Depth Charges

MICHAEL P. DEAN

Continued explorations of the I/O depths uncover more and more UDF power

dBASE IV version 1.5 offers low level file I/O functions that give you even greater flexibility and power when writing user defined functions (UDFs). Adding to this flexibility is a change that was made to the way in which dBASE IV handles parameters passed to a program or UDF. In version 1.5, you can have a variable number of parameters for a procedure or function. To accomplish this, simply define the maximum number of parameters that you want to allow with the PARAMETERS statement. You can then use the new function PCOUNT() to determine how many parameters were actually passed to your procedure or UDF. Below is a small example:

```
FUNCTION Test
   PARAMETERS one, two, three

   DO CASE
      CASE PCOUNT( ) = 0
         ? "No parameters were passed"
      CASE PCOUNT( ) = 1
         ? "One Parameter was passed"
      CASE PCOUNT( ) = 2
         ? "Two Parameters were passed"
      CASE PCOUNT( ) = 3
         ? "Two Parameters were passed"
      OTHERWISE
         ? "Too many parameters were passed"
   ENDCASE
RETURN .T.
```

As you will see in the UDFs that follow, the *variable parameters* functionality was used to add help text to the functions. If you don't pass any parameters, then a help screen is displayed. Optional parameters were also added so that you can specify a value or use the default value.

One note about variable parameters: the type and value of a declared parameter (declared by the PARAMETERS statement) that does not have a value passed to it will become a logical variable with a value of false. You can use this to your advantage for establishing default values. You can set the default value to false, or if the default value is to be of another character type, you can set it to the new value.

UDF Overview

IsUdf(): This function will open up a .PRG file and scan it for the FUNCTION command. One note: for speed purposes, this UDF will only search the first 100 lines of a program file. You can change this by changing the value of the variable ln_max. If the .PRG file is a UDF, then IsUdf() will return a true value (.T.).

```
Syntax: IsUdf(<expC>)
<expC>: The filename of the UDF to process
        (with or without the path).
```

FindFile(): dBASE IV has the FILE() function which allows you to determine whether a file exists. While this function is useful, it has one major drawback: you must know the exact name (and the path) of the file. You have no way to get a filename when you have only the filespec (for example, *.prg). In the past, you could get this functionality by using a .BIN file, or by routing the output of the DIR command to a file, bringing that file into a database, and then searching the database. Now you can easily do this by using the low level file I/O functions. Even though you still have to create a text file, it can be easily handled with the file I/O functions, then deleted when you are done.

```
Syntax: FindFile(<expC1>, [<expC2>])
<expC1>: Filespec to look for
<expC2>: Optional – Filename of the last file
found. By using the name of the last file found
(and the same filespec), you can find the name of
the next file that meets the filespec.
```

Prg2Proc(): If your programs use lots of subroutines, you've probably found that putting these common routines into a procedure file makes sense (especially since you can have many procedure files open at once). This UDF converts single .PRG files into a procedure file. This UDF shows the use of the low level file I/O functions in reading from a file and writing to another file.

```
Syntax: Prg2Proc(<expC1>, <expL1>, <expL2>
[[[,<expL3>, [<expC2>, [<expC3>]]]
<expC1>: Procedure file to create
<expL1>: Delete source .PRG files
<expL2>: Process UDFs only
<expL3>: Include UDFs
<expC2>: Optional.  .PRG File list (If not
         included or blank, then all .PRG files in
         the selected path are included.)
<expC3>: Optional. .PRG file path (If not
         included, or if it is blank, the current
         directory is used.)
```

Although the third (<expL2> — process UDFs only) and the fourth (<expL3> — Include UDFs) may seem like they perform the same action, they are quite different. When you specify TRUE for *Process UDFs only*, the resultant .PRG will only have UDFs in it. By setting *Include UDFs* to TRUE (and *UDFs Only* to FALSE), you will have a file that has both regular programs, and your UDFs. This functionality allows you to either create a library file (which would contain only your UDFs) that you could activate with SET LIBRARY, and/or a regular PROCEDURE file (containing only your regular programs) that would be activated with SET PROCEDURE.

