GnuCOBOL guide to interfacing COBOL and C

By Ron Norman, August 2020

Contents

Table of Contents

Combining GnuCOBOL and C Programs

The upcoming sections deal the issues pertaining to calling C language programs from GnuCOBOL programs, and vice versa. Two additional sections provide samples illustrating specifics as to how those issues are overcome in actual program code.

GnuCOBOL Run-Time Library Requirements

Like most other implementations of the COBOL language, GnuCOBOL utilizes a run-time library. When the first program executed in a given execution sequence is a GnuCOBOL program, any run-time library initialization will be performed by the compiled COBOL code in a manner that is transparent to the C-language programmer. If, however, a C program is the first to execute, the burden of performing GnuCOBOL run-time library initialization falls upon the C program. See [C Main Programs Calling GnuCOBOL Subprograms].

String Allocation Differences Between GnuCOBOL and C

Both languages store strings as a fixed-length continuous sequence of characters. COBOL stores these character sequences up to a specific quantity limit imposed by the "PICTURE" clause of the data item. For example: "01 LastName PIC X(15).".

There is never an issue of exactly what the length of a string contained in a "USAGE DISPLAY" data item is — there are always exactly how ever many characters as were allowed for by the "PICTURE" clause. In the example above, "LastName" will always contain exactly fifteen characters; of course, there may be anywhere from 0 to 15 trailing SPACES as part of the current LastName value.

C actually has no "string" data type; it stores strings as an array of "char" data type items where each element of the array is a single character. Being an array, there is an upper limit to how many characters may be stored in a given "string". For example:

```
char lastName[15]; /* 15 chars: lastName[0] through lastName[14] */
```

C provides a robust set of string-manipulation functions to copy strings from one char array to another, search strings for certain characters, compare one char array to another, concatenate char arrays and so forth. To make these functions possible, it was necessary to be able to define the logical end of a string. C accomplishes this via the expectation that all strings (char arrays) will be terminated by a NULL character (x'00'). Of course, no one forces a programmer to do this, but if [s]he ever expects to use any of the C standard functions to manipulate that string they had better be null-terminating their strings!

So, GnuCOBOL programmers expecting to pass strings to or receive strings from C programs had best be prepared to deal with the null-termination issue, as follows:

Pass a quoted literal string from GnuCOBOL to C as a zero-delimited string literal (Z'<string>').

Pass alphanumeric (PIC X) or alphabetic (PIC A) data items to C subroutines by appending an ASCII NULL character (X'00') to them. For example, to pass the 15- character LastName data item described above to a C subroutine:

```
01 LastName-Arg-to-C PIC X(16).
...
MOVE FUNCTION CONCATENATE(LastName,X'00') TO LastName-Arg-to-C
```

And then pass LastName-Arg-to-C to the C subprogram! When a COBOL program needs to process string data prepared by a C program, the embedded null character must be accounted for. This can easily be accomplished with an "INSPECT" statement such as the following:

```
INSPECT Data-From-a-C-Program
REPLACING FIRST X'00' BY SPACE
CHARACTERS BY SPACE AFTER INITIAL X'00'
```

Matching C Data Types with GnuCOBOL USAGE's

Matching up GnuCOBOL numeric Usage's with their C language data type equivalents is possible via the following chart:

COBOL	**C**
BINARY-CHAR UNSIGNED	unsigned char
BINARY-CHAR [SIGNED]	signed char
BINARY-SHORT UNSIGNED	unsigned short
BINARY-SHORT [SIGNED]	short
BINARY-LONG UNSIGNED	unsigned long
BINARY-LONG [SIGNED]	long
BINARY-INT	int
BINARY-C-LONG [SIGNED]	long
BINARY-DOUBLE UNSIGNED	unsigned long long
BINARY-DOUBLE [SIGNED]	long long
BINARY-LONG-LONG	long long
COMPUTATIONAL-1	float
COMPUTATIONAL-2	double
N/A (no equivalent)	long double
COMPUTATIONAL-3 / PACKED-DECIMAL	N/A (no equivalent)
COMPUTAIONAL-4 / BINARY	Big-endian format binary
COMPUTATIONAL-5	Native endian binary
COMPUTAIONAL-X	Big-endian variable size unsigned binary

These sizes conform to the COBOL standard and the minimum sizes of the COBOL types are the same as the minimum sizes of the corresponding C data types." There's no official compatibility between them. Note that values in square braces '[ ]' are the defaults.

