Subject: Compiling IDL ... ever likey ?
Posted by julien on Tue, 16 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

Are there any plans to provide an IDL compiler which produces standalone
code (including support for X/Windoze) ?

The major advantage would be speed of execution (given a decent compiler).
Technically it is possible (although not easy, I imagine). However, it may not
be in RSI's best interests as standalone code does not require IDL
to run. However, the compiler could come as an expensive add on ...

IDL is great for prototyping. It would be nice to translate that power
straight into something slimline and fast ...

Any comments or gossip ?

--
Julien Flack, CSIRO, division of exploration and mining, Perth, W.A.
julienf@per.dms.csiro.au

Subject: Re: Compiling IDL ... ever likey ?
Posted by James Tappin on Thu, 18 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

It stikes me that the biggest technical problem would be handling the way IDL
is able to change the type of variables without the user knowing explicitly
that it is doing so. While the merit of this practice may be debatable, it is
difficult for me (as a user of serveral languages rather than a compiler
expert) to see an *efficient* compiled code able to do something like:

```
a = indgen(20)
a = a+0.5
```

and have a come out of it as a real quantity. I agree that "a=findgen(20)+0.5"
is probably better from all standpoints anyway but it's merely to illustrate
the point.

Apart from that problem is is not very difficult to translate IDL "analysis"
code into fortran-90. Graphical interfaces are another matter and would need an
"IDL graphics & widgets library".

--
```
 +-----------------------+--------------------------------- --+---------+
| James Tappin,          | School of Physics & Space Research |  O__    |
| sjt@star.sr.bham.ac.uk | University of Birmingham       | --  \/` |
| Ph: 0121-414-6462. Fax: 0121-414-3722          |  |
```

+--- --+---------+

Subject: Re: Compiling IDL ... ever likey ?
Posted by thompson on Sun, 21 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

James Tappin <sjt> writes:

> It stikes me that the biggest technical problem would be handling the way IDL
> is able to change the type of variables without the user knowing explicitly
> that it is doing so. ...

Actually, there's nothing to keep a "compiled" version of an IDL program from acting the same way as in the normal version of IDL. Such a compiled executable would consist of two parts:

1. A program section, which contains a restricted version of the basic IDL executable. It would be able to do everything IDL can do, except compile procedures, or accept commands from a command line.

2. A data section, which contains the IDL procedures to be executed, in a binary interpreted format--presumably the same format that IDL stores the procedure in a SAVE file.

Such an object would act just like an executable--i.e. it would be a single file that one would simply run--but it would preserve all the interpretive-language advantages that IDL currently has. It would still be IDL.

Some people might be disappointed that the performance would be the same. There would not be the performance increase that compiled binaries typically enjoy.

The principal objection to such a scheme would be that such an executable would only be runable on the platform it was compiled for. If one wanted to be able to run the program on a variety of platforms--e.g. Windows, Solaris, MacOS, OpenVMS, etc.--one would have to compile it separately for each of those platforms. Probably, most interest in a compiler would be for the MSWindows and MacOS platforms.

William Thompson

Subject: Re: Compiling IDL ... ever likey ?
Posted by steinhh on Mon, 22 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <4dttc7$cmb@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) writes:
|>
|> James Tappin <sjt> writes:
|>
|> >It stikes me that the biggest technical problem would be handling the way IDL
|> >is able to change the type of variables without the user knowing explicitly
|> >that it is doing so. ...
|>
|> Actually, there's nothing to keep a "compiled" version of an IDL program from
|> acting the same way as in the normal version of IDL. Such a compiled
|> executable would consist of two parts:
|>
|> 1. A program section, which contains a restricted version of the basic IDL
|>   executable. It would be able to do everything IDL can do, except compile
|>   procedures, or accept commands from a command line.
|>
|> 2. A data section, which contains the IDL procedures to be executed, in a
|>   binary interpreted format--presumably the same format that IDL stores the
|>   procedure in a SAVE file.
|>
|> Such an object would act just like an executable--i.e. it would be a single
|> file that one would simply run--but it would preserve all the
|> interpretive-language advantages that IDL currently has. It would still be
|> IDL.
|>

