Subject: IDL v5.1 impressions
Posted by J.D. Smith on Wed, 13 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

I'll go ahead and say it: I like IDL v5.1. Call_Method is a valuable addition (and I like that it's both a procedure and a function ... kind of makes you think call_function and call_procedure could have been implemented this way huh?). "?:" is a great new operator that I had given up hope on years ago. Haven't tried the profiler or conditional branches, but they would seem bring IDL closer to the level of, e.g., C for debuggability (although it's already light-years ahead in terms of average bugs per 1000 lines produced). Many of the serious limitations I have griped about have apparently been heard, the most important of which is the need for pass-by-reference keywords. I got a direct response from RSI that they would look into this, and less than one-tenth version later, here it is...

But having said that I have to bring down the joy level and remark that, although I applaud RSI for implementing pass by reference inherited keywords, I don't understand why they did it in such a complicated and non-intuitive way! Having two keyword inheritence mechanisms seems unnecessary. The manual quotes:

****
By contrast, the "pass by value" (_EXTRA) keyword inheritance mechanism is useful in the following situations:

1. Your routine needs access to the values of the extra keywords for some reason.

2. You want to ensure that variables specified as keyword parameters are not changed by a called routine.
****

Number 1 is silly. Of course I might need the values along the way! When I give a positional argument to a procedure, like:

myfunc, array

I both expect to have access to the value of array, and to have any changes I make to array be available to the calling environment. That is the whole beauty of pass by reference. Passing by reference *doesn't* mean passing without value! An example of when you might need the values in the set of extra parameters if for general pupose error checking; suppose your procedure would only allow keywords with floats, no matter what keyword it is, for passing on up to some other procedure. We don't want to define all possible keywords, we just want

to ensure that for any passed containing a value or a variable defined with a value, that value is a float.

And as for Number 2... why don't we have different mechanisms for positional argument passing, then? If I pass an array to myfunc, I have no way of ensuring that array isn't modified, but I don't care, since I know what myfunc does and act accordingly.

I have been happy with the choice of reference vs. value passing for the positional arguments. We are all used to the rules. Why didn't RSI simply extend these same rules to include keywords? I think the answer might be that this was a much harder problem to code...

And one more problem. The ever useful fucntion arg_present() chokes when confronted with _ref_extra keywords.... let's make an example, using the by reference mechanism that RSI has provided us:

```
pro testit, VAR=v
  help,v
  print,'Arg present: ',arg_present(v)
end

pro testrefext, _REF_EXTRA=re
  help,re
  testit,_EXTRA=re
end
```

first, lets try:

```
IDL> a=6
IDL>  testrefext, var=a
RE              STRING    = Array[1]
V               INT       =        6
Arg present:        1
IDL> help,b
B               UNDEFINED = <Undefined>
IDL> testrefext, var=b
RE              STRING    = Array[1]
V               UNDEFINED = <Undefined>
Arg present:        1
```

which is good so far... both a and b are valid arguments for sticking things in for return to the main level. A already has a value, b does not. But look at:

```
IDL> testrefext, var=10
RE              STRING    = Array[1]
V               INT       =       10
```

Arg present: 1

10 is a valid variable for passing back? I don't think so.
Arg_present() is therefore entirely useless for keyword parameters. It
simply returns 1 or 0 depending on whether we used _REF_EXTRA or _EXTRA,
but we already knew that when we wrote it! We can never know if the
keyword variable passed by reference is just a constant, or really an
IDL variable suitable for sticking things into.

Anyway, this new mechanism will certainly have many uses, and RSI does
get an A for effort. I can see where they were going with this one, and
it even solves the specific problem which I originally used to
illustrate the need for such a mechanism (except for the arg_present()
issue), but I think it could have been implemented much more cleanly,
without extra confusion (no pun intended).

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 |*|