Also note that COBOL has a large collection of possible data types when compared to the C language. COBOL has fixed size character fields (PIC X) and numeric fields can vary in size, sign, data format (binary, packed, display, float) and vary in decimal places as well as numeric edited fields.

GnuCOBOL Main Programs CALLing C Subprograms

Here's a sample of a GnuCOBOL program that CALLs a C and a COBOL subprogram.

COBOL Calling Program

```
======================
     IDENTIFICATION DIVISION.
     PROGRAM-ID. maincob.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01 Arg1 PIC X(7).
     01 Arg2 PIC X(7).
     01 Big2 PIC X(20) VALUE "Hello World".
     01 Arg3 USAGE BINARY-INT.
     PROCEDURE DIVISION.
     000-Main.
         DISPLAY 'Starting maincob'
         CALL "sleep" USING BY VALUE 5.
```

```
MOVE Z'Arg1' TO Arg1
MOVE Z'Arg2' TO Arg2
MOVE 123456789 TO Arg3
CALL 'subc' USING BY CONTENT Arg1,
                  BY REFERENCE Arg2,
                  BY REFERENCE Arg3
DISPLAY 'Back in COBOL'
DISPLAY 'Arg1=' Arg1 ',' 'Arg2=' Arg2 ',' 'Arg3=' Arg3
DISPLAY 'Returned value=' RETURN-CODE

DISPLAY 'Call "subc" with normal COBOL data'
MOVE 'Arg1' TO Arg1
MOVE 'Arg2' TO Arg2
MOVE 31415926   TO Arg3
DISPLAY 'Arg1=' Arg1 ',' 'Arg2=' Arg2 ',' 'Arg3=' Arg3
CALL 'subc' USING BY CONTENT Arg1,
                  BY REFERENCE Arg2,
                  BY REFERENCE Arg3
DISPLAY 'Back in COBOL'
DISPLAY 'Arg1=' Arg1 ',' 'Arg2=' Arg2 ',' 'Arg3=' Arg3
DISPLAY 'Returned value=' RETURN-CODE

DISPLAY 'Call "subcob" with normal COBOL data'
MOVE 'Arg1' TO Arg1
MOVE 31415926   TO Arg3
CALL 'subcob' USING BY CONTENT Arg1,
                  BY REFERENCE Big2,
                  BY REFERENCE Arg3
DISPLAY 'Back in COBOL main'
DISPLAY 'Arg1=' Arg1 ',' 'Big2=' Big2 ',' 'Arg3=' Arg3
DISPLAY 'Returned value=' RETURN-CODE

MOVE 31415926   TO Arg3
MOVE Z"Yellow Submarine" TO Big2.
CALL 'subvalc' USING Big2, BY VALUE LENGTH OF Arg3
STOP RUN RETURNING 0.
```

C subroutines

```
====================
```

```
#include <stdio.h>
int subc(char *arg1, char *arg2, unsigned int *arg3)
{
    char nu1[7]="New1";
    char nu2[7]="New2";
    printf("On entry to subc: ");
    printf("Arg1='%s', ",arg1);
    printf("Arg2='%s', ",arg2);
    printf("Arg3=%d\n",*arg3);
    arg1[0]='X';
    arg2[0]='Y';
    *arg3=987654321;
    printf("Return from subc: ");
    printf("Arg1='%s', ",arg1);
    printf("Arg2='%s', ",arg2);
    printf("Arg3=%d\n",*arg3);
    return 2;
}
int subvalc(char *arg1, unsigned int arg2)
{
    printf("On entry to subvalc: ");
    printf("Arg1='%s', ",arg1);
    printf("Arg2=%d\n",arg2);
    return 2;
}
int subfltc(char *arg1, double arg2)
{
    printf("On entry to subfltc: ");
    printf("Arg1='%s', ",arg1);
    printf("Arg2=%.4f\n",arg2);
    return 2;
}
```

COBOL subroutine

```
===============
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. subcob.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LN-ARG2 PIC 99.
LINKAGE SECTION.
01 Arg1 PIC X(7).
01 Arg2 PIC X ANY LENGTH.
01 Arg3 USAGE BINARY-INT.
PROCEDURE DIVISION USING BY VALUE Arg1, BY REFERENCE Arg2, Arg3.
000-Main.
    DISPLAY 'Starting cobsub.cbl'
    MOVE LENGTH OF ARG2 TO LN-ARG2.
    DISPLAY 'Arg1=' Arg1 '.'
    DISPLAY 'Arg2=' Arg2 '. Len:' LN-ARG2
    DISPLAY 'Arg3=' Arg3 '.'
    MOVE 'X' TO Arg1 (1:1)
    MOVE 'Y' TO Arg2 (1:1)
    MOVE 987654321 TO Arg3
    MOVE 2 TO RETURN-CODE
    GOBACK.
```

The idea is to pass two strings and one full-word unsigned arguments to the C subprogram, have the subprogram print them out, change all three and pass a return code of 2 back to the caller. The caller will then re-display the three arguments (showing changes only to the two "BY REFERENCE" arguments), display the return code and halt.

