LNG HDY

North American Response Centers

HP 3000 APPLICATION NOTE #21

COBOLII AND MPE INTRINSICS

HEWLETT PACKARD

January 15, 1987
Document P/N 5958-5824/2703

HP 3000 APPLICATION NOTES *are published by the North American Response Centers twice a month and distributed with the Software Status Bulletin. These notes address topics, where the volume of calls received at the Centers indicates a need for addition to or consolidation of information available through HP support services. You may obtain previous notes (single copies only, please) by returning the attached Reader Comment Sheet listing their numbers.*

Note #	*Published*	*Topic*
1	2/21/85	*HP 3000 Printer Configuration Guide (superseded by note #4)*
2	10/15/85	*Terminal Types for HP 3000 HPIB Computers (superseded by note #13)*
3	4/01/86	*HP 3000 Plotter Configuration Guide*
4	4/15/86	*HP 3000 Printer Configuration Guide - Revised*
5	5/01/86	*MPE System Logfile Record Formats*
6	5/15/86	*HP 3000 Stack Operation*
7	6/01/86	*Cobol II/3000 Programs: Tracing Illegal Data*
8	6/15/86	*KSAM Topics: Cobol's Index I/O; File Data Integrity*
9	7/01/86	*Port Failures, Terminal Hangs, TERMDSM*
10	7/15/86	*Serial Printers - Configuration, Cabling, Muxes*
11	8/01/86	*System Configuration or System Table Related Errors*
12	8/15/86	*Pascal/3000 - Using Dynamic Variables*
13	9/01/86	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	9/15/86	*Laser Printers - A Software and Hardware Overview*
15	10/01/86	*FORTRAN Language Considerations - A Guide to Common Problems*
16	10/15/86	*IMAGE: Updating to TurboImage & Improving Data Base Loads*
17	11/01/86	*Optimizing VPLUS Utilization*
18	11/15/86	*The Case of the Suspect Track for 792X Disc Drives*
19	12/01/86	*Stack Overflows: Causes & Cures for COBOL II Programs*
20	1/01/87	*Output Spooling*

COBOLII and MPE INTRINSICS

The purpose of this article is to give application programmers a guide for using COBOLII with MPE intrinsics as defined in the Intrinsics manual such that programmers are *comfortable* using the intrinsics when the need arises. This article will describe the COBOLII interface with these intrinsics, how to interpret the Intrinsics manual for COBOLII use, and finally will recommend certain intrinsics which are useful for many applications.

Intrinsic Overview

MPE intrinsics can be thought of as system defined subroutines which provide specified functions to user programs. The functions were thought to be common routines which most application programmers would encounter during their software development. Most of these intrinsics were created during the mid 1970s and specifically geared for use with the System Programming Language (SPL) and FORTRAN. As a result, most of the terminology and usage examples were geared for those languages.

Since that time, many of the intrinsics have been incorporated as features of the high level languages (such as COBOLII). Therefore, *many of the intrinsics should never need to be used in a COBOLII shop.* However, the sophistication of COBOLII use has risen such that some of the intrinsics may be needed by specific applications.

Like most subroutines, MPE intrinsics are passed parameters. Many of these parameters are optional, or have defaults provided. Several return values to the user. Many make use of a condition code which describes the status of the intrinsic call. The compiler refers to the SPLINTR file to determine the type and requirements for each parameter. The SPLINTR file is described in Appendix C-1 of the *SPL Reference Manual* (Part No. 30000-90024).

Using the Intrinsics Manual

The goal of this section is to describe how a COBOLII programmer should read and interpret the Intriniscs manual. Once certain fundamental concepts are mastered, it will be possible to read the description of any intrinsic, and to comfortably use them in a COBOLII program.

Let's examine a common, but complicated, MPE intrinsic: FOPEN. FOPEN is the heart of the file system, and has many different options. It embodies most of the features of MPE intrinsics, and thus serves as an excellent example for exploring the COBOLII to MPE intrinsic relationship.

Note: You can find a complete description of the FOPEN intrinsic in the *MPE Intrinsic Manual* (Part No. 32033-90007). You may want to use it to follow along as this section describes line by line, what information is contained in this description.