In fact, every interpreted --- uh, most! (self-modifying languages such
as lisp are a bit awkward) -- programming languages can be compiled.
I guess the IDL interpreter goes in a loop that looks something
like

```
   WHILE <program-not-ended> DO BEGIN
       TOKEN = PROGRAM_STORE(PROGRAM_COUNTER)
       PROGRAM_COUNTER = PROGRAM_COUNTER + 1
       CASE TOKEN OF
          <ASSIGNMENT_OPERATION> : DO_ASSIGNMENT(PROGRAM_COUNTER)
          <PRINT_STATEMENT>: DO_PRINT_STATEMENT(PROGRAM_COUNTER)
          :
          :
        END

   END
```

The various functions like DO_ASSIGNMENT or e.g., EVALUATE_EXPRESSION
have branching code to deal with different data types/dimensions etc.

In the same way that this program is possible to compile into somehing
machine executable, the *tokens* themselves could have been compiled

into executable machine code (e.g., mainly just subroutine calls).
This is not very difficult, but there's not much time saved!

|> Some people might be disappointed that the performance would be the same.
|> There would not be the performance increase that compiled binaries typically
|> enjoy.

The key to improving performance is declaring the type and
dimensionality of the data that are to be manipulated. Very often,
IDL subroutines are made to deal with very specific data,
but there's no way to tell IDL about this -- it has to do all
the checks all the time. In the survey about the future of IDL
I suggested the possibility of having "pseudocode blocks", where
all the data to be manipulated are declared in the beginning.
If some of the input data do not match the declaration, a
runtime error occurs.
The statement part of the code would look like e.g., F90, and
it would be quite easy to compile into native machine language.
This is especially suitable for array operations that are not
possible to do without FOR loops -- there are some, consider:

```
B(0) = A(0)
for i=1,N-1 do B(i) = B(i-1) + A(i)
```

or:

```
tmp = 0.0
for i=0,N-1 do tmp = tmp + A(i) * B(i)
```

(In order to get array performance on the last one, one would
have written tmp = total(A*B), but IDL translates this to
temp = A * B
tmp = total(temp) ; -- which is a waste of time and space!)

Stein Vidar
(In the hope that this will some day come true)

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Mon, 22 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

On the subject of compiling IDL, I thought I would just throw a little
gasoline on the fire and see what happens:

Languages that compile to the host machine level are rapidly becoming
obsolete and investing programming time into turning a very efficient
pseudo-compiled, array oriented language like IDL into a true compiled

language is a waste of resources that could be better used to improve
other features of IDL. For example: I haven't heard anyone suggesting
that JAVA be turned into a compiled language.

Computers are simply getting so fast and memory is getting so cheap and
abundant, that a few cycles here and there or a little misuse of memory
doesn't matter for the average application. On the other hand,
programmers are not getting faster or cheaper; therefore, the payoff is
bigger if time is invested in making IDL more programmer friendly.

IMHO, RSI should concentrate on making IDL a terrific application
development platform so that serious applications can be developed for
mass distribution. What IDL really needs is a very cheap run-time
system license for MACs and IBM-clones, an option for compile time type
checking, rapid application development tools, cleaner APIs for plotting
and widgets, ...

If you need to do a calculation that requires looping and will take a
while, and you need to do it over and over, day after day, week after
week, then I agree, you need to use fortran or C. If this is the case,
then it is worth the time that it will take to write a FORTRAN or C
routine to do it and to use CALL_EXTERNAL to interface with IDL.

Sincerely,

Ken Knighton knighton@gav.gat.com knighton@cts.com
Fusion Division
General Atomics
San Diego, CA

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Tue, 23 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

Rick White <rlw@stsci.edu> wrote:
> Ken Knighton wrote:
>
>> Languages that compile to the host machine level are rapidly becoming
>> obsolete and investing programming time into turning a very efficient
>> pseudo-compiled, array oriented language like IDL into a true compiled
>> language is a waste of resources that could be better used to improve
>> other features of IDL. For example: I haven't heard anyone suggesting
>> that JAVA be turned into a compiled language.
>
> You're wrong -- there are definitely plans to produce Java compilers
> that turn Java binaries into native machine code.