While simple, the COBOL main and C subroutine programs illustrate the techniques required quite nicely.

Note how the COBOL program ensures that a null end-of-string terminator is present on both string arguments. Since the C program is planning on making changes to all three arguments, it declares all three as pointers in the function header and references the third argument as a pointer in the function body. It actually had no choice for the two string (char array) arguments – they must be defined as pointers in the function even though the function code references them without the leading * that normally signifies pointers. Note use of 'PIC X ANY LENGTH' in the COBOL subroutine.

These programs are compiled and executed as follows.

```
$ cobc -x -o maincob maincob.cbl subc.c subcob.cbl
$ maincob
Starting maincob
On entry to subc: Arg1='Arg1', Arg2='Arg2', Arg3=123456789
Return from subc: Arg1='Xrg1', Arg2='Yrg2', Arg3=987654321
Back in COBOL
Arg1=Arg1  ,Arg2=Yrg2  ,Arg3=+0987654321
Returned value=+000000002
Call "subc" with normal COBOL data
Arg1=Arg1   ,Arg2=Arg2   ,Arg3=+0031415926
On entry to subc: Arg1='Arg1   ', Arg2='Arg2   ', Arg3=31415926
Return from subc: Arg1='Xrg1   ', Arg2='Yrg2   ', Arg3=987654321
Back in COBOL
Arg1=Arg1   ,Arg2=Yrg2   ,Arg3=+0987654321
Returned value=+000000002
Call "subcob" with normal COBOL data
Starting cobsub.cbl
Arg1=Arg1   .
Arg2=Hello World         . Len:20
Arg3=+0031415926.
Back in COBOL main
Arg1=Arg1   ,Big2=Yello World         ,Arg3=+0987654321
Returned value=+000000002
On entry to subvalc: Arg1='Yellow Submarine', Arg2=4
$
```

Remember that the null characters are actually in the GnuCOBOL "Arg1" and "Arg2" data items. They don't appear in the output, but they ARE there. Did you notice the output showing the contents of "Arg1" after the subroutine was called? Those contents were unchanged! The subroutine definitely changed that argument, but since the COBOL program passed that argument "BY CONTENT", the change was made to a copy of the argument, not to the "Arg1" data item itself.

In the COBOL subroutine 'PIC X ANY LENGTH' means that the length of field is taken from the COBOL calling routine. This feature is unique to COBOL-to-COBOL interfaces. For the C routine `subvalc`, it is passed a numeric field BY VALUE, so the C code defines that as a simple 'unsigned int'.

C Main Programs Calling GnuCOBOL Subprograms

Now, the roles of the two languages in the previous section will be reversed, having a C main program execute a GnuCOBOL subprogram. There are several ways for C to call COBOL passing parameters.

The simple way is to static link the C and COBOL modules together and then the C code can just call the COBOL module by name. The first example below is passing all three parameters. The second invocation of 'subcob' is passing a NULL as the third parameter and the COBOL module test for that with 'IF ARG3 OMITTED' since this is supposed to be a pointer.

For C to call a COBOL module which may need to be dynamically loaded use the cob_call_cobol function:

```
int      cob_call_cobol  (const char *name, const int argc, ...);
```

name	is a string holding the COBOL module's name
argc	is the number of parameters being passed
…	parameters as expected by the COBOL module

For C to call a COBOL module when the C code has the address of the COBOL module use the cob_call_entry function:

```
int      cob_call_entry  (void *addr, const int argc, ...);
```

addr	as the address of the COBOL routine to be called
argc	is the number of parameters being passed
…	parameters as expected by the COBOL module