The first part of the description contains a list of the parameters associated with FOPEN.

```
   I      O-V        BA                  LV        LV        IV
filenum:=FOPEN(formaldesignator,foptions,aoptions,recsize,

                    BA        BA        IV          IV          IV
                 device,formmsg,userlabels,blockfactor,numbuffers,

                     DV         IV          IV         IV
                  filesize,numextents,initialloc,filecode);
```

To someone who has never used intrinsics before, much less COBOL-II with intrinsics, this can appear quite formidable.

The notation *filenum* := indicates that FOPEN is a **function**. For FORTRAN, SPL, and PASCAL programmers, this means that the intrinsic can be used within an arithmetic expression and, much like a variable, will return a value (*filenum*) to be used in that expression. Since COBOL-II doesn't provide a way to call functions from within expressions, this value is acquired instead via the GIVING clause of the CALL statement.

The value so returned is described in the manual under FUNCTIONAL RETURN. For FOPEN, the description of *filenum* is "an integer file number used to identify the opened file in other intrinsic calls". Bear in mind, however, that values can also be returned to the calling program through the passed parameters. *Not all intrinsics are functions, so be sure that you use GIVING only when the intrinsic you're calling is so specified.*

Following the name of the intrinsic is a list of the parameters it requires. These are the items that should appear in the USING clause of CALL. Above each parameter is a code which describes its type, specified in SPL terms. Using the table below, you can translate these to the corresponding COBOL-II data types:

When The Intrinsics Manual Specifies:	What COBOLII Really Needs:
BA - Byte Array	PIC X(n).
I - Integer	PIC S9(4) COMP.
L - Logical	PIC 9(4) COMP.
LA - Logical Array	PIC X(n).
DI - Double Integer	PIC S9(9) COMP.
DA - Double Array	PIC S9(4) OCCURS n.
R - Real	No COBOL-II equivalent.

If a V appears in the type code (see *foptions*), the parameter is expected to be passed **by value**. Otherwise, the intrinsic will expect the parameter to be passed **by reference**. If the notation O-V (**option variable**) is shown above the intrinsic name, one or more of the parameters is optional (Otherwise, all parameters are required.) We shall describe pass-by-value, pass-by-reference, and optional parameters later in this Note.

Below the parameter list is a brief description of the intrinsic, followed by a detailed discussion of each parameter. The parameter descriptions include information about what the intrinsic expects or returns in the parameter, whether it is optional, and the effect and meaning of various values that the parameter may have. It's important that you have a good understanding of this material before you use the intrinsic.

Calling the Intrinsic

Now that you understand the type and meaning of the intrinsic's parameters, it's time to code the actual call. COBOL's normal CALL statement will pass by default all parameters as word addresses by-reference. This won't do for FOPEN, since most parameters are specified as by-value and those that aren't are byte-addressed. In addition, FOPEN is *option variable* which requires special handling not provided by the normal COBOL-II CALL statement.

So how does one call FOPEN from COBOL-II? There are two ways. The first and simpler method is to use the INTRINSIC option on the CALL. This option causes the compiler to look up the intrinsic definition in the file SPLINTR.PUB.SYS, and then to pass the parameters you specify according to that definition. In this case, you needn't worry about value vs. reference parameters, byte addresses, or option variable processing.

The second and by far more difficult method is to omit INTRINSIC from the CALL and generate the correct sequences yourself. While COBOL-II provides ways for you to pass by-value parameters, byte addresses, and the option variable information, it has no way of checking whether you've done this correctly. As you can imagine, this method is error prone and therefore we don't recommend it.

Using either method, it's essential that the order of the parameters as stated in the manual be followed exactly.

Using the INTRINSIC option, a call to FOPEN might look like:

```
CALL INTRINSIC "FOPEN" USING  FILE-NAME  FILE-OPTIONS  ACCESS-OPTIONS  \\
  DEVICE-NAME  GIVING  FILE-NUMBER.
```

This calls FOPEN specifying file name, file options, access options, omitting record size, and specifying a device. Since the other parameters were not needed (and were shown as optional in the manual), they did not need to be included. FOPEN returns a file number to the variable specified after GIVING, in this case FILE-NUMBER. COBOL allows you to include commas between the parameters if you wish, but they have no effect. The \\ following ACCESS-OPTIONS indicates that *recsize* is omitted from this call.