Proc2Prg(): The opposite of the function Prg2Proc() will take a procedure file and break it up into separate .PRG files. As with the above functions, this function also shows the use of variable parameters.

```
Syntax: Proc2Prg(<expC1>, <expL> [, <expC2>)
<expC1>: Procedure name to process
<expL>:  Delete source file?
<expC2>: Optional.  Drive/directory to write the
         .PRG files to.  If not given, the current
         directory is assumed.
```

Function: IsUDF()

```
FUNCTION IsUDF
  PARAMETER lc_file
  PRIVATE lc_file, ln_fhandle, lc_string, ln_count, ln_max
  *----------------------------------------------------------------------*
  * Author .......: Michael P. Dean
  * Versions .....: dBASE IV 1.5
  * Notes ........: Determine if a program is a UDF.

  *-- Variable Definition
  *   lc_file:      File name
  *   ln_count:     Line counter
  *   ln_max:       Maximum number of lines to check
  *   ln_fhandle:   Program file
  *   lc_string:    String that was read in
  *   ll_return:    Return Value
  *----------------------------------------------------------------------*
  * Return Values:
  *   .T.:  Program file is a UDF
  *   .F.:  Program file is not a UDF
  *   "-1": Program file could not be opened
  *----------------------------------------------------------------------*
  *-- Initialize variables.
  ln_count  = 1
  ln_max    = 100
  ll_return = .F.

  *-- Open file.
  ln_fhandle = FOPEN(lc_file)
  IF FERROR() # 0
```

Function: IsUDF() *continued*

```
    RETURN "-1"
  ENDIF

  *-- Parse the file.
  DO WHILE ln_count <= ln_max
    lc_string = UPPER(LTRIM(FGETS(ln_fhandle)))
    IF lc_string = "FUNCTION"
      ll_return = .T.
      EXIT
    ENDIF
    ln_count = ln_count + 1
  ENDDO

  IF .NOT. FCLOSE(ln_fhandle)
    CLOSE ALL
  ENDIF
RETURN (ll_return)
```

Function: FindFile()

```
FUNCTION FindFile
  PARAMETERS lc_filspec, lc_lastfil
  PRIVATE   lc_filspec, lc_lastfil, ln_fhandle, lc_string, lc_return
  *-------------------------------------------------------------------------------*
  * Author .......: Michael P. Dean
  * Notes ........: Determine if a file exists (allows file specs).
  * Syntax .......: ?/mvar = FindFile("<file spec>" [, "<last file found>"])
  *-- Variable Definition
  *   lc_filspec:   File spec to look for
  *   lc_lastfil:    Last file found (allows for finding next file)
  *   ln_lastpos:   Position in the string of the last file
  *   ln_blank:     Position of a blank in the string
  *   lc_return:    Return Value
  *   ln_fhandle:   File handle
  *   lc_string:    String read in from file
  *   lc_parse:     Character string for parsing
  *   lc_console:   Status of SET CONSOLE
  *-------------------------------------------------------------------------------*
  *   Return Values:
  *    <filename>:  File found
  *    NULL:  No Files found (If this was a find first, no files matched the
  *           filespec, else, there are no more files in the directory
  *            matching the filespec)
  *   -1:  Not enough parameters passed
  *   -2:  Could not open directory file with Low Level File I/O
  *   -3:  Directory File would not close, had to do a CLOSE ALL
  *-------------------------------------------------------------------------------*
  *-- Make sure required parameters are passed.
  IF PCOUNT() < 1
    CLEAR
    TEXT
      The syntax is:

        ?/mvar = FindFile(<expC1> [,<expc2>)

          <expC1>:  filespec to search for
          <expC2>:  By including the name of the last file that was found,
```