C (main) Calling Program

==========

```
#include <stdio.h>
#include <string.h>
#include <libcob.h> /* COB RUN-TIME */
extern int subcob (char *, char *, unsigned int *);
int main (int argc, char **argv)
{
     int returnCode;
     char arg1[20] = "Arg1";
     char arg2[20] = "Arg2";
     unsigned int arg3 = 123456789;
     printf("Starting mainc...calling COBOL\n");
     cob_init (argc, argv); /* COB RUN-TIME */
     returnCode = subcob(arg1,arg2,&arg3);
     printf("Back from COBOL\n");
     printf("Arg1='%s', ",arg1);
     printf("Arg2='%s', ",arg2);
     printf("Arg3=%d\n",arg3);
     printf("Returned value=%d\n",returnCode);

     strcpy(arg2,"Bigger Arg");
     returnCode = subcob(arg1,arg2,NULL);
     printf("Back from COBOL\n");
     printf("Arg1='%s', ",arg1);
     printf("Arg2='%s'\n",arg2);

     strcpy(arg2,"Hello World");
     arg3 = 123456789;
     returnCode = cob_call_cobol ("subcob",2,arg1,arg2,NULL);
     printf("Back from COBOL via cob_call_cobol\n");
     printf("Arg1='%s', ",arg1);
     printf("Arg2='%s'\n",arg2);

     strcpy(arg2,"Confucius Says");
     arg3 = 123456789;
```

```
    returnCode = cob_call_entry ((void*)subcob,3,arg1,arg2,&arg3);
    printf("Back from COBOL via cob_call_entry\n");
    printf("Arg1='%s', ",arg1);
    printf("Arg2='%s', ",arg2);
    printf("Arg3=%d\n",arg3);
    return 0;
}
```

COBOL subroutine

```
==========
    IDENTIFICATION DIVISION.
    PROGRAM-ID. subcob.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01 LN-ARG2 PIC 99.
    LINKAGE SECTION.
    01 Arg1 PIC X(7).
    01 Arg2 PIC X ANY LENGTH.
    01 Arg3 USAGE BINARY-INT.
    PROCEDURE DIVISION USING BY VALUE Arg1, BY REFERENCE Arg2, Arg3.
    000-Main.
    DISPLAY 'Starting cobsub.cbl'
    MOVE LENGTH OF ARG2 TO LN-ARG2.
    DISPLAY 'Arg1=' Arg1 '.'
    DISPLAY 'Arg2=' Arg2 '. Len:' LN-ARG2
    IF ARG3 OMITTED
      DISPLAY 'Arg3 was OMMITTED.'
      MOVE 'Z' TO Arg2 (1:1)
    ELSE
      DISPLAY 'Arg3=' Arg3 '.'
      MOVE 987654321 TO Arg3
      MOVE 'Y' TO Arg2 (1:1)
    END-IF.
    MOVE 'X' TO Arg1 (1:1)
    MOVE 2 TO RETURN-CODE
    GOBACK.
```

Since the C program is the one that will execute first, before the GnuCOBOL subroutine, the burden of initializing the GnuCOBOL run-time environment lies with that C program; it will have to invoke the "cob_init" function, which is part of the "libcob" library. The two required C statements are shown highlighted.

The arguments to the "cob_init" routine are the argument count and value parameters passed to the main function when the program began execution. By passing them into the GnuCOBOL subprogram, it will be possible for that GnuCOBOL program to retrieve the command line or individual command-line arguments. If that won't be necessary, "`cob_init(0,NULL);`" could be specified instead.

Since the C program wants to allow "arg3" to be changed by the subprogram, it prefixes it with a "&" to force a CALL BY REFERENCE for that argument. Since "arg1" and "arg2" are strings (char arrays), they are automatically passed by reference. Here's the output of the compilation process as well as the program's execution.

```
> cobc -fstatic-call -x -g -o cmain cmain.c subcob.cbl
> cmain
Starting mainc...calling COBOL
Starting cobsub.cbl
Arg1=Arg1.
Arg2=Arg2. Len:04
Arg3=+0123456789.
Back from COBOL
Arg1='Xrg1', Arg2='Yrg2', Arg3=987654321
Returned value=2
Starting cobsub.cbl
Arg1=Xrg1.
Arg2=Bigger Arg. Len:10
Arg3 was OMMITTED.
Back from COBOL
Arg1='Xrg1', Arg2='Zigger Arg'
Starting cobsub.cbl
Arg1=Xrg1.
Arg2=Hello World. Len:11
Arg3 was OMMITTED.
Back from COBOL via cob_call_cobol
Arg1='Xrg1', Arg2='Zello World'
Starting cobsub.cbl
Arg1=Xrg1.
Arg2=Confucius Says. Len:14
Arg3=+0123456789.
Back from COBOL via cob_call_entry
Arg1='Xrg1', Arg2='Yonfucius Says', Arg3=987654321
```

Note that even though we told GnuCOBOL that the 1st argument was to be "BY VALUE", it was treated as if it were "BY REFERENCE" anyway. String (char array) arguments passed from C callers to GnuCOBOL subprograms will be modifiable by the subprogram. It's best to pass a copy of such data if you want to ensure that the subprogram doesn't change it. The third argument is different, however. Since it's not an array you have the choice of passing it either "BY REFERENCE" or "BY VALUE".