If a parameter is specified as pass-by-value (V in the type code), you may pass either a data name or a specific numeric value in the call. Pass-by-reference parameters only allow a data name to be used. If you wished to call FOPEN with *foptions* and *aoptions* both set to 1, you could code the call this way:

```
CALL INTRINSIC "FOPEN" USING  FILE-NAME  1  1  \\  DEVICE-NAME
  GIVING FILE-NUMBER.
```

If you are on MPE version G.A1.04 ("T-delta-4"), G.A2.A0 ("UA-MIT") or later, you may also use the BY REFERENCE and BY CONTENT phrases available with COBOL-85. For example:

```
CALL INTRINSIC "FOPEN" USING FILE-NAME BY REFERENCE,
                             FILE-OPTIONS BY CONTENT,
                             ACCESS-OPTIONS BY CONTENT,
                             \\,
                             DEVICE-NAME BY REFERENCE,
                             GIVING FILE-NUMBER.
```

To give you an idea of the amount of effort that the INTRINSIC option saves you, here's what the *same* call to FOPEN would look like without it:

```
CALL "FOPEN" USING  @FILE-NAME  \FILE-OPTIONS\  \ACCESS-OPTIONS\  \\
  @DEVICE-NAME  \\  \\  \\  \\  \\  \\  \\  \\  \\  \%16400\
  GIVING FILE-NUMBER.
```

Note that we were forced to include one-word place holders (\\) for all of the parameters we didn't pass (for *filesize* we needed *two* place holders, since it's a two word parameter passed by value), we needed to specify which parameters were to be passed by-value (those are the ones inside back-slashes), we had to specify byte addressing (@) for some, and that we included the option variable information (the \%16400\) indicating which parameters were actually present.

Intrinsic Parameters

This section describes how to format and use certain types of intrinsic parameters in COBOL-II.

FOPEN's *formaldesignator* parameter requires a string of alphanumeric characters and special characters as used in file names. The name must be terminated by a non-alphanumeric character, usually a single space. Therefore, your WORKING-STORAGE Section might have a entry that looks like this:

```
01  FILE-NAME        PIC X(8) VALUE "MYFILE ".
```

The level is 01 (actually all that is needed is to ensure that the parameter starts on a word boundry). Because the parameter type is a byte array, a PIC X field is used. The size of that field depends on the size of my file name plus a single space. So,

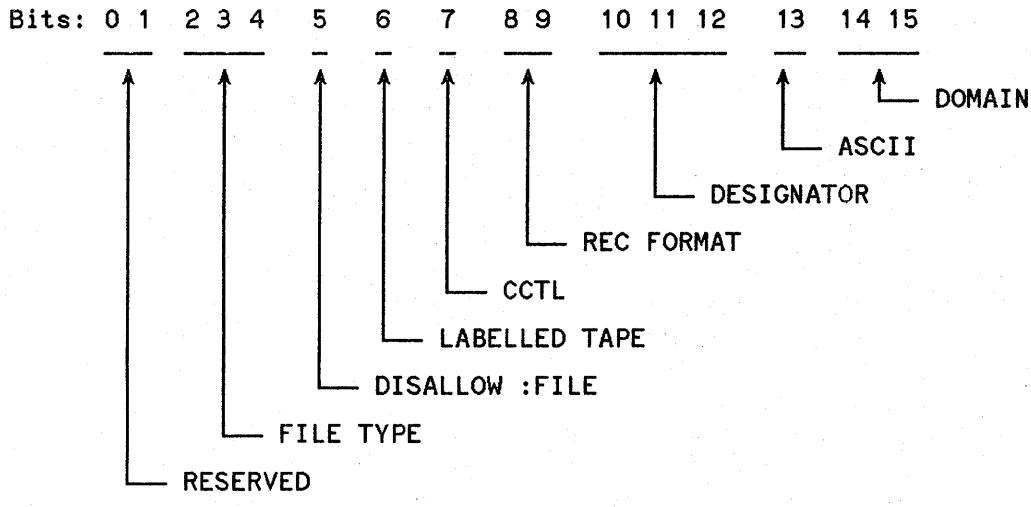
```
01  FILE-NAME        PIC X(17)  VALUE "MYFILE.PUB.SMITH ".
```

would be equally as vaild.

