Subject: Re: Ellipsis in IDL?
Posted by David Fanning on Wed, 21 Jul 2004 23:10:10 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> How do you define a procedure to take N number of arguments when you
> don't know what N is before the procedure call? For those of you who
> have worked with C, what I'm after is something similar to the ellipsis
> (...) which allows N many arguments to be specified for functions such
> as printf.
>
> In IDL, the print command is obvious example of what I'm trying to do.
> The signature of print is:
>
> print [, Expr1, Expr2, ... , ExprN]
>
> So, how can I write a procedure to take N many arguments?

I don't think this is possible, not even if you
write it in C and try to link it to IDL. A
fundamental property of a procedure or function
in IDL (I think) is that there is a defined number
of arguments.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Ellipsis in IDL?
Posted by Michael Wallace on Wed, 21 Jul 2004 23:16:19 GMT
View Forum Message <> Reply to Message

> I don't think this is possible, not even if you
> write it in C and try to link it to IDL. A
> fundamental property of a procedure or function
> in IDL (I think) is that there is a defined number
> of arguments.

If this is the case, how does print work? It *appears* to be a
procedure. If it's not a procedure, then what is it?

-Mike

Subject: Re: Ellipsis in IDL?
Posted by David Fanning on Wed, 21 Jul 2004 23:17:09 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> If this is the case, how does print work? It *appears* to be a
> procedure. If it's not a procedure, then what is it?

A C program, just like the rest of IDL. :-)

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Ellipsis in IDL?
Posted by Mark Hadfield on Thu, 22 Jul 2004 00:00:50 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
> How do you define a procedure to take N number of arguments when you
> don't know what N is before the procedure call? For those of you who
> have worked with C, what I'm after is something similar to the ellipsis
> (...) which allows N many arguments to be specified for functions such
> as printf.
>
> In IDL, the print command is obvious example of what I'm trying to do.
> The signature of print is:
>
> print [, Expr1, Expr2, ... , ExprN]
>
> So, how can I write a procedure to take N many arguments?

You can't do exactly that, but you can write a routine to accept a large number of arguments (better called positional parameters) and then use the various inquiry functions (like N_PARAMS, SIZE, N_ELEMENTS & ARG_PRESENT) in your code to handle the different cases.

For example, one of my general-purpose routines (MGH_NEW, a procedure wrapper for OBJ_NEW) looks like this (simplified):

```
pro MGH_NEW, name, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, $
          RESULT=result

   if size(name, /TYPE) ne 7 then $
      message, 'The first parameter must be a class name.'

   case n_params() of
     1:  result = obj_new(Name)
     2:  result = obj_new(Name, P1)
     3:  result = obj_new(Name, P1, P2)
     4:  result = obj_new(Name, P1, P2, P3)
     5:  result = obj_new(Name, P1, P2, P3, P4)
     6:  result = obj_new(Name, P1, P2, P3, P4, P5)
     7:  result = obj_new(Name, P1, P2, P3, P4, P5, P6)
     8:  result = obj_new(Name, P1, P2, P3, P4, P5, P6, P7)
     9:  result = obj_new(Name, P1, P2, P3, P4, P5, P6, P7, P8)
     10:  result = obj_new(Name, P1, P2, P3, P4, P5, P6, P7, $
                           P8, P9)
     11:  result = obj_new(Name, P1, P2, P3, P4, P5, P6, P7, $
                           P8, P9, P10)
     else:  message,  'Too many parameters'
   endcase

end
```

It used to take 15 positional parameters, but I trimmed it down to 10, which is still more than enough. I don't know the maximum number supported by IDL, but it's way more than you would want to use in normal circumstances.

Inquiring about the number and status of your parameters is a subtle business. I'm sure David has some useful tutorials.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Ellipsis in IDL?
Posted by Michael Wallace on Thu, 22 Jul 2004 04:37:40 GMT
View Forum Message <> Reply to Message

> You can't do exactly that, but you can write a routine to accept a large
> number of arguments (better called positional parameters) and then use
> the various inquiry functions (like N_PARAMS, SIZE, N_ELEMENTS &

> ARG_PRESENT) in your code to handle the different cases.