Also, since Arg2 is defined as PIC X ANY LENGTH and even when the caller is C, the entry code to the 'subcob' knows that it has been called by a C module and internally uses 'strlen' to figure out how long the string is so that 'Arg2' will be processed by COBOL as a PIC X(n) where 'n' is 'strlen' of the parameter passed.

COBOL to COBOL and ANY NUMERIC

A feature supported by GnuCOBOL is that a LINKAGE SECTION parameter can be defined as PIC 9 ANY NUMERIC and the entry code of the subroutine will adjust the parameter as viewed by the subroutine to match that of the calling COBOL module.

COBOL main module
==========

```
            IDENTIFICATION DIVISION.
            PROGRAM-ID. mainnum.
            DATA DIVISION.
            WORKING-STORAGE SECTION.
            01 PACK-NUM  PIC 99V9999 COMP-3.
            01 COMP-NUM  PIC 999V99  COMP-4.
            01 COMP5-NUM PIC 9V99    COMP-5.
            01 COMPX-NUM PIC XXX     COMP-X.
            01 FLT-NUM   COMP-1.
            01 NE-NUM    PIC Z9.999.
            01 Char-7    PIC X(7)  VALUE "Fubar".
            01 Char-20   PIC X(20) VALUE "Hello World".
            PROCEDURE DIVISION.
            000-Main.
                DISPLAY 'Starting mainnum'
                MOVE 3.1415926 TO PACK-NUM COMP-NUM FLT-NUM
                MOVE 3.1415926 TO COMP5-NUM NE-NUM
                MOVE 31415     TO COMPX-NUM
                CALL 'subnum' USING PACK-NUM, Char-7.
                CALL 'subnum' USING COMP-NUM, Char-20.
                CALL 'subnum' USING FLT-NUM, "FLOAT/COMP-1".
                CALL 'subnum' USING COMP5-NUM, "COMP-5".
                CALL 'subnum' USING COMPX-NUM, "COMP-X"
                CALL 'subnum' USING NE-NUM, "Edited"
                CALL 'subfltc' USING "COMP-1", BY VALUE FLT-NUM
                STOP RUN RETURNING 0.
```

COBOL subroutine
==========

```
           IDENTIFICATION DIVISION.
           PROGRAM-ID. subnum.
           DATA DIVISION.
           WORKING-STORAGE SECTION.
           01 LN-ARG2 PIC 99.
           LINKAGE SECTION.
           01 Arg1 PIC 9 ANY NUMERIC.
           01 Arg2 PIC X ANY LENGTH.
           PROCEDURE DIVISION USING  Arg1, arg2.
           000-Main.
               DISPLAY 'In cobnum.cbl with ' Arg1 ' & "' Arg2 '"'.
               EXIT PROGRAM.
```

The programs are compiled and executed as follows

```
>cobc -x mainnum.cbl subnum.cbl subc.c
>./mainnum
Starting mainnum
In cobnum.cbl with 03.1415 & "Fubar  "
In cobnum.cbl with 003.14 & "Hello World         "
In cobnum.cbl with 3.1415925 & "FLOAT/COMP-1"
In cobnum.cbl with 00314 & "COMP-5"
In cobnum.cbl with 00031415 & "COMP-X"
In cobnum.cbl with  3.141 & "Edited"
On entry to subfltc: Arg1='COMP-1', Arg2=3.1416
```

Note that Arg1 gets displayed as you would expect of the field being passed on the call is defined. Since ‘arg2’ is PIC X ANY LENGTH it is also assigned the length of the parameter that was passed on the CALL.

The CALL “subfltc” is calling a C subroutine and passing the COMP-1 variable by VALUE so the C subroutine declares the variable as type ‘double’.

COBOL CALL parameters

When COBOL does a CALL the parameter field descriptions are stored in an internal data structure. The parameters are then passed as address of the field data for BY REFERENCE or contents of the field for BY VALUE. This is done so that a C routine gets the parameters as normal C parameters on the execution stack. Application C modules may want to exchange data with some independence of the data format.