No, I said that I haven't heard of it. Obviously, you are more in touch with the JAVA community than I am. But what you are saying simply reinforces the fact that languages of the old paradigm such as Fortran and C, are becoming obsolete. Actually, after thinking about it some more, I should have said:

"Traditional text-based, editor-entered, languages that are compiled directly to the machine level are becoming obsolete."

By the way, this doesn't mean that the billions of lines of COBOL, FORTRAN, C, C++, etc. are going anywhere soon. Heck, some of the computers we use are almost 20 years old. What I am saying is that new language systems that are successful will have to be multi-platform (JAVA, IDL, etc.), provide rapid application development tools (DELPHI, Visual BASIC, etc), or provide a short learning curve and ease of use to new users (IDL, MATLAB, Visual Basic, EXCEL). My inclusion of EXCEL shows that traditional approaches to using computers are losing ground to "End-User Programming".

> are seen by the Java community as crucial to getting good performance
> on computationally intensive applications.

Since you are a Java expert, does Java have built-in:

1. Vector based arithmetic and array manipulation functions
2. Plotting/imaging
3. Image processing and numeric functions
4. Mapping functions

These are mostly done at the machine level by IDL because they are atomic operations in IDL. Because of this, they wouldn't be improved much by compilation. If they are not present in Java, then that explains why compiling Java is seen as crucial for computationally intensive applications. Perhaps what IDL need is more atomic level functions to perform common computationally intensive tasks.

>
> I think the Java approach could serve as a good model for an IDL
> compiler.

I am all for improving efficiency so long as it is not at the expense of removing portability and ease of use.

> Java gets compiled to an intermediate, machine independent
> pseudo-code which is (ordinarily) executed by the Java interpreter.
> The execution penalty of the interpreted code compared with C is
> about a factor of 20. The new compilers will replace the
> interpreter and will translate the Java pseudo-code into native machine

> instructions at execution time. This on-the-fly (or "just-in-time")
> compilation will produce code that executes at nearly the same speed
> as C (I haven't seen any hard numbers.)

And a different compiler will have to be written for every platform.
JAVA has a lot more people and resources behind it than does IDL.

>
> I think this approach could work very well for IDL.

I agree, but whether implementing it is in the best interest of RSI is
another matter. They would have to do a cost-benefit analysis, and I
think that this feature would not pay for itself.

> Perhaps someone should be working on an IDL-to-Java translator or even
> an IDL-to-Java pseudo-code compiler!

If you feel that it would make money, perhaps you should do it yourself.

Hey, just thought I'd liven things up a bit on our usually straight-laced
and boring newsgroup. :)

Ken Knighton   knighton@gav.gat.com knighton@cts.com
General Atomics
San Diego, CA   (Sunny, 70 degrees, wore my shades into work)

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Tue, 23 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

zawodny@arbd0.larc.nasa.gov (Joseph M Zawodny) wrote:
> In article <4e0m1n$k1@rosebud.sdsc.edu> Ken Knighton <knighton@gav.gat.com> writes:
>>
>> On the subject of compiling IDL, I thought I would just throw a little
>> gasoline on the fire and see what happens:
>>
>> Languages that compile to the host machine level are rapidly becoming
>> obsolete ...
>
> Yes, but I cannot write a useful (marketable) program with IDL and sell
> it for 39.95 without making the buyer also spend $1500 to buy IDL first!

But what's in it for RSI? If you are going to make all of the money, perhaps you should hire some compiler writers to reproduce IDL=
 as was done for DBASE. Then you could sell the compiler and your 39.95 programs, and not have to worry about it.

Or, you could just go out and buy DELPHI, Borland C++, and some graphics class libraries and away you go. Or you can buy Visual Bas=
ic and pay a small royalty for run-time licenses and make Bill "Kneel before me" Gates a little bit richer. This is in fact what I =
am advocating for IDL. RSI already sells run-time only licenses, but I don't think they have targeted mass market applications that=
 sell for small amounts of money. As I said in my previous post, it would be nice to be able to buy very cheap run-time licenses fo=
r Macs and PCs.