Function: FindFile() *continued*

```
                FindFile() can return the filename of the next file
                that matches the filespec

        NOTE: If you know the entire file name (name and extension), and you
        just want to see if it exists, use the dBASE FILE() function.


      Examples:
        mvar = FindFile("*.PRG")
        mvar = FindFile("*.PRG", "MYPROC.PRG")
        mvar = FindFile("P*.*)

  ENDTEXT
  RETURN "-1"
ENDIF

*-- Check Environment.
lc_console = SET("CONSOLE")
SET CONSOLE OFF

*-- Check for the alternate file
IF FILE("FindFile.DMP")
  ERASE FindFile.DMP
ENDIF

*-- Setup alternate file and initialize variables.
lc_string = ""
lc_return = ""
SET ALTERNATE TO FindFile.DMP
SET ALTERNATE ON

*-- Run the Directory.
DIR &lc_filspec

SET ALTERNATE TO
SET ALTERNATE OFF

*-- Open The Directory File.
ln_fhandle = FOPEN("FindFile.DMP")
IF FERROR() # 0
  ERASE FindFile.DMP
  RETURN "-2"
ENDIF

*-- The alternate file always starts with a blank line.  The first FGETS()
*   is done to get by that line, then the second is done to get the first
*   line of programs.

lc_string = FGETS(ln_fhandle)
lc_string = FGETS(ln_fhandle)

*-- Return the name of the first file.
IF TYPE("lc_lastfil") = "L" .OR. ISBLANK(lc_lastfil)
  IF .NOT. ISBLANK(lc_string)
    lc_return = LTRIM(SUBSTR(lc_string, 1, (AT(" ", lc_string) - 1)))
  ENDIF
```

Function: FindFile() *continued*

```
  ELSE
    DO WHILE .NOT. ISBLANK(lc_string)
      ln_lastpos = AT(lc_lastfil, lc_string)
      IF ln_lastpos = 0
        lc_string = FGETS(ln_fhandle)
        LOOP
      ELSE

        *-- Is the filename actually part of another filename
        IF ln_lastpos > 1
          IF SUBSTR(lc_string, (ln_lastpos - 1), 1) # " "
            lc_string = SUBSTR(lc_string, (ln_lastpos + 1))
            LOOP
          ENDIF
        ENDIF
      ENDIF

      *-- Parse out the last file name.
      lc_parse = RTRIM(SUBSTR(lc_string, ln_lastpos))
      ln_blank = AT(" ", lc_parse)
      IF ln_blank = 0

        *-- File was the last one on that line, get the first file name from
        *   the next line.
        lc_string = FGETS(ln_fhandle)
        IF ISBLANK(lc_string)

          *-- No More Files
          lc_return = ""
        ELSE
          lc_return = LTRIM(SUBSTR(lc_string, 1, (AT(" ", lc_string) - 1)))
        ENDIF
        EXIT
      ELSE

        *-- Parse out the last file name.
        lc_parse = LTRIM(SUBSTR(lc_parse, ln_blank))
        ln_blank = AT(" ", lc_parse)
        IF ln_blank = 0
          lc_return = lc_parse
        ELSE
          lc_return = SUBSTR(lc_parse, 1, (ln_blank - 1))
        ENDIF
        EXIT
      ENDIF
    ENDDO
  ENDIF

  *-- Close and delete the file.
  IF .NOT. FCLOSE(ln_fhandle)
    CLOSE ALL
    lc_return = "-3"
  ENDIF

  *-- ERASE the file.
  ERASE FindFile.DMP
```

Function: FindFile() *continued*

```
  *-- Reset System.
  SET CONSOLE &lc_console

RETURN (lc_return)
```