Ah, I see. I think I can work with this. I don't expect to ever have
more than eight or so argu-- positional parameters. I have used SIZE
and N_ELEMENTS in the past, but I had never run across N_PARAMS until
now. That greatly simplifies things. It's a good thing you mentioned
it or I probably would have written my own. ;-) I say that because one
time I got part of the way through writing my own KEYWORD_SET type of
function before realizing that IDL already had it's own handy
KEYWORD_SET function.

Subject: Re: Ellipsis in IDL?
Posted by Mark Hadfield on Thu, 22 Jul 2004 04:58:08 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
>> You can't do exactly that, but you can write a routine to accept a
>> large number of arguments (better called positional parameters) and
>> then use the various inquiry functions (like N_PARAMS, SIZE,
>> N_ELEMENTS & ARG_PRESENT) in your code to handle the different cases.
>
>
> Ah, I see. I think I can work with this. I don't expect to ever have
> more than eight or so argu-- positional parameters. I have used SIZE
> and N_ELEMENTS in the past, but I had never run across N_PARAMS until
> now. That greatly simplifies things. It's a good thing you mentioned
> it or I probably would have written my own. ;-) I say that because one
> time I got part of the way through writing my own KEYWORD_SET type of
> function before realizing that IDL already had it's own handy
> KEYWORD_SET function.

Actually, I don't expect you'll want to use N_PARAMS all that much. All
it tells you is how many positional parameters there are. So it's useful
when you want to pass these parameters on to another routine, without
knowing *anything* about them, as in the example I included. More
usually, you'll want to know if each parameter is currently associated
with data; you can use N_ELEMENTS for that, eg:

```
function add, p0, p1, p2, p3
   if n_elements(p0) eq 0 then message, 'Nothing to add'
   result = p0
   if n_elements(p1) gt 0 then result = result + p1
   if n_elements(p2) gt 0 then result = result + p2
   if n_elements(p3) gt 0 then result = result + p3
   return, result
end
```

Sometimes you'll want to know if changes you make to parameter data will
be retained when you exit. If not you may be able to save time by not
computing it. You can use ARG_PRESENT for that, eg

```
pro set, p0, p1, p2, p3
   if arg_present(p0) then p0 = <expensive operation>
   if arg_present(p1) then p1 = <expensive operation>
   if arg_present(p2) then p2 = <expensive operation>
   if arg_present(p2) then p3 = <expensive operation>
end
```

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: Ellipsis in IDL?
Posted by Jeff Guerber on Thu, 22 Jul 2004 21:46:34 GMT
View Forum Message <> Reply to Message

On Wed, 21 Jul 2004, Michael Wallace wrote:

> How do you define a procedure to take N number of arguments when you
> don't know what N is before the procedure call? For those of you who
> have worked with C, what I'm after is something similar to the ellipsis
> (...) which allows N many arguments to be specified for functions such
> as printf.
>
> In IDL, the print command is obvious example of what I'm trying to do.
> The signature of print is:
>
> print [, Expr1, Expr2, ... , ExprN]
>
> So, how can I write a procedure to take N many arguments?

I don't know how print works internally, but remember that you can have
fewer actual arguments (in the call) than dummy arguments (in the
procedure definition). It's a bit cumbersome, but one thing you can do is
define your procedure with more arguments than you expect to need:

```
pro ManyArgs, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, $
   arg11, arg12, arg13, arg14, arg15, arg16, arg17, arg18, arg19, arg20, $
   arg21, arg22, arg23, arg24, arg25, arg26, arg27, arg28, arg29, arg30
```

Then as you process each one, check whether the caller used it with
statements like:

```
if (n_elements(arg23) NE 0) then begin
    ...process arg23...
endif
```

If you want to loop over all the arguments, one obvious possibility would be to use a case statement inside the loop, to check n_elements of the appropriate argument. (I said it was a bit cumbersome!)

I'll bet even print has some limit on the number of arguments it can take. Hope this helps,

Jeff Guerber