The next parameter is the file options which describe the file to be opened. Here the parameter requires the setting of bits with a logical BY VALUE parameter. The bits are described with standard SPL notation.

Any COBOLII PIC S9(4) COMP field (required here) contains 16 bits (1 word) of information. Depending on the parameter, a single bit may have meaning, or a group of two or more bits in a word may have a meaning. The bits are numbered 0 to 15 starting from the left. SPL notation is of the form s:n where s is the starting bit and n is the number of conscutive bits which together have a specific meaning. For example, the notation 14:2 means bits 14 and 15, while 2:4 means bits 2 through 5 (4 bits).

So, in the FOPTIONS parameter, the bit groupings have the following meanings:



Groups that have a single bit have two options (on or off). Groups with more than a single bit have many options. These options are stated in the manual. For example, the domain group with two bits has four value 00,01,10, and 11. If you wanted to specify and old file, the 01 pattern would be indicated. Therefore, in an empty word, this pattern would be inserted:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	1

The next group is a single bit specifying ascii or binary. To indicate ASCII, the next bit would be filled in:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	-	-	-	-	-	-	-	-	-	-	-	-	1	0	1

and so on. Suppose you have determined what all the groups should indicate resulting in this pattern:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	0	0	0	1	0	0	0	1	0	1

This indicates an existing permanent ASCII file, which has its name indicated in the file name parameter. The file has variable length records, file equations disallowed, no carriage control, it is not a labeled tape, and it is a message file. Bits zero and one are set to zero for MPE.

You need to translate this into something COBOLII will understand since COBOLII cannot directly set bit patterns. This involves a binary to octal conversion grouping from right to left. For those not comfortable with octal and binary numbering schemes, the following charts are provided:

The first chart indicates that bits 13,14,and 15 form the first octal digit; 10, 11, and 12 form the second, etc.

The second table indicates which bit patterns form which octal numbers.

BITS	DIGIT
131415	1
101112	2
789	3
456	4
123	5
0	6

BIT PATTERN	OCTAL NUMBER
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Grouping bits 131415 together forms a pattern of 101 which is an octal 5. So the first octal digit of our number which will set the desired bit pattern for FOPTION is 5. Completing the translation we get 032105 as the octal number. Fortunately, COBOLII accepts octal values as indicated by a leading % sign.

To set our desired FOPTIONS, we would then set our WORKING-STORAGE as follows:

```
01  FILE-OPTIONS          PIC S9(4) COMP VALUE %032105.
```

The next parameter is the AOPTIONS (access options). These are set using the same technique of bit patterns. For demonstration purposes, we set several of the file access options. However, most of the defaults typically would serve our purposes.

In the case of AOPTIONS, if we left everything at their default values, this would give us read access to the file.

The rest of the parameters do various things to manipulate the file system. The file system is beyond the scope of this paper, but it is relevant to note that the vast majority of FOPEN parameters have corresponding FILE equation options. Many of these options can be specified in the SELECT statement as well.

Suggested uses of the FOPEN intrinsic follow in the next section.

FOPEN does, however, return a condition code. As the manual states a CCE means successful, a CCL states that the request was denied, and a CCG is not returned.

In order to trap the condition code in COBOLII, you must use the SPECIAL-NAMES portion of the ENVIRONMENT DIVISION. For example,

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. HP3000
OBJECT-COMPUTER. HP3000.
SPECIAL-NAMES.
CONDITION-CODE IS <<your choice>>.
```

Suppose you had chosen the name of 'CC' for your condition code name. In COBOLII, CC<0 means CCL; CC=0 means CCE; and CC>0 means CCG.

Therefore, a complete call to FOPEN might look something like this:

```
CALL INTRINSIC "FOPEN" USING FILE-NAME, 1, 0, GIVING FILE-NUMBER.
                             GIVING FILE-NUMBER.

