Subject: Re: IDL/v5 to /v4 converter
Posted by thompson on Wed, 22 Jul 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Earlier, I wrote

> One of the changes introduced in IDL/v5 was to allow the use of square []
> brackets when referencing parts of arrays, instead of the older round ()
> brackets. This change was introduced to unambiguously differentiate between
> arrays and function calls. Many routines now developed for IDL/v5 could also
> be used in IDL/v4 if the square brackets were converted into round brackets.
> Before I write such a beast, I thought I'd ask if anyone already has one.

Wayne Landsman was kind enough to point me to a routine he wrote, which I did have some success with. However, for various reasons, I ended up writing my own version anyway. I'd be interested to know if anyone can come up with any situations which it doesn't handle properly. If anyone has any suggestions for improvement, I'd also be interested.

Thank you,

William Thompson

```
 ================================================================
==================
 PRO IDL5TO4, FILENAME, OUTDIR, ECHO=ECHO, ERRMSG=ERRMSG
;+
; Project     : SOHO - CDS
;
; Name        : IDL5TO4
;
; Purpose     : Convert IDL/v5 array[] syntax to IDL/v4 array() syntax.
;
; Category    : Utility
;
; Explanation : In IDL version 5, square brackets were introduced as an
;  alternate way of referencing an array, to avoid confusion
;  between arrays and functions.  In prior versions of IDL, the
;  syntax FRED(3) could be interpreted as either the third element
;  of the array FRED, or as a call to the function FRED.  This
;  syntax is still supported in IDL/v5, but the new syntax FRED[3]
;  can be used to show that it is the third element of the array
;  FRED which is desired, and not a function call.
;
;  One of the problems that the new syntax resolves is that of
;  routines where FRED(3) was intended to be used as an array
;  subscripting, and was developed in an environment where there
;  was no FRED function which could cause confusion.  If this
```

```
;  routine was then moved to an environment where there was a FRED
;  function, then the routine would no longer work correctly.
;  Using FRED[3] solves this problem.
;
;  Unfortunately, the syntax FRED[3] is not supported in earlier
;  versions of IDL.  However, it may be that many routines would
;  actually work in the earlier IDLs if the FRED[3] syntax was
;  changed to FRED(3).  The routine IDL5TO4 makes this conversion.
;  The modified routine should then work in earlier versions of
;  IDL, assuming that there are no other version-specific aspects
;  to the code.
;
; Syntax      : IDL5TO4, FILENAME  [, OUTDIR ]
;
; Examples    : IDL5TO4, '*.pro'  ;Converts all procedure files
;      ;in current directory
;
;  IDL5TO4, '*.pro', 'idlv4' ;Converted files written to
;      ;idlv4 subdirectory.
;
; Inputs      : FILENAME = The name of the file(s) to process.  May be an array
;      of filenames, and may also contain wildcard
;      characters.
;
; Opt. Inputs : OUTDIR = The name of a directory to write the converted files
;    to.  The directory must aleady exist--the routine will
;    not try to create it.  If not passed, then the files
;    will be replaced with the converted version at their
;    present location.
;
; Outputs     : None.
;
; Opt. Outputs: None.
;
; Keywords    : ECHO   = If set, the a message line is printed for every file
;    processed.
;
;  ERRMSG = If defined and passed, then any error messages will be
;    returned to the user in this parameter rather than
;    depending on the MESSAGE routine in IDL.  If no errors
;    are encountered, then a null string is returned.  In
;    order to use this feature, ERRMSG must be defined
;    first, e.g.
;
;    ERRMSG = ''
;    IDL5TO4, ERRMSG=ERRMSG, ...
;    IF ERRMSG NE '' THEN ...
;
```

```
; Calls       : DATATYPE, FIND_FILE, FILE_EXIST, BREAK_FILE, CONCAT_DIR
;
; Common      : None.
;
; Restrictions: Although most situations should be accounted for, there may
;  still be situations which are not adequately addressed by this
;  routine.  One possible failure scenario is when a structure
;  contains a tag name which exactly matches an IDL operator,
;  e.g. "AND", "OR", "MOD", "EQ", etc.
;
;  This routine may not work correctly in Windows or MacOS, if it
;  needs to overwrite a file which already exists.  However, it
;  should work okay if the output files are directed to an empty
;  subdirectory.
