Subject: IDL excels in debugging??? Do you know something I dont?
Posted by Russ Welti on Thu, 18 May 1995 07:00:00 GMT
View Forum Message <> Reply to Message

Wally Gross wrote:

> >It's nice to know that a well written IDL program can be faster than
> >the corresponding C program, but on the projects I've been on speed
> >wasn't the most important concern. Getting the system developed
> >and debugged was always the biggest problem. In that area, IDL really
> >excels though systems like Turbo C for the PC are terrific for
> >development and debugging also. IMHO the real challenge with IDL is
> >creating maintainable code.

I agree about the maintainable code problem. Being interpreted (IMHO)
tends to make any language at risk for quick and dirty programming --
prototypes and experimental code changes that "never got documented".
But then we chose IDL *because* we wanted rapid prototyping capabilities ;)

Of course I blame myself if I fall into bad habits, but IDL is not very
consistent in its overall design and methods for accomplishing various
programming tasks. For example, there are multiple ways to accomplish
several fundamental operations, sometimes undocumented, and without
the kind of overall simplicity of concept that characterizes, for example,
object-oriented paradigms. But IDL *evolved*; it was not "designed".

In IDL, online help is really a must, because the lack of consistency
means one must constantly refer to each procedure/function to remember
how that particular creature works... Early in my exposure to IDL I read
here that "IDL is a hacker's language". I have often reflected on that...

I would REALLY take exception to the statement that IDL is debuggable.
If you have used a good Unix debugger, it is hard to even compare the total
lack of debugging tools (usable anyway) in IDL. You may notice this is a
hot button for me. Breakpoints are a JOKE. Even perl has a very useful,
line-oriented debugger which RSI should use as a model, I believe.
idltool is an admirable attempt, but unwieldy and unreliable.

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.

Russ Welti
University of Washington
Molecular Biotechnology
PO Box 352145
Seattle, WA 98195
rwelti@u.washington.edu
(206) 685 3840 voice (206) 685 7344 FAX
http://chroma.mbt.washington.edu/graphics/gif/russ.gif

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

Subject: Re: IDL excels in debugging??? Do you know something I dont?
Posted by zawodny on Mon, 22 May 1995 07:00:00 GMT
View Forum Message <> Reply to Message

In article <3pg7pi$b1u@newsstand.cit.cornell.edu> patterso@astrosun.tn.cornell.edu (Tim Patterson) writes:
> Mark Rivers (rivers@cars3.uchicago.edu) wrote:
>
> : >In fact, I would love to read a discussion of what other people are using
> : >for debugging techniques / tools.
>
> : Why not just use the STOP statement in your routine? It stops IDL, leaving you
> : at the command line, allowing you to examine all variables' values, etc.
> : without generating the error. Once you are done examining variable, etc. you
> : can continue on by just typing .CON.
>
> I just use control C and .con (or xmanager in a Windows situation)
> to debug my code. But it's not pretty :)
> I'd love it if the step command would actaully just oprint the relevant
> line of code to the screen so I knew where I was in the code.
>

Control C is a bit difficult to use in practice since you really cannot control where you stop. Try stopping in a routine that spends most of its time in other subroutines and you'll see what I mean.

Editing a proceedure to add a STOP gets tiresome after awhile. My biggest gripe when it comes to debugging IDL is that the reported line number is usually wrong. This may be due in part to my programming habits, but it seems that whenever I use an @file to include common blocks and/or 'standard' code or if I have mulltiple statements on the same line (using the & operator) IDL cannot figure out where the offending line is when it bombs. I end up adding a series of lines like

```
print,'I am at A'
```

and recompiling and rerunning the code to narrow down where the error is. I guess it would be nice to have true interactive breakpoint setting. A routine like

SET_BREAK,module_name,line_number {,/cancel_break}

which could be called interactively either before or during (after a STOP or programming error halts execution) would be really useful and speed code development (for me at least). I have no idea what this would do for execution speed (IDL and other interpreted languages are slow enough as it is [reading things like "A well written IDL program can actually run faster than a C program" has kept me from learning C, but I digress further]).

--

Joseph M. Zawodny (KO4LW) NASA Langley Research Center
Internet: j.m.zawodny@larc.nasa.gov MS-475, Hampton VA, 23681-0001
TCP/IP: ko4lw@ko4lw.ampr.org Packet: ko4lw@n4hog.va.usa.na