> I think that with some subset of IDL features, one could eliminate the
> command interpreter stage and go directly to an efficient compiled
> executable. As noted in a previous post, that might require being more
> rigorous with the use of variable types.

I think that anything can be done with enough money and skilled man-power.

>> While I agree with any agruement that gets the cost of IDL down:
> Even if IDL cost $299.95, there would have to be a plethora of desireable
> applications out there to warrant the initial cost of the interpreter.

I suspect that if one could buy the IDL run-time system for 10-20% of the gross sales of a product written in IDL, then it would be =
economically feasible to sell mass market applications written in IDL. Actually, this may be available since RSI already sells run-=
time system licenses.

Ken Knighton   knighton@gav.gat.com knighton@cts.com
General Atomics
San Diego, CA

Subject: Re: Compiling IDL ... ever likey ?
Posted by Rick White on Tue, 23 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

Ken Knighton wrote:

> Languages that compile to the host machine level are rapidly becoming
> obsolete and investing programming time into turning a very efficient
> pseudo-compiled, array oriented language like IDL into a true compiled
> language is a waste of resources that could be better used to improve
> other features of IDL. For example: I haven't heard anyone suggesting
> that JAVA be turned into a compiled language.

You're wrong -- there are definitely plans to produce Java compilers
that turn Java binaries into native machine code. These compilers
are seen by the Java community as crucial to getting good performance

on computationally intensive applications.

I think the Java approach could serve as a good model for an IDL
compiler. Java gets compiled to an intermediate, machine independent
pseudo-code which is (ordinarily) executed by the Java interpreter.
The execution penalty of the interpreted code compared with C is
about a factor of 20. The new compilers will replace the
interpreter and will translate the Java pseudo-code into native machine
instructions at execution time. This on-the-fly (or "just-in-time")
compilation will produce code that executes at nearly the same speed
as C (I haven't seen any hard numbers.)

I think this approach could work very well for IDL. Most IDL code
executes efficiently enough in interpreted mode, especially when you're
doing large array operations where each step takes a considerable
amount of compute time. The slowdown comes if some part of
your calculation can't be done as an array operation but must be
written as an explicit loop over array elements. Then IDL is, like
Java, a factor of 20 (or more) slower than C (so it's not just a matter
of a few cycles here and there.) If such loops could be compiled
on-the-fly then IDL's speed would be comparable to C/Fortran for
essentially all applications.

Perhaps someone should be working on an IDL-to-Java translator or even
an IDL-to-Java pseudo-code compiler!

--
Richard L. White   rlw@stsci.edu   http://sundog.stsci.edu/rick/
Space Telescope Science Institute
Baltimore, MD

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Thu, 25 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

thompson@orpheus.nascom.nasa.gov (William Thompson) wrote:
> steinhh@amon.uio.no (Stein Vidar Hagfors Haugan) writes:
>
>
>> The key to improving performance is declaring the type and
>> dimensionality of the data that are to be manipulated. Very often,
>> IDL subroutines are made to deal with very specific data,
>> ...
>> If some of the input data do not match the declaration, a
>> runtime error occurs.
>
> Yeah, but then it wouldn't be IDL. You might as well write it in FORTRAN at

> that point, IMHO.

I disagree. IDL has tons of functionality built into it that are not present in languages like Fortran or C. IDL is like having Fo=
rtran, a graphics package, a widget toolkit, a numerics package, ... all rolled into one integrated product.

I develop GUI applications in IDL that generally run into thousands of lines of code. It would save me many, many hours of testing =
time if simple type mismatches could be detected at compile time. If there were an option for strong typing, an IDL lint program th=
at would find problems like this, or some other method for preventing simple mistakes that are caught at compile time by most langua=
ge systems, it would be fantastic. Actually, all that would have to happen is for a warning (as opposed to an error) to be generate=
d. I would then have a list of potential program killers that I could investigate.