;
; Side effects: None.
;
; Prev. Hist. : Partially based on IDLV5_TO_V4 by Wayne Landsman
;
; History     : Version 1, 22-Jul-1998, William Thompson, GSFC
;
; Contact     : WTHOMPSON
;-
;
;
 ON_ERROR, 2
;
;  Check the number of parameters.  Make sure that FILENAME and OUTDIR are
;  strings, and that OUTDIR is a scalar.
;
 IF N_PARAMS() LT 1 THEN BEGIN
    MESSAGE = 'Syntax:  IDL5TO4, FILENAME  [, OUTDIR ]'
    GOTO, HANDLE_ERROR
 ENDIF
;
 IF DATATYPE(FILENAME,1) NE 'String' THEN BEGIN
    MESSAGE = 'FILENAME must be a string'
    GOTO, HANDLE_ERROR
 ENDIF
;
 IF N_ELEMENTS(OUTDIR) GE 1 THEN BEGIN
    IF DATATYPE(OUTDIR,1) NE 'String' THEN BEGIN
  MESSAGE = 'OUTDIR must be a string'
  GOTO, HANDLE_ERROR
    ENDIF
;
    IF N_ELEMENTS(OUTDIR) GT 1 THEN BEGIN
  MESSAGE = 'OUTDIR must be a scalar'
```

```
  GOTO, HANDLE_ERROR
    ENDIF
 ENDIF
;
;  If FILENAME is an array, then call this routine for each element.
;
 IF N_ELEMENTS(FILENAME) GT 1 THEN BEGIN
    FOR I=0,N_ELEMENTS(FILENAME)-1 DO BEGIN
  MESSAGE = ''
  IDL5TO4, FILENAME(I), OUTDIR, ECHO=ECHO, ERRMSG=MESSAGE
  IF MESSAGE NE '' THEN GOTO, HANDLE_ERROR
    ENDFOR
    RETURN
 ENDIF
;
;  If FILENAME contains a wildcard character, then expand the wildcards, and
;  call this routine for each filename derived.
;
 FILES = FIND_FILE(FILENAME)
 IF N_ELEMENTS(FILES) GT 1 THEN BEGIN
    FOR I=0,N_ELEMENTS(FILES)-1 DO BEGIN
  MESSAGE = ''
  IDL5TO4, FILES(I), OUTDIR, ECHO=ECHO, ERRMSG=MESSAGE
  IF MESSAGE NE '' THEN GOTO, HANDLE_ERROR
    ENDFOR
    RETURN
 ENDIF
;
;  Determine the byte equivalences of various characters to be used in the
;  program.
;
 TAB    = 9B
 BLANK  = (BYTE(' '))(0)
 SQUOTE = (BYTE(''''))(0)
 DQUOTE = (BYTE('"'))(0)
 ZERO   = (BYTE('0'))(0)
 NINE   = (BYTE('9'))(0)
 DOLLAR = (BYTE('$'))(0)
 SLEFT  = (BYTE('['))(0)
 RLEFT  = (BYTE('('))(0)
 SRIGHT = (BYTE(']'))(0)
 RRIGHT = (BYTE(')'))(0)
 CHARA  = (BYTE('A'))(0)
 CHARZ  = (BYTE('Z'))(0)
 PLUS   = (BYTE('+'))(0)
 ATSIGN = (BYTE('@'))(0)
 PERIOD = (BYTE('.'))(0)
 SEMICOLON  = (BYTE(';'))(0)
```

```
 UNDERSCORE = (BYTE('_'))(0)
;
;  Determine the commands for moving, depending on the operating system.
;
 CASE OS_FAMILY() OF
    'vms': BEGIN
  RENAME = 'RENAME'
  CURDIR = '[]'
  END
    ELSE: BEGIN
  RENAME = 'mv'
  CURDIR = '.'
  END
 ENDCASE
;
;  Check the input filename.  If OUTDIR was passed, then
;
 IF NOT FILE_EXIST(FILENAME) THEN BEGIN
    MESSAGE = 'Input file ''' + FILENAME + ''' does not exist'
    GOTO, HANDLE_ERROR
 ENDIF
;
;  Determine the name of the input file, and the ultimate output file.  If the
;  output file already exists, then write to a temporary file.