So, these routines can be used to get parameters from the most recent COBOL CALL statement. '`num_param`' is 1 relative, so the first parameter is 1.

```
int       cob_get_num_params ();
int       cob_get_param_type ( int num_param ) // type values from common.h
int       cob_get_param_size ( int num_param ) // size in bytes
int       cob_get_param_digits( int num_param )// number of digits in field
int       cob_get_param_scale( int num_param ) // numeric scale value
int       cob_get_param_sign ( int num_param ) // 1 for signed, else 0
int       cob_get_param_constant ( int num_param ) // 1 if constant, else 0
void *    cob_get_param_data ( int num_param ) // address of field data
cob_s64_t cob_get_s64_param  ( int num_param ) // signed numeric value
cob_u64_t cob_get_u64_param  ( int num_param ) // unsigned numeric value

char *    cob_get_picx_param ( int num_param, void *charfld, size_t charlen)

void      cob_put_s64_param  ( int num_param, cob_s64_t value )
void      cob_put_u64_param  ( int num_param, cob_u64_t value )
void      cob_put_picx_param ( int num_param, void *charfld )
void *    cob_get_grp_param  ( int num_param, void *charfld, size_t charlen);
void      cob_put_grp_param  ( int num_param, void *charfld, size_t charlen);
```

cob_get_picx_param	returns string holding contents of field. If 'charfld' is NULL, then memory is allocated and the caller must release it later using cob_free ((void*string). If 'charfld' is not NULL, then charlen indicates the size of the field to receive the characters from the parameter. The data will have trailing spaces removed and be NUL terminated.
cob_put_s64_param	Store the signed 'value' into the field passed as parameter
cob_put_u64_param	Store the unsigned 'value' into the field passed as parameter
cob_get_grp_param	Does an exact copy of the parameter data to 'charfld'
cob_put_grp_param	Does an exact copy from 'charfld' to the data field
cob_get_param_type	May return one of the following values (defined via libcob.h) COB_TYPE_GROUP COB_TYPE_NUMERIC_DISPLAY COB_TYPE_NUMERIC_BINARY COB_TYPE_NUMERIC_PACKED COB_TYPE_NUMERIC_FLOAT COB_TYPE_NUMERIC_DOUBLE COB_TYPE_NUMERIC_COMP5 COB_TYPE_NUMERIC_EDITED COB_TYPE_ALPHANUMERIC
cob_get_param_digits	Returns the number of digits in the field. For PIC 9(5) it returns 5
cob_get_param_scale	Returns decimal places. For PIC 9(5)V99 the scale is 2 and digits is 7
cob_get_grp_param	Does an exact copy of the field data into 'charfld'
cob_put_grp_param	Does an exact copy of 'charfld' into the field data area.

If 'num_param' is out of range, there will be a run-time warning message displayed.

The C programmer is responsible for making sure the value is suitable considering COBOL field digits and scale all of which can be retrieved.

Complex example of GnuCOBOL C API

This example is rather large and more complex, but read through it carefully. The COBOL program makes many calls to a C subroutine which inspects the calling parameters and reports what it finds. This is likely more complex than you would normally need but it is here as an example what is possible if there is need. (Many of the functions have a #define with the Microfocus routine name for compatibility.)