Ken Knighton   knighton@gav.gat.com   knighton@cts.com
General Atomics
San Diego CA

Subject: Re: Compiling IDL ... ever likey ?
Posted by thompson on Thu, 25 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

steinhh@amon.uio.no (Stein Vidar Hagfors Haugan) writes:

> The key to improving performance is declaring the type and
> dimensionality of the data that are to be manipulated. Very often,
> IDL subroutines are made to deal with very specific data,
> but there's no way to tell IDL about this -- it has to do all
> the checks all the time. In the survey about the future of IDL
> I suggested the possibility of having "pseudocode blocks", where
> all the data to be manipulated are declared in the beginning.
> If some of the input data do not match the declaration, a
> runtime error occurs.

Yeah, but then it wouldn't be IDL. You might as well write it in FORTRAN at
that point, IMHO.

Almost all the IDL code that I write expects to be able to ingest data in a
variety of data types and dimensionality. That's what I like about IDL, and
a good part of why I use it.

People generally ask for IDL compilers for two reasons:

1. To be able to distribute IDL code without having to require other people to
    buy IDL. It was that possibility I was considering in my previous post. I
    think that it is perfectly possible to do this, and still let IDL be IDL.

2. To speed up execution time on tasks that cannot easily be vectorized (or
    which are not efficiently written). I don't see anyway of doing this
    without making fundamental changes in the way IDL works.

Just my $0.02 worth,

Bill Thompson

Subject: Re: Compiling IDL ... ever likey ?
Posted by David Foster on Fri, 26 Jan 1996 08:00:00 GMT

View Forum Message <> Reply to Message

Looking at this from the perspective of RSI, I don't think we can
ever expect to see an IDL compiler capable of creating full-featured
IDL executable prgrams. In terms of economics, this seems pretty
obvious to me...we're asking RSI to spend it's time and money to
make a product that will put itself out of business! There would
never be a reason for a single location to purchase more than one
license.

I think we're talking about two different things:
   1) A compiler
   2) A run-time version of IDL that one could distribute,
 giving IDL functionality to someone that doesn't
 have IDL.

Personally, I don't care about #2, and I don't think the market
for IDL is ever going to be big enough to make it possible, so...

What I would find VERY useful would be a compiler that could compile
individual routines into efficient machine code, and then you
would call these routines using, say, CALL_COMPILED() (just a
suggestion :)). This would give you improved performance for
iterative operations and would still require that you own IDL.
Many users would appreciate having this functionality without
having to learn another programming language. Blah... blah...

David Foster
UCSD Brain Image Analysis Lab
foster@bial1.ucsd.edu

Subject: Re: Compiling IDL ... ever likey ?
Posted by steinhh on Fri, 26 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <4e8p4h$9oe@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) writes:
|> steinhh@amon.uio.no (Stein Vidar Hagfors Haugan) writes:
|> > ... In the survey about the future of IDL
|> >I suggested the possibility of having "pseudocode blocks", where
|> >all the data to be manipulated are declared in the beginning.
|> >If some of the input data do not match the declaration, a
|> >runtime error occurs.
|>
|> Yeah, but then it wouldn't be IDL. You might as well write it in FORTRAN at
|> that point, IMHO.
|>

Hi there Bill,

True, it wouldn't be IDL, and the language could just as well look (something)
like FORTRAN, but it would be *inside* IDL, and you needn't tell anyone to
compile the source code, place the shareable objects anywhere special, or...

|> Almost all the IDL code that I write expects to be able to ingest data in a
|> variety of data types and dimensionality. That's what I like about IDL, and
|> a good part of why I use it.
|>

I agree. But many (maybe most) people write IDL programs for their own use
only, and have no need to supply programs that cope with "everything".
For many computationally intensive applications that people
write, it's only one kind of data that's going through the pipeline. The
sizes of the arrays may change, but seldom the number of dimensions, the
array types etc.

|> 1. To be able to distribute IDL code without having to require other people to
|>   buy IDL. It was that possibility I was considering in my previous post. I
|>   think that it is perfectly possible to do this, and still let IDL be IDL.

