> rivers@cars3.uchicago.edu (Mark Rivers) writes:

>> The most useful debugging technique (other than the good ole PRINT statement)
>> I know of is the following 2 line routine, offered to me once by
>> rep2857@sbsun0010.sbrc.hac.com (Mike Schienle)
>>
>> ; BREAK.PRO: a "debugging" routine. it always causes an error. Period.
>> ; A call to 'break' in IDL will break IDL and return to the routine
>> ; which called it, allowing you to examine all variables' values at
>> ; the point it was called. There is generally no way to continue execution,
>> ; you must "RETALL & XMANAGER" (aargh!).   R. Welti; from M.Schienle
>>
>> PRO
>> END
>>
>> In fact, I would love to read a discussion of what other people are using
>> for debugging techniques / tools.

Mark> Why not just use the STOP statement in your routine? It stops
Mark> IDL, leaving you at the command line, allowing you to examine
Mark> all variables' values, etc. without generating the error. Once
Mark> you are done examining variable, etc. you can continue on by
Mark> just typing .CON.

STOP will not work with event callback routines. The above does work when you install callback routines with xmanager. Breakpoints do not work in callback routines either. When a stop or breakpoint is encountered the IDL execution context is in XMANAGER and not the routine where you wanted to stop.

The above break.pro is a clever idea. After causing the break you can skip over it using .skip.

For my own debugging of callback routines, I print out a undefined variable (e.g., "print,dummyvar") within the routine where I want to cause a break. After the stop in execution I would define the variable and issue a .continue.

One thing of note regarding debugging. There was a comment that one can do a lot more in standard machine code debuggers (e.g. using xdb to debug compiled C code).

Even though there are some problems with IDL's handling of
breakpoints, there are advantages to debugging in IDL over C debuggers
or debuggers in general for machine language programes. Once
execution is stopped, I can look at any variables, change variable
values (even to new types), recompile other programs (without having
to "exit the debugger" and lose data), even define new variables
(until the symbol talble for the procedure fills up), execute other
procedures and functions. The C debugger that I use does _not_ let me
call functions and procedures within the current context, define new
variables, change variables (to a new type/size), recompile. I have
heard of interactive C debugger/interpreters, but I do not know what
their full capabilities are. Without an interpreter, the variety of
expressions that one can use in a C debugger is not as rich as what
can be used via the command line of the IDL interpreter.

In my experience, debugging IDL code is much easier than debugging C
code (or whatever your favorite High level compiled language might
be). IDL just needs some small improvements in breakpoint handling
and the addition of examining variables within different execution
contexts along the calling chain (i.e. examining variables in the
calling routine and above. I think this may already be provided in
the undocumented function routine_names. Perhaps it is supported in
IDL v4.0?).

Chris

--
==============================
Bldg 24-E188
The Applied Physics Laboratory
The Johns Hopkins University
Laurel, MD 20723-6099
(301)953-6000 x8529
chris.chase@jhuapl.edu