Function: Prg2Proc()

```
FUNCTION Prg2Proc
  PARAMETERS  lc_procfil, ll_srcdel, ll_udfonly, ll_incudf, lc_prglist, lc_path
  PRIVATE     lc_procfil, lc_path, lc_prglist, lc_file, ln_fhandle, ln_infile,;
              lc_dump, lc_string, ln_write, ln_dot, ll_udf, lc_return,;
              ll_srcdel, ll_incudf, ll_udfonly

  *-------------------------------------------------------------------------*
  * Author .......: Michael P. Dean
  * Notes ........: Take multiple .PRG files and make them into 1 PROCEDURE or
  *                 LIBRARY file.
  * Syntax .......: ?/mvar = Prg2Proc(<proc name>, <delete source>, <UDFs Only>
  *                                   [[[,<include UDFS>], <prg file list>], <prg path>])
  *-- Variable Definition
  *   lc_procfil: Procedure file name
  *   ll_srcdel:  Delete the source
  *   ll_incudf:  Include UDFs
  *   lc_path:    Path of .PRG files
  *   lc_prglist: List of programs to process
  *   lc_file:    File name to process
  *   ln_fhandle: OutPut File handle
  *   ln_infile:  InPut File handle
  *   lc_dump:    Miscellaneous variable
  *   lc_string:  String that was read in
  *   ln_write:   Number of characters written
  *   ln_dot:     Position of the "." in the filename, or position of the ","
  *               in a file list.
  *   ll_udf:     Is the file a UDF
  *   ll_udfonly: Process UDFs only (for a LIBRARY file)
  *   lc_return:  Return Value
  *-------------------------------------------------------------------------*
  *-- Return Values
  *   NULL:  Everything okay
  *   -1:    Procedure file could not be opened.
  *   -2:    No files to process
  *   -3:    One of the input files could not be opened
  *   -4:    One of the input files could not be closed, a CLOSE ALL was done
  *   -5:    Procedure file could not be closed, a CLOSE ALL was done
  *-------------------------------------------------------------------------*
  *-- Make sure required parameters were passed
  IF PCOUNT() < 3
    ? CHR(7)
    CLEAR
    TEXT
      The syntax is:

        ?/mvar = Prg2Proc(<expC1>, <expL1>, <expL2> [, <expL3> [,<expC2>,
                          [<expC3>]]])

          <expC1>:  Procedure file to create
```

Function: Prg2Proc()

```
            <expL1>:  Delete source .PRG files

            <expL2>:  Process UDFs only.  Use this option to create a "UDF
                      LIBRARY" file.  If this option is TRUE, <expL3> is
                      ignored.

            <expL3>:  Include UDFs (Optional:  DEFAULT VALUE: False)

            <expC2>:  .PRG File list (optional:  If not included, or if it is
                      blank, then all .PRG files in the selected path are
                      included.)

            <expC3>:  .PRG file path (optional:  If not included, or if it is
                      blank, the the current directory is used.)

        Examples:
          mvar = Prg2Proc("Newfile", .T., .F.)
          mvar = Prg2Proc("Newfile", .F., .F., "File1, File2.PRG, File3")
          mvar = Prg2Proc("Newfile", .T., .T., "", "C:\MYDIR")
      ENDTEXT

      RETURN "-1"
    ENDIF

    *-- Initialize Variables.
    lc_path    = IIF(PCOUNT() < 6 .OR. (PCOUNT() >= 5 .AND. ISBLANK(lc_path)),;
                 SET("DIRECTORY"), lc_path)
    lc_prglist = IIF(PCOUNT() < 5 .OR. (PCOUNT() >= 4 .AND. ISBLANK(lc_prglist)),;
                 "*.PRG", lc_prglist)
    lc_procfil = IIF(UPPER(RIGHT(lc_procfile, 4)) = ".PRG", lc_procfil,;
                 lc_procfil + ".PRG")
    ll_incudf  = IIF(ll_udfonly, .T., IIF(PCOUNT() < 4, .F., ll_incudf))
    lc_file    = ""
    lc_return  = ""
    ll_udf     = .F.

    *-- Open the Procedure File.
    IF FILE(lc_procfil)
      CLEAR
      lc_dump = " "
      @ 3,5 SAY "File " + lc_procfil + " Already Exists"
      @ 5,5 SAY "Overwrite?" GET lc_dump PICTURE "@M Y,N" MESSAGE;
        "Press <space bar> to toggle choice"
      READ
      IF UPPER(lc_dump) = "Y"
        ln_fhandle = FCREATE(lc_procfil)
      ELSE
        RETURN "-1"
      ENDIF
    ELSE
      ln_fhandle = FCREATE(lc_procfil)
      IF FERROR() # 0
        RETURN "-1"
      ENDIF
    ENDIF
```