;
 TEMPFILE = 'TEMPORARY.TEMPORARY'
 IF N_ELEMENTS(OUTDIR) EQ 0 THEN OUTFILE = FILENAME ELSE BEGIN
    TEMPFILE = CONCAT_DIR(OUTDIR, TEMPFILE)
    BREAK_FILE, FILENAME, DISK, DIR, NAME, EXT
    OUTFILE = CONCAT_DIR(OUTDIR, NAME+EXT)
    IF NOT FILE_EXIST(OUTFILE) THEN TEMPFILE = OUTFILE
 ENDELSE
;
;  Open up the input file and the temporary output file.
;
 MESSAGE = 'Unable to open input file ''' + FILENAME + ''''
 ON_IOERROR, HANDLE_ERROR
 OPENR, IN, FILENAME, /GET_LUN
 IF KEYWORD_SET(ECHO) THEN PRINT, 'Processing file ' + FILENAME
;
 MESSAGE = 'Unable to open output file ''' + TEMPFILE + ''''
 ON_IOERROR, HANDLE_ERROR
 OPENW, OUT, TEMPFILE, /GET_LUN
;
 MESSAGE = 'Read/write error encountered'
 ON_IOERROR, HANDLE_ERROR
;
;  Step through all the lines in the file.
```

```
;
 WHILE NOT EOF(IN) DO BEGIN
    N_LINES = 0
    LASTCHAR = DOLLAR
    TEMP = ''
    WHILE (NOT EOF(IN)) AND ((LASTCHAR EQ DOLLAR) OR $
     (STRLEN(TEMP) EQ 0)) DO BEGIN
  READF, IN, TEMP
;
;  Convert the line to uppercase, and then to a byte array.
;
  BTEMP = BYTE(STRUPCASE(TEMP))
;
;  Convert all tabs to blanks.
;
  W = WHERE(BTEMP EQ TAB, COUNT)
  IF COUNT GT 0 THEN BTEMP(W) = BLANK
;
;  Ignore all strings delimited by single quotes.
;
  W = [WHERE(BTEMP EQ SQUOTE, COUNT), STRLEN(TEMP)-1]
  FOR I = 0, COUNT-1, 2 DO BEGIN
     I1 = W(I) + 1
     I2 = W(I+1) - 1
     IF I1 LE I2 THEN BTEMP(I1:I2) = BLANK
  ENDFOR
;
;  Ignore all strings delimited by double quotes.  However, first check to see
;  if the character immediately following the first quote is a numeral.  If it
;  is, then it's an octal constant instead.
;
  W = [WHERE(BTEMP EQ DQUOTE, COUNT), STRLEN(TEMP)-1]
  FOR I = 0, COUNT-1, 2 DO BEGIN
     TEST = BTEMP(W(I)+1)
     IF (TEST GE ZERO) AND (TEST LE NINE) THEN I=I-1 ELSE BEGIN
   I1 = W(I) + 1
   I2 = W(I+1) - 1
   IF I1 LE I2 THEN BTEMP(I1:I2) = BLANK
     ENDELSE
  ENDFOR
;
;  Ignore any characters after (and including) the comment character.
;
  W = WHERE(BTEMP EQ SEMICOLON, COUNT)
  IF (W(0) LT N_ELEMENTS(BTEMP)) AND (COUNT GT 0) THEN $
   BTEMP(W(0):*) = BLANK
;
;  Find out if the last character is a continuation character.  If it is, then
```

```
;  remove it from the byte array.
;
  W = WHERE(BTEMP NE BLANK, COUNT)
  IF COUNT GT 0 THEN LASTCHAR = BTEMP(W(COUNT-1)) ELSE $
   LASTCHAR = BLANK
  IF LASTCHAR EQ DOLLAR THEN BTEMP(W(COUNT-1)) = BLANK
;
;  Store the line in the concatenated LINE and BLINE arrays.
;
  IF N_LINES EQ 0 THEN BEGIN
      LINE = TEMP
      BLINE = BTEMP
      N_CHAR = STRLEN(TEMP)
  END ELSE BEGIN
      LINE = LINE + TEMP
      BLINE = [BLINE, BTEMP]
      N_CHAR = [N_CHAR, STRLEN(TEMP)]
  ENDELSE
  N_LINES = N_LINES + 1
    ENDWHILE
;
;  Make sure that any null characters, introduced by zero-length lines, are
;  removed from the byte array.