C Subroutine

==========

```
#include <stdio.h>
#include <string.h>
#include <libcob.h>

static char *
getType(int type, int byvalue)
{
    static char wrk[24];
    switch (type) {
    case COB_TYPE_GROUP:            return "Group";
    case COB_TYPE_ALPHANUMERIC:     return "X";
    case COB_TYPE_NUMERIC_BINARY:   return "COMP-4";
    case COB_TYPE_NUMERIC_COMP5:    return byvalue==2?"COMP-4":"COMP-5";
    case COB_TYPE_NUMERIC_DISPLAY:  return "DISPLAY";
    case COB_TYPE_NUMERIC_DOUBLE:   return "COMP-2";
    case COB_TYPE_NUMERIC_EDITED:   return "EDITED";
    case COB_TYPE_NUMERIC_FLOAT:    return "COMP-1";
    case COB_TYPE_NUMERIC_PACKED:   return "COMP-3";
    }
    sprintf(wrk,"Type %04X",type);
    return wrk;
}
int
CAPI(void *p1, ...)
{
    int         k,nargs,type,digits,scale,size,sign,byvalue;
    cob_s64_t   val;
    char        *str, wrk[80],pic[24];
    void        *cbldata;
    double      dval;
    nargs = cob_get_num_params();
    if ((k = cob_get_name_line (wrk, NULL)) > 0) {
        printf("%s Line%3d: ",wrk,k);
    }
    printf("CALL with %d parameters\n",nargs);
    for(k=1; k <= nargs; k++) {
        type   = cob_get_param_type (k);
        digits = cob_get_param_digits (k);
        scale  = cob_get_param_scale (k);
        size   = cob_get_param_size (k);
        sign   = cob_get_param_sign (k);
        byvalue = cob_get_param_constant(k);
        cbldata = cob_get_param_data (k);
        printf("     P%d: %-8s ",k,getType(type,byvalue));
        if (byvalue == 3)
            printf("BY CONTENT   ");
        else if (byvalue == 2)
            printf("BY VALUE     ");
        else if (byvalue == 1)
            printf("LITERAL      ");
        else
            printf("BY REFERENCE ");
        if (type == COB_TYPE_ALPHANUMERIC) {
            sprintf(pic,"X(%d)",size);
            str = cob_get_picx_param (k, NULL, 0);
            printf("%-11s '%s'",pic,str);
```

```
            cob_free ((void*)str);
            if (byvalue != 3 && byvalue != 1)
                cob_put_picx_param (k, "Bye!");
        } else if (type == COB_TYPE_GROUP) {
            sprintf(pic,"(%d)",size);
            str = cob_get_grp_param (k, NULL, 0);
            printf("%-11s '%.*s'",pic,size,str);
            cob_free ((void*)str);
            memset(wrk,' ',sizeof(wrk));
            memcpy(wrk,"Bye-Bye Birdie!",15);
            cob_put_grp_param (k, wrk, sizeof(wrk));
            str = cob_get_grp_param (k, NULL, 0);
            printf(" --> '%.*s'",size,str);
            cob_free ((void*)str);
        } else if (type == COB_TYPE_NUMERIC_EDITED) {
            if(scale > 0) {
                sprintf(pic,"%s9(%d)V9(%d)",sign?"S":"",digits-scale,scale);
            } else {
                sprintf(pic,"%s9(%d)",sign?"S":"",digits-scale);
            }
            val = cob_get_s64_param (k);
            printf("%-11s %lld ",pic,val);
            val = val + 130;
            val = -val;
            if (byvalue != 3 && byvalue != 1)
                cob_put_s64_param (k, val);
            cob_get_grp_param (k, wrk, sizeof(wrk));
            printf(" to %.*s",size,wrk);
        } else if (type == COB_TYPE_NUMERIC_FLOAT) {
            dval = (double)cob_get_dbl_param (k);
            printf("%-11s %.6f","COMP-1",(double)dval);
            cob_put_comp1 ((float)(dval + 1.5), cbldata);
        } else if (type == COB_TYPE_NUMERIC_DOUBLE) {
            dval = (double)cob_get_dbl_param (k);
            printf("%-11s %.8f","COMP-2",(double)dval);
            cob_put_dbl_param (k, (double)(dval + 1.5));
        } else {
            if(scale > 0) {
                sprintf(pic,"%s9(%d)V9(%d)",sign?"S":"",digits-scale,scale);
            } else {
                sprintf(pic,"%s9(%d)",sign?"S":"",digits-scale);
            }
            val = cob_get_s64_param (k);
            printf("%-11s %lld",pic,val);
            if (type == COB_TYPE_NUMERIC_PACKED) {
                dval = (double)cob_get_dbl_param (k);
                printf(" double[%.4f]",dval);
            }
            if (byvalue != 3 && byvalue != 1)
                cob_put_s64_param (k, val + 3);
        }
        printf(";\n");
        fflush(stdout);
    }
    if (nargs > 2
     && byvalue != 3 && byvalue != 1)
        cob_put_s64_param (7, val + 3);
    return 0;
}
```

COBOL Main Program

==========

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. maincapi.
000003*
000004 DATA DIVISION.
```