Function: Prg2Proc() *continued*

```
*-- Start Process.
CLEAR
@ 3,5 SAY "Creating Procedure file: " + lc_procfil
DO WHILE .T.
  IF lc_prglist = "*.PRG"

    *-- If lc_file is blank, then this is the first run.
    IF ISBLANK(lc_file)
      lc_file = FindFile(lc_path + IIF(RIGHT(lc_path, 1) = "\", "", "\") + "*.PRG")
      *-- If lc_file is blank, no files were found.
      IF ISBLANK(lc_file)
        lc_return = "-2"
        EXIT
      ENDIF
    ELSE
      lc_file = FindFile(lc_path + IIF(RIGHT(lc_path, 1) = "\", "", "\") +;
                "*.PRG", lc_file)
      *-- If lc_file is blank, all files have been processed
      IF ISBLANK(lc_file)
        lc_return = ""
        EXIT
      ENDIF
    ENDIF
  ELSE
    IF ISBLANK(lc_prglist)
      EXIT
    ENDIF

    ln_dot     = AT(",", lc_prglist)
    lc_file    = UPPER(RTRIM(SUBSTR(lc_prglist, 1, IIF(ln_dot > 0,;
                 (ln_dot - 1), LEN(lc_file)))))
    lc_file    = lc_file + IIF(UPPER(RIGHT(lc_file, 4)) # ".PRG", ".PRG", "")
    lc_prglist = IIF(ln_dot = 0, "",;
                     LTRIM(SUBSTR(lc_prglist, (ln_dot + 1))))
  ENDIF

  *-- Make sure you don't open this file or the PROCECEDURE file.
  IF "PRG2PROC" = UPPER(lc_file) .OR. UPPER(lc_procfil) = UPPER(lc_file)
    LOOP
  ENDIF

  *-- See if the file is a UDF.
  ll_udf = IsUDF(lc_file)

  IF (.NOT. ll_incudf .AND. ll_udf) .OR. (ll_udfonly .AND. .NOT. ll_udf)
    LOOP
  ENDIF

  *-- Open Input File.
  @ 5,5 SAY "Processing File: " + lc_file

  ln_inhandle = FOPEN(lc_path + IIF(RIGHT(lc_path, 1) = "\", "", "\") +;
                lc_file)

  *-- Write out PROCEDURE Line.
  IF .NOT. ll_udf
```