;
    W = WHERE(BLINE NE 0B, COUNT)
    IF COUNT GT 0 THEN BLINE = BLINE(W)
;
;  If the first character in the line is an @ sign, or a period, then the line
;  is an IDL directive, and shouldn't be changed.
;
    IF (BLINE(0) EQ ATSIGN) THEN GOTO, WRITE_LINE
    W = WHERE(BLINE NE BLANK, COUNT)
    IF COUNT GT 0 THEN IF BLINE(W(0)) EQ PERIOD THEN GOTO, WRITE_LINE
;
;  Remove all words that are actually IDL operators.
;
    OPS = ['EQ','NE','LE','LT','GE','GT','AND','OR','XOR','MOD']
    OPS = ' ' + OPS + ' '
    BTEMP = BLINE
    W = WHERE(((BLINE LT CHARA) OR (BLINE GT CHARZ)) AND $
     ((BLINE LT ZERO) OR (BLINE GT NINE)) AND $
     (BLINE NE UNDERSCORE), COUNT)
    IF COUNT GT 0 THEN BTEMP(W) = BLANK
    TEMP = ' ' + STRING(BTEMP) + ' '
    FOR I = 0,N_ELEMENTS(OPS)-1 DO BEGIN
       W = -1
  REPEAT BEGIN
     W = STRPOS(TEMP,OPS(I),W+1)
```

```
      IF W GE 0 THEN FOR J=0,STRLEN(OPS(I))-3 DO $
       BLINE(W+J) = PLUS
  ENDREP UNTIL W LT 0
     ENDFOR
;
;  Find all [] pairs.  Work from the innermost outward.
;
     REPEAT BEGIN
  WRIGHT = WHERE(BLINE EQ SRIGHT, RCOUNT)
  IF RCOUNT GT 0 THEN BEGIN
      WRIGHT = WRIGHT(0)
      BLINE(WRIGHT) = RRIGHT
      WLEFT = WHERE(BLINE(0:WRIGHT(0)) EQ SLEFT, LCOUNT)
      IF LCOUNT GT 0 THEN BEGIN
   WLEFT = WLEFT(LCOUNT-1)
   BLINE(WLEFT) = RLEFT
;
;  Find the first nonblank character to the left of the left bracket.  If it's
;  alphanumeric, an underscore, or the round right bracket, then change the
;  brackets from square to round.
;
   I = WLEFT - 1
   WHILE (I GT 0) AND (BLINE(I) EQ BLANK) DO I = I - 1
   IF ((BLINE(I) GE CHARA) AND (BLINE(I) LE CHARZ)) OR $
    ((BLINE(I) GE ZERO) AND (BLINE(I) LE NINE)) $
    OR (BLINE(I) EQ RRIGHT) OR $
    (BLINE(I) EQ UNDERSCORE) THEN BEGIN
       STRPUT, LINE, '(', WLEFT
       STRPUT, LINE, ')', WRIGHT
   ENDIF
      ENDIF
  ENDIF
     ENDREP UNTIL RCOUNT EQ 0
;
;  Write the modified line to the output file.
;
WRITE_LINE:
     FOR I = 0,N_LINES-1 DO BEGIN
  PRINTF, OUT, STRMID(LINE,0,N_CHAR(I))
  IF N_CHAR(I) LT STRLEN(LINE) THEN LINE = $
   STRMID(LINE,N_CHAR(I),STRLEN(LINE)-N_CHAR(I))
     ENDFOR
 ENDWHILE
;
;  Close the input and output files.  If necessary, then rename the temporary
;  file to the final filename.
;
 FREE_LUN, IN, OUT
```

```
 IF TEMPFILE NE OUTFILE THEN SPAWN, $
  RENAME + ' ' + TEMPFILE + ' ' + OUTFILE
;
;  If the ERRMSG keyword was passed, then set it to the null string to signal
;  success.
;
 IF N_ELEMENTS(ERRMSG) NE 0 THEN ERRMSG = ''
 RETURN
;
;  Error handling point.
;
HANDLE_ERROR:
 IF N_ELEMENTS(ERRMSG) NE 0 THEN ERRMSG = 'IDL5TO4: ' + MESSAGE $
  ELSE MESSAGE, MESSAGE
 END
```