IF CC NOT = 0
  PERFORM ERROR-ROUTINE.
```

In the above example, an FOPEN was made to an old permanent file requesting read access. If for any reason, the access was not granted, a negative condition code would be returned which would send us into an error routine.

Note: The condition code check should immediately follow the call to FOPEN since other code (such as a display statement) will affect the status of the condition code.

One of the most common problems new programmers encounter is failing to check error or condition codes whenever they are available.

Assuming the call was successful, the program might continue to an FREAD which might look like this:

```
CALL INTRINSIC "FREAD" USING FILE-NUMBER, DATA-BUFFER, 72
  GIVING DATA-LENGTH.

IF CC > 0
  PERFORM END-OF-FILE-ROUTINE
ELSE IF CC < 0
  PERFORM ERROR-ROUTINE.
```

The FREAD call needs the FILE-NUMBER from the FOPEN call. The *target* parameter is a logical array (PIC X) field where the data read is stored, and the *tcount* is the length of data to be read (in this case 72 words). The length of data actually returned is stored in the integer DATA-LENGTH.

INTRINSIC USAGE

As mentioned above, many of the functions of the MPE Intrinsics have been incorporated into the COBOLII language features. In this section we will try to indicate which intrinsics are most useful to application programmers, and which can be avoided by using the high level features.

Generally speaking, the more you can avoid using intrinsics, the better off you are. Intrinsics are highly machine dependent, and extensive use of them ties application code down to a specific operating system. If the language incorporates the intrinsic features, you should try to use those high level features.

In our experience, features such as process handling and extra data segments should be avoided whenever possible in the application design phase. Their performance overhead and programming complexity are considerable. Hence, those intrinsics are not described below.

FOPEN

Since many of the parameters of the FOPEN intrinsic are duplicated in the FILE command, or can be specified in the SELECT clause, the standard OPEN command is generally preferred. Since the FILE-NUMBER can be trapped from the OPEN command for use with later file system intrinsics, the most common need to use the FOPEN intrinsic is to create a new file with userlabels. Since this feature is not part of the BUILD command, you need to use FOPEN, specifying the creation of a new file with FOPTIONS.

The other intrinsics used with userlabels are FREADLABEL and FWRITELABEL. Both need the file number from the OPEN statement.

Once a file has been opened with the OPEN statement, the file name can be used in place of the file number parameter in subsequent intrinsic calls. For example, portions of a COBOLII program might look like this:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MYFILE ASSIGN TO "REALNAME.PUB.SYS"
    ORGANIZATION IS SEQUENTIAL
    FILE STATUS IS FILE-STAT-CODE.

SPECIAL-NAMES.
    CONDITION-CODE IS CC.

DATA DIVISION.
FILE SECTION.
FD MYFILE.
01 DATA-RECORD   PIC X(72).

WORKING-STORAGE SECTION.

01  FILE-STAT-CODE.
    05 STAT-CODE-1  PIC X.
    05 STAT-CODE-2  PIC X.

01  TIME-OUT  PIC S9(4) COMP VALUE 10.

PROCEDURE DIVSION.
FIRST-SECTION.
    OPEN I-O MYFILE.
    IF STAT-CODE-1 = "9"
      PERFORM CHECK-FILE-ERROR.
```

```
CALL INTRINSIC "FCONTROL" USING MYFILE, 4, TIME-OUT.
IF CC NOT = 0
  PERFORM CHECK-INTRINSIC-ERROR.

READ DATA-RECORD
  AT END PERFORM EOF-MYFILE.
IF STAT-CODE-1 = "9" THEN PERFORM CHECK-FILE-ERROR.
```

The above example shows how to place a timeout interval of 10 seconds on the reading of the data record from REALFILE.PUB.SYS. If REALFILE was an empty message file, then the program, instead of waiting for data to be placed in the file, would 'give up' after 10 seconds, and continue on.

Notice, the use of FCONTROL without an FOPEN. Also notice how the MPE file error can be trapped using COBOLII features. (See the FILE-STATUS description in the COBOLII manual for further details).