Function: Prg2Proc() *continued*

```
        ln_dot    = AT(".", lc_file)
        ln_write  = FPUTS(ln_fhandle, "PROCEDURE " + UPPER(SUBSTR(lc_file, 1,;
                    IIF(ln_dot > 0, (ln_dot - 1), LEN(lc_file)))))
      ENDIF
      ll_udf = .F.

      *-- Write out the file
      DO WHILE .NOT. FEOF(ln_inhandle)
        lc_string = FGETS(ln_inhandle)

        *-- Make sure the program does not have a PROCEDURE statement.
        IF LTRIM(lc_string) = "PROC"
          lc_string = FGETS(ln_inhandle)
        ENDIF

        *-- Write out the line.
        ln_write = FPUTS(ln_fhandle, lc_string)
      ENDDO

      *-- Write out an "EOP" marker
      ln_write = FPUTS(ln_fhandle, CHR(13) + CHR(10) + "*-- EOP: " + lc_file +;
                 CHR(13) + CHR(10) + "*" + REPLICATE("-", 45) + "*" +;
                 CHR(13) + CHR(10))

      *-- Close the file.
      @ 5, 5 CLEAR TO 7,79
      IF .NOT. FCLOSE(ln_inhandle)
        CLOSE ALL
        lc_return = "-4"
        EXIT
      ENDIF

      *-- Delete the source if requested.
      IF ll_srcdel
        ERASE lc_file

        *-- The variable lc_file must be cleared since the file was deleted.
        *   This way FindFile() will start with the first file in the list.
        lc_file = ""
      ENDIF

    ENDDO

    *-- Close File.
    IF .NOT. FCLOSE(ln_fhandle)
      CLOSE ALL
      lc_return = "-5"
    ENDIF

  RETURN (lc_return)
```

More UDFs on page 26 - 28

Function: Proc2Prg()

```
FUNCTION Proc2Prg
  PARAMETERS lc_procfil, ll_srcdel, lc_path
  PRIVATE   lc_procfil, lc_path, lc_file, ln_fhandle, ln_infile,;
            lc_string, ln_write, lc_return, ll_srcdel

  * Author .......: Michael P. Dean
  * Notes ........: Convert a Procedure File to multiple .PRG files.
  * Syntax .......: ?/mvar = Proc2Prg(<proc name>, [<delete source>,
  *                                   [<path to put .PRG files>]])
  *-- Variable Definition
  *   lc_procfil: Procedure file name
  *   ll_srcdel:  Delete the source
  *   lc_path:    Path to put .PRG files in
  *   lc_file:    File name being processed
  *   ln_fhandle: OutPut File handle
  *   ln_infile:  InPut File handle
  *   lc_string:  String that was read in
  *   ln_write:   Number of characters written
  *   lc_dump:    Dump Variable
  *   ln_line:    Line counter
  *   lc_return:  Return Value
  *-------------------------------------------------------------------------*
  *-- Return Values
  *   NULL:  Everything okay
  *   -1:    Procedure file could not be opened.
  *   -2:    No PROCEDURE/FUNCTION statement could be found
  *   -3:    One of the output files could not be opened
  *   -4:    One of the output files could not be closed, a CLOSE ALL was done
  *   -5:    Procedure file could not be closed, a CLOSE ALL was done
  *-------------------------------------------------------------------------*
  * --- Make sure that the correct number of parameters was passed.
  IF PCOUNT() < 2
    ? CHR(7)
    CLEAR
    TEXT
      The syntax is:

        ?/mvar = Proc2Prg(<expC1>, <expL> [, <expC2>)

          <expC1>:  Procedure procedure name to process
          <expL>:   Delete source file?
          <expC2>:  Drive/directory to write the .PRG files to.  This
                    parameter is optional.  If not given, the current
                    directory is assumed.

      Examples:

        ?/mvar = Proc2Prg("MyProc", .T., "C:\MYDIR")
        ?/mvar = Proc2Prg("MyProc.PRG", .F.)
    ENDTEXT
    RETURN "-1"
  ENDIF

  * --- Initialize variabales
  lc_procfile = UPPER(lc_procfile) + IIF(UPPER(RIGHT(lc_procfile, 4)) = ".PRG",;
                "", ".PRG")
```

