Subject: Re: Am I stupid?
Posted by Martin Schultz on Tue, 24 Jul 2001 09:25:15 GMT
View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:

> Bill wrote:
>>
>> A better way to describe what could, but is not, implemented for IDL's keywords is a multise
>> stage process
>>
>> 1. If a keyword on an invocation exactly matches a keyword in the functions's definition then it
>> is that keyword, else
>>
>> 2. if the keyord on invocation is an abbreviation for exactly one keyword in the functions's
>> definitition then it is that keyword, else
>>
>> 3. It is an error that can be determined statically.
>
> I understand your point (and Jaco's and James'.... and Craig's too I think) but, to me,
> the above rules are defined with only the programmer's (i.e. the person that wrote the
> code that has some potential for ambiguousness (?) in the keywords) viewpoint.
>
> My main, err, discomfort with allowing "The keyword is too short to be unique, therefore
> it is unabbreviated" type of behaviour is that it does not take into account the person
> who is using this code cold and is not an IDL whiz. I think that with a little bit of
> forethought, these issues can be eliminated by the code writer to save the puir wee
> unsuspecting future IDL user from some code that was written with potential ambiguous
> keyword problems. To paraphrase Reverend Lovejoy's wife: "will somebody *please* think of
> the users!" :o)
>
> Phew.
>
> O.k., no more poking pointy sticks at windmills for me. :o)
>
> paulv
>

Oh, how often have I asked that question in the subject to myself!

Maybe a little late to warm up this thread again, but with a topic that is
as near to my heart as keyword abbreviation, I just can't resist. What I would
like to propose is thinking a little out of the box and pick up the trends
that are out there in the software world. Why bother with ambiguities at all:
let the program decide what it wants to do with the keyword. Modern software
should just add three buttons to every program, one of which would have to be
activated after each computation: "Yes", "No", "Don't know". Hence, the user
"indicates" to the software if he or she is content with the result. If

so, the software will be happy and celebrate by taking a day off, if not,
the software will file this reply in its database and perhaps attempt to
learn from this experience in a future session (or version). If the user
continuously hits the "Don't know" button, he or she will automatically
be registered for a programming course.

Now, internally, this could probably be realized with an algorithm that
was recently developed by some smart mathematician in Hungary (or was it
Bulgaria ?): If a keyword has more than !PI letters, and is not one of the
words "TO", "I", "N", "GR" (this only for heritage reasons), "D", "T",
then the likelihood for a keyword that can be abbreviated is proportional
to the fifth power of the number of letters-!PI+1. Exceptions are: the
specific keywords "TIME", "TIMEUNIT", and "TIMESTEP" which are treated
seperately due to a user request. And because IDL shall also be usable
by minors, certain words containing explicit language or violence are
excluded from the vocabulary.

My final suggestion is a little more serious: Why not put everything
related to TIME in a structure?

```
   Do_Something, TIME={value:0., unit:'years', step:1L }
```

has the virtue of unambiguity. If you use my little ChkStru program
(available on http://www.mpimet.mpg.de/~schultz.martin/idl/)
you can easily analyse this structure argument as in:

```
   IF ChkStru(time, 'value') then timevalue = time.value
   IF ChkStru(time, 'unit') then timeunit = time.unit
   IF ChkStru(time, 'step') then timestep = time.step
```

Altogether, I think there is clearly a need for RSI to properly define
the line between IDL for programmers and IDL for scientists, and to
try to make this distinction clear in future versions. A few key
differences between these groups:

programmers:
*like variable declarations and hate type changes that are not asked for
*like fixed variable dimensions and hate secretly deleted trailing dimensions
*like as much syntax checking as possible at compile time

scientists:
*hate variable declarations and want the software to decide on the type
*hate limitations imposed by fixed variable dimensions
*hate error messages from the compiler and believe the program should
 know what they intend

Go figure!

Cheers,

Martin

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[