PAUSE

The PAUSE intrinsic involves a real parameter for which COBOLII has no equivalent. Please see *RC Q&A July 15, 1986* for a complete description of its use.

COMMAND

Often times it is useful to execute a MPE command such as a FILE equation from a user program. An example of calling this intrinsic can be found in the *COBOLII Reference Manual.*

FCONTROL

FCONTROL is one of the most useful of all MPE intrinsics. Through it, you may direct carriage control, echo, timed reads, and enable/disable various I/O functions with files. A complete description of all codes is contained in the *MPE Intrinsics Reference Manual.*

FCONTROL requires the file number to be trapped from the OPEN statement.

WHO

The WHO intrinsic is useful to obtain user atributes. The calling of the intrinsic is straightforward, but the interpretation of some of the parameters is not, since they require bit extraction which COBOLII does not perform. For example, the capability parameter states that if the first word, bit 15, is equal to 1, the user has Save File capability.

The following technique will obtain the needed information:

Suppose a call to the WHO intrinsic had returned these values in the first word:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- - - - - - - - - - -- -- -- -- -- --
0 0 0 0 0 1 1 1 1 0  0  0  0  0  1  1
```

Determine which bit you are interested in. For example, bit 8, which tells if the user has User Logging capability. To see if the bit is on or off, first divide the number returned by the binary place of the bit as determined from right to left as listed in the following table:

Bit Number	Binary Place	Bit Number	Binary Place
15	1	7	256
14	2	6	512
13	4	5	1024
12	8	4	2048
11	16	3	4096
10	32	2	8192
9	64	1	16384
8	128	0	32768

Then, take the file access attributes and divide the results in that word by 128 and store it in a temporary variable, 'B'.

Next, divide B by 2 and multiply the result by 2. Store this in another variable, 'C'.

Next, compare B to C. If B = C, the bit is off. If B <> C then the bit is on.

QUIT

The QUIT intrinsic is useful for two reasons. First, it allows you to put any number you choose in its parameter. This will allow you to indicate the section of code from which your program aborted, e.g. CALL INTRINSIC "QUIT" USING 1 could indicate that your program aborted in the first routine.

The second use is to ensure that if your program runs in batch mode, the job stream will abort if the program aborts, since the QUIT intrinsic changes the system JCW.

FFILEINFO, FGETINFO

These are very useful intrinsics if your program needs to obtain specific information about a particular file. You will have to use the bit extraction technique described above.

FINDJCW, GETJCW, SETJCW, PUTJCW

These intrinsics are very useful for the manipulation of large complicated jobstreams.

ERROR CHECK LIST

The following is a check list to assist you in debugging a COBOLII program with MPE intrinsics:

Symptom: Errors during compile/prepare.

- *Incompatible number of parameters for intrinsic xxxxx*
  Make sure that all required parameters are present.

- *Illegal xxxxx operand in intrinsic xxxxx*
  Make sure that parameters are of the correct type.

- *Intrinsic xxxxx not found in intrinsic file*
  Make sure that the intrinsic name is spelled correctly.

Symptom: Program Aborts.

- *Bounds Violation*
  Check to make sure parameters are being passed correctly.

- *Bounds Violation or Stack Overflow/Underflow*
  Make sure that the contents of the parameters make sense. Locate the offending CALL, and use DISPLAY, TOOLSET or DEBUG to check the contents.

Symptom: Program Runs, but does not work

- Are all error conditions and status codes being trapped and acted upon?

- Are there code statements between the intrinsic calls and the error checks?

- Do the parameters contain the expected values, e.g. has other data been placed in them by accident?

  Are their lengths correct?

  Are their types correct (PIC S9(4) COMP vs. 9(4).)?

  Is a value in WORKING-STORAGE above the parameter values too long, or in an array that has overrun into other WORKING-STORAGE areas?

- Have previous calls on which this call is dependent been successful?

Remember, make generous use of DISPLAY calls, DEBUG, or TOOLSET when first coding/testing your code. The most common errors with intrinsics fail to be diagnosed because the programmer assumes something strange, secret, or complicated is going on with MPE.

Intrinsic calls are very straightforward once you get comfortable with the terminology.