Function: Proc2Prg() ***continued***

```
lc_path    = IIF(PCOUNT() < 3, SET("DIRECTORY"), IIF(ISBLANK(lc_path),;
               SET("DIRECTORY"), RTRIM(lc_path)))
lc_path    = lc_path + IIF(RIGHT(lc_path, 1) = "\", "", "\")
lc_return  = ""
lc_string  = ""
ln_line    = 1

* --- Open procedure file
ln_inhandle = FOPEN(lc_procfile)
IF FERROR() # 0
  RETURN "-1"
ENDIF

CLEAR
@ 3, 5 SAY "Processing File: " + lc_procfile

* --- Find the start of the first Procedure/Function
DO WHILE LTRIM(lc_string) # "PROC" .AND. LTRIM(lc_string) # "FUNC" .AND.;
  .NOT. FEOF(ln_inhandle)

  @ 5, 5 SAY "Line Number: "+ LTRIM(STR(ln_line))
  lc_string = FGETS(ln_inhandle)
  ln_line = ln_line + 1
ENDDO

* --- Make sure that there was a Procedure/Function Statement
IF FEOF(ln_inhandle)
  IF .NOT. FCLOSE(ln_inhandle)
    CLOSE ALL
  ENDIF
  RETURN "-2"
ENDIF

DO WHILE .NOT. FEOF(ln_inhandle)

  * --- Parse out file name
  lc_file = LTRIM(lc_string)
  lc_file = lc_path + LTRIM(SUBSTR(lc_string, (AT(" ", lc_file) + 1))) +;
            IIF(RIGHT(RTRIM(lc_string), 4) # ".PRG", ".PRG", "")

  * --- Is the file name the same as this program?
  IF lc_file = PROGRAM()
    lc_string = FGETS(ln_inhandle)
    ln_line   = ln_line + 1
    DO WHILE LTRIM(lc_string) # "PROC" .AND. LTRIM(lc_string) # "FUNC" .AND.;
      .NOT. FEOF(ln_inhandle)

      @ 5, 5 SAY "Line Number: "+ LTRIM(STR(ln_line))
      lc_string = FGETS(ln_inhandle)
      ln_line = ln_line + 1
    ENDDO
  ENDIF

  * --- Make sure that the ouput filename does not exist
  IF FILE(lc_file)
    CLEAR
```

Function: Proc2Prg() *continued*

```
        lc_dump = "Y"
        @  9, 5 SAY "OutPut Program File " + lc_file + " Already Exists"
        @ 11, 5 SAY "Overwrite?" GET lc_dump PICTURE "@M Y,N" MESSAGE;
          "Press <space bar> to toggle choice"
        READ
        @ 9, 1 CLEAR TO 11, 79
        IF UPPER(lc_dump) = "Y"
          ln_fhandle = FCREATE(lc_file)
        ELSE
          lc_return = "-3"
          EXIT
        ENDIF
      ELSE
        ln_fhandle = FCREATE(lc_file)
        IF FERROR() # 0
          lc_return = "-3"
          EXIT
        ENDIF
      ENDIF

      * -- Read in the string
      lc_string = FGETS(ln_fhandle)
      DO WHILE LTRIM(lc_string) # "PROC" .AND. LTRIM(lc_string) # "FUNC" .AND.;
        .NOT. FEOF(ln_inhandle)

        ln_line = ln_line + 1
        @ 5, 5 SAY "Line Number: "+ LTRIM(STR(ln_line))
        * -- Write out the string to the new file and read in the next line.
        ln_write  = FPUTS(ln_fhandle, lc_string)
        lc_string = FGETS(ln_inhandle)
      ENDDO

      * --- Close the output file
      IF .NOT. FCLOSE(ln_fhandle)
        CLOSE ALL
        lc_return = "-4"
        EXIT
      ENDIF
    ENDDO

    * --- Close input file
    IF .NOT. FCLOSE(ln_inhandle)
      CLOSE ALL
      lc_return = "-5"
    ENDIF

    * --- If the user wants the source file deleted, and all program files were
    * -   written out, and the source file was closed, erase the source file.
    IF ll_srcdel .AND. ISBLANK(lc_return)
      ERASE &lc_procfil
    ENDIF

  RETURN (lc_return)
```