
Subject: Re: [Object IDL] self-documenting objects ...

Posted by [steinhh](#) on Fri, 31 Jul 1998 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Darran wrote:

- > Rather than having to duplicate documentation in this method
- > *and* the code header, can I not simply display the source
- > header to the user? This approach has the benefit of encouraging
- > the IDL programmer to write well-documented code headers.
- > Comments on this convention?

It's a very good convention!

- > Question: how can I locate the source file assuming that
- > I know the file name (myobject__define.pro) and that it
- > is located in the IDL_PATH? (IDL's filepath procedure
- > gives incorrect info for user written routines.)

Hmm. You needn't "locate" the file, just ask IDL where it found it: The `HELP,CALLS=CALLS` keyword will give you the information for free. It can, however, be a little bit awkward to extract info from the returned string array, so I've produced a function called `GET_CALLDETAILS()` which returns a structure with the info. To get info inside a routine about who/what/where called you:

```
info = get_calldetails() ;; An implicit depth = 2 argument here.
```

The routine is appended below (with a code header :-)

The reason I wrote this routine was that I've seen the need to have something like "inline text constants" introduced to IDL. In most shells, you can do e.g.,

```
unix> myprog <<STOP
This text will be put into the stdin of myprog.
This one, too...
STOP
```

I never got around to doing this, however, until this morning when you asked this question I thought I should give it a try. Lo and behold, a new function has been born: `INLINE_TEXT()`
Use:

```
txt = inline_text(';STOP')
;This text will be returned by the INLINE_TEXT function.
;This one, too.
```

```
;STOP
```

Those lines (in a program that's saved in a file - not compiled directly with the .run command) correspond to:

```
txt = [';This text will be returned by the INLINE_TEXT function.', $
      ';This one, too.', $
      ';STOP' ]
```

In my eyes, the new solution is a *lot* better, from an esthetic point of view. The default MARKER argument to `INLINE_TEXT()` is `';-'`, which is the standard "end-of-doc" marker for library routines.

The `INLINE_TEXT()` function follows, but with a slight twist: An extra function called `INLINE_TEXT_HELP()` is put at the head of the file, so the documentation header for `INLINE_TEXT` is available at the beginning of the file as well as through the function `INLINE_TEXT_HELP()`: Try

```
PRINT, INLINE_TEXT_HELP(), FORMAT='(A)'
```

Ah - a nice start on my day, at least...

(Note that the "forward_function inline_text" statement in `inline_text_help` is necessary only for *this* particular "help" function)

Stein Vidar

```
inline_text.pro-----
```

```
FUNCTION inline_text_help
  forward_function inline_text & return, inline_text(';-')
;+
; Project   : SOHO - CDS
;
; Name      : INLINE_TEXT()
;
; Purpose   : Return inline text immediately following call line
;
; Explanation : Returns verbatim text immediately following the line calling
;               INLINE_TEXT, up to and including the line that contains the
;               end MARKER.
;
;               Uses GET_CALLDETAILS to find out the file name and the line
;               number of the calling line.
;
```

```

;      The default MARKER is ';' (the standard "end of
;      documentation" marker in the library routines).
;
; Use      : TEXT = INLINE_TEXT( [MARKER] )
;
; Inputs   : None required
;
; Opt. Inputs : MARKER : The text appearing at the *beginning* of the last
;                  line of inline text.
;
; Outputs   : Returns the inline text, or message about failure to read.
;
; Opt. Outputs: None.
;
; Keywords  : None.
;
; Calls     : GET_CALLDETAILS()
;
; Common    : None.
;
; Restrictions: Needs to find the program source file!
;
; Side effects: None.
;
; Category  : General.
;
; Prev. Hist. : None.
;
; Written    : S.V.H.Haugan, UiO, 31 July 1998
;
; Modified   : Not yet.
;
; Version    : 1, 31 July 1998
;-
END

```

```

FUNCTION inline_text,marker

```

```

;; Default maker ';'

```

```

IF n_params() EQ 0 THEN marker = ';'

```

```

;; Get caller details

```

```

info = get_calldetails()

```

```

;; Return message instead of inline text in case of trouble

```

ON_IOERROR,abort

openr,lun,info.file,/get_lun

tx = strarr(info.lineno)
readf,lun,tx

start = fstat(lun)

nlines = 0

tx = "

REPEAT BEGIN

 readf,lun,tx

 nlines = nlines+1

END UNTIL strpos(tx,marker) EQ 0

point_lun,lun,start.cur_ptr

text = strarr(nlines)
readf,lun,text

close,lun

free_lun,lun

return,text

abort:

IF n_elements(lun) EQ 1 THEN BEGIN

 close,lun

 free_lun,lun

END

return,['Error in reading inline text from file :'+info.file]

END

get_calldetails.pro-----

;+

```

; Project   : SOHO - CDS
;
; Name      : GET_CALLDETAILS()
;
; Purpose   : Return details of calling program (at any stack depth)
;
; Explanation : The HELP,CALLS=CALLS utility is nice, but it's often a bit
;               awkward to extract information. This routine returns the
;               information in a more edible form, as a structure:
;
;               {GET_CALLDETAILS_STC,
;               TEXT  : The full text of CALLS(DEPTH)
;               MODULE : The procedure or function name of CALLS(DEPTH)
;               FILE   : The source file for the procedure
;               LINENO : The line number of the calling statement
;               DEPTH  : The depth used (default 2)
;               TOTALDEPTH : The maximum possible depth allowed in this call}
;
;               Depth=0 means *this* program (GET_CALLDETAILS)
;               1 means the caller of GET_CALLDETAILS
;               2 means the caller of the program calling
;               GET_CALLDETAILS (DEFAULT)
;               3 means the caller of the caller of the program calling
;               GET_CALLDETAILS ... etc
;
; Use       : STC = GET_CALLDETAILS( [ DEPTH ] )
;
; Inputs    : None required
;
; Opt. Inputs : DEPTH : See Explanation
;
; Outputs   : Returns information structure.
;
; Opt. Outputs: None.
;
; Keywords  : None.
;
; Calls     : None.
;
; Common    : None.
;
; Restrictions: None.
;
; Side effects: None.
;
; Category  : General.
;
; Prev. Hist. : None.

```

```

;
; Written   : S.V.H.Haugan, UiO, 6 May 1998
;
; Modified  : Not yet.
;
; Version   : 1, 6 May 1998
;-

```

```

FUNCTION get_calldetails,depth

```

```

    IF n_elements(depth) EQ 0 THEN depth = 2

```

```

    help,calls=calls

```

```

    IF depth GE n_elements(calls) THEN message,"Depth exceeds call stack"

```

```

    IF depth LT 0 THEN message,"Do whatever you want! I can't foretell it"

```

```

    line = calls(depth)

```

```

    IF strpos(line,'$MAIN$') EQ 0 THEN $
        line = '$MAIN$ <$MAIN$( 0)>'

```

```

    blank = strpos(line,' ')
    langle = strpos(line,'<')
    rangle = strpos(line,'>')
    lparen = strpos(line,'(')
    rparen = strpos(line,')')

```

```

    module = strmid(line,0,blank)
    file = strmid(line,langle+1,lparen-(langle+1))
    lineno = long(strmid(line,lparen+1,rparen-(lparen+1)))

```

```

    stc = {get_calldetails_stc,$
        text:line,$
        module:module,$
        file:file,$
        lineno:lineno,$
        depth:depth,$
        totaldepth:n_elements(calls)-1}
    return,stc

```

```

END

```

```

-----

```