```
000005 WORKING-STORAGE SECTION.
000006 01  BIN5FLD     PIC  9(5) COMP-5  VALUE  5555.
000007 01  BINFLD5S    PIC S9(5) BINARY  VALUE  4444.
000008 01  BINFLD9     PIC  9(9) BINARY  VALUE  6666.
000009 01  COMP3       PIC  9(8) COMP-3  VALUE  3333.
000010 01  COMP3V99    PIC S9(7)V99 COMP-3  VALUE  12.50.
000011 01  PIC9        PIC S9(8) DISPLAY VALUE  8888.
000012 01  NE          PIC Z(4)9.99-.
000013 01  FLT-NUM     COMP-1.
000014 01  DBL-NUM     COMP-2.
000015 01  CHRX        PIC  X(9)         VALUE 'Hello'.
000016 01  GRPX.
000017     05  FILLER  PIC  X(9)         VALUE 'Hello'.
000018     05  FILLER  PIC  X(9)         VALUE 'World'.
000019 01  MYOCC        PIC 9(8) COMP.
000020 01  MYTAB.
000021     03  MYBYTE   PIC XX OCCURS 1 TO 20
000022                  DEPENDING ON MYOCC.
000023*
000024 PROCEDURE DIVISION.
000025     MOVE -512.77 TO NE.
000026     CALL "CAPI" USING BY CONTENT
000027                       FUNCTION CONCATENATE("ABC" "DEF").
000028     CALL "CAPI" USING 2560 BY VALUE 16.
000029     CALL "CAPI" USING BIN5FLD, NE.
000030     CALL "CAPI" USING BINFLD5S.
000031     CALL "CAPI" USING BINFLD9.
000032     MOVE 3.1415926 TO FLT-NUM, DBL-NUM
000033     CALL "CAPI" USING FLT-NUM, DBL-NUM.
000034     DISPLAY "Float: " FLT-NUM "; Double:" DBL-NUM.
000035     CALL "CAPI" USING BY CONTENT BIN5FLD, NE.
000036     CALL "CAPI" USING BY CONTENT BIN5FLD, NE.
000037     CALL "CAPI" USING BY CONTENT BINFLD5S.
000038     CALL "CAPI" USING BY CONTENT BINFLD5S.
000039     CALL "CAPI" USING BY CONTENT BINFLD9.
000040     CALL "CAPI" USING BY CONTENT BINFLD9.
000041     CALL "CAPI" USING BY VALUE BIN5FLD, NE.
000042     CALL "CAPI" USING BY VALUE BIN5FLD, NE.
000043     CALL "CAPI" USING BY VALUE BINFLD5S.
000044     CALL "CAPI" USING BY VALUE BINFLD5S.
000045     CALL "CAPI" USING BY VALUE BINFLD9.
000046     CALL "CAPI" USING BY VALUE BINFLD9.
000047     MOVE  512.77 TO NE.
000048     CALL "CAPI" USING COMP3, NE.
000049     DISPLAY "GRPX     was    " GRPX ";".
000050     CALL "CAPI" USING PIC9 BINFLD5S CHRX GRPX.
000051     DISPLAY "GRPX     is now " GRPX ";".
000052     CALL "CAPI" USING COMP3, NE, CHRX.
000053     CALL "CAPI" USING BIN5FLD, NE.
000054     MOVE "Hello!" TO CHRX.
000055     DISPLAY "BIN5FLD BY VALUE & " CHRX ";".
000056     CALL "CAPI" USING BY VALUE BIN5FLD, CHRX.
000057     CALL "CAPI" USING BY VALUE BIN5FLD, CHRX.
000058     CALL "CAPI" USING LENGTH OF GRPX.
000059     MOVE "Anyone out there?" TO GRPX.
000060     DISPLAY "GRPX     was    " GRPX ";".
000061     CALL "CAPI" USING BY VALUE GRPX LENGTH OF GRPX.
000062     DISPLAY "GRPX     is now " GRPX ";".
000063     CALL "CAPI" USING "Fred Fish", COMP3.
000064     CALL "CAPI" USING COMP3V99.
000065     CALL "CAPI" .
000066     DISPLAY "COMP3    is now " COMP3 ";".
000067     DISPLAY "COMP4    is now " BIN5FLD ";".
000068     DISPLAY "BINFLD5S is now " BINFLD5S ";".
000069     DISPLAY "CHRX     is now " CHRX ";".
000070     DISPLAY "NE       is now " NE ";".
000071     CALL "CAPI" USING BY CONTENT 1.
```

```
000072     CALL "CAPI" USING BY VALUE 1.
000073     CALL "CAPI" USING BY REFERENCE 1.
000074     MOVE 9 TO MYOCC.
000075     DISPLAY "Now BY CONTENT LENGTH OF MYTAB;".
000076     CALL "CAPI" USING BY CONTENT LENGTH OF MYTAB.
000077     DISPLAY "Now BY CONTENT LENGTH OF MYOCC;".
000078     CALL "CAPI" USING BY CONTENT LENGTH OF MYOCC.
000079     MOVE 7 TO MYOCC.
000080     DISPLAY "Now LENGTH OF MYTAB;".
000081     CALL "CAPI" USING LENGTH OF MYTAB.
000082     DISPLAY "Now LENGTH OF MYOCC;".
000083     CALL "CAPI" USING LENGTH OF MYOCC.
000084     MOVE 5 TO MYOCC.
000085     DISPLAY "Now BY VALUE LENGTH OF MYTAB;".
000