And I agree that from this perspective, no compilation is necessary (and it
would probably be very bug-prone as well).

|>
|> 2. To speed up execution time on tasks that cannot easily be vectorized (or
|>   which are not efficiently written). I don't see anyway of doing this
|>   without making fundamental changes in the way IDL works.
|>

I don't want to change the existing functionality of IDL. I just want to be able to write something like:

```
--------------------------------
  ;; Normal IDL
  a = findgen(10)
  b = findgen(10)

  :
  ;; Manipulations of a and b (preserving the type)
  :

  tmp = 0.0;

  compileblock( C : FLTARR(N:INTEGER) = A, $  ; Declarations and "name assosiation"
                D : FLTARR(M:INTEGER) = B, $  ; If A, B or TMP don't fit the bill,
                T : FLOAT = TMP)              ; we want a run-time error.
     i : INTEGER
  begin
     ;; Here goes the compiling statements

     IF M NE N THEN ASSERTION_FAILED("C and D unequal size")
     FOR i = 0,N-1 DO T = T + C(i)*D(i)

  endb

  ;; Normal IDL again
  PRINT,TEMP  ;; Has the value of TOTAL(A*B)
              ;; But *without* calculating temp = A*B
              ;; and then taking TOTAL(temp)

-------------------------------
```

The key here is that even for operations that are possible to vectorize, IDL wastes a lot of time because it's an interpreting language: Ok, multiply A and B. Let's see: A is a float array, 10 elements, and B is a float array, 10 elements, so the result will be float, 10 elements: allocate space for that. Do the multiplication and store the result element by element. Done. Now, take the total of the temporary. Let's see, what was it again, oh yes, it's a 10 element float, so i'll use the code for adding up contiguous floats, 10 pieces of it.

All this "figuring out", plus actually storing the temporary takes time and space.

Doing an atomic array function in IDL is *extremely* optimized, though: I tried to beat IDL's array operations once, when I was doing some Fourier filtering of real (as opposed to complex) data. I used every trick in the book (the book being Num. Recipes), but I didn't gain anything (it might have been a few per cent).

On the other hand, when wanted to sum up some square differences
between two arrays with a windowing function, i.e., taking

chisq = sum-over-i [(A(i)-B(i))^2*1.0/(1.0+i^2/const)]

then the call_external code (in C) beat the hell out of IDL by
a substantial factor.

If the "pseudocode" language is kept very simple, then
it shouldn't be difficult to compile such operations into
quite efficient code (although not as optimized as IDL's own).

In my opinion, the fact that call_external exists indicates
that there is a need for *both* high-level IDL *and* in some
cases a low-level compiling language, so why not lower the
threshold a little?

Of course cost is a concern, but if PV-WAVE gets this pseudocode
and IDL doesn't, I know which one I'd choose (if starting from
scratch, at least) for serious work.

The other thing that could change my mind is supplying a proper
handle/pointer syntax. It's such a waste of screen space writing

```
HANDLE_VALUE,ID,A,/NO_COPY
PRINT,A.MESSAGE(5)
HANDLE,ID,A,/SET,/NO_COPY
```

instead of simply

```
PRINT,A^.MESSAGE(5)
```

I don't need pointer arithmetic or anything, nor do I need
to be able to point to elements inside an array, I just want
a compression of statements like the two extra lines above
down to *one* character. Please!

Stein Vidar

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Sat, 27 Jan 1996 08:00:00 GMT

View Forum Message <> Reply to Message

David Foster <foster@bial1.ucsd.edu> wrote:
>

> never be a reason for a single location to purchase more than one
> license.

I think if the compiler and interactive IDL were different products,
then most locations would want the interactive IDL and some would want
both. I wouldn't want to have an IDL compiler without the interactive
system.

Regards,

Ken Knighton   knighton@gav.gat.com knighton@cts.com
General Atomics
San Diego, CA

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ken Knighton on Sat, 27 Jan 1996 08:00:00 GMT
View Forum Message <> Reply to Message

steinhh@amon.uio.no (Stein Vidar Hagfors Haugan) wrote:
> In my opinion, the fact that call_external exists indicates
> that there is a need for *both* high-level IDL *and* in some
> cases a low-level compiling language, so why not lower the
> threshold a little?

IMHO, I believe that a better CALL_EXTERNAL would go a long way towards
accomplishing this. Why is it that args are passed to a C routine as
argc, argv instead of passing the actual parameters: arg1, arg2, arg3...
This means that wrappers have to be written to call most non-unix
routines. Why is it that there is no good way to call legacy code
without having to resort to this? Also, why is there no good way to
shut down IDL's use of interrupts (such as unix signals) while the call
is taking place?

CALL_EXTERNAL(..., /BLOCK _INTERRUPTS)

In order to do i/o from an external routine, one has to worry about
this stuff when it shouldn't be necessary.

> The other thing that could change my mind is supplying a proper
> handle/pointer syntax. It's such a waste of screen space writing
>
> HANDLE_VALUE,ID,A,/NO_COPY
> PRINT,A.MESSAGE(5)
> HANDLE,ID,A,/SET,/NO_COPY
>
> instead of simply
>

> PRINT,A^.MESSAGE(5)

One wonders. The same problem exists with the wordy WIDGET routines.
Even if some functional notation such as:

PRINT, (H_VALUE(A))(5)

or

PRINT, W_UVALUE(wId)   ;For widgets

were available, it would be nicer. Of course these functions could
be written in IDL, but would be inefficient for some uses.

Regards,

Ken Knighton   knighton@gav.gat.com   knighton@cts.com
General Atomics
San Diego, CA

Subject: Re: Compiling IDL ... ever likey ?
Posted by David Ritscher on Fri, 02 Feb 1996 08:00:00 GMT

View Forum Message <> Reply to Message

A few thoughts about the (ever-recurring) IDL compiler discussion:

1. One exists for internal use at RSI. When one makes a big
application for redistribution they can create an executable version of
of your application for you. This is quite expensive. The Visible
Human CD is a demonstration of this approach.

2. MATLAB has recently released a compiler.

3. AVS (Advanced Visual Systems) seems to be making some good progress
in this direction. You can build any of your own routines (in C, C++,
etc.) into their system. Here in Germany they have a run-time
licensing which is very cheap (something like $30) within any of the
universities, and still pretty cheap (about $100) in industry. It's
much more expensive when one needs the network editor available. They
are headed in the object-oriented direction, at least in a certain
sense of the word, which I also find appealing. They seem to do the
graphics stuff quite well, but I still prefer IDL/PVWave for pure
mathematical work.

4. There are run-time versions of IDL and PV-Wave available.
(with run-time licensing). Something that both lack is the capability
to also protect (i.e., license) your application.

5. I often hear comments that 'it would be impossible to make an IDL
compiler, for reason X.' However, I think it's relatively clear that
the full functionality of IDL or PVWave can be packed into a library,
callable from C, etc. There would be, for example, a function
'idl_plot();' or 'pvwave_plot();' that would function exactly like the
current plot function. The question is how much of the interpreter to
bring along for this purpose, vs. how much do you have to handle all
the data manipulation yourself. It would be important to bring along
the IDL/PVWave memory allocation system, so that the normal
manipulation of data is possible (extract of parts of arrays, the
WHERE function, etc.).

--

David Ritscher

Raum 47.2.279
Zentralinstitut fuer Biomedizinische Technik
Albert-Einstein-Allee 47
Universitaet Ulm
D-89069 ULM
Germany

Tel: ++49 (731) 502 5313
Fax: ++49 (731) 502 5343
                 or   5317
internet: david.ritscher@zibmt.uni-ulm.de

Subject: Re: Compiling IDL ... ever likey ?
Posted by Andy Loughe on Mon, 05 Feb 1996 08:00:00 GMT
View Forum Message <> Reply to Message

David Ritscher wrote:

> 2. MATLAB has recently released a compiler.

Bingo!

--
Andrew F. Loughe (afl@cdc.noaa.gov)
University of Colorado, CIRES * Campus Box 449 * Boulder, CO 80309
phone: (303) 492-0707   fax: (303) 497-7013

Subject: Re: Compiling IDL ... ever likey ?
Posted by Ferdinand Jamitzky on Mon, 12 Feb 1996 08:00:00 GMT
View Forum Message <> Reply to Message

To which language ?
To C++ ?
On what system ?
Where can I get some information?

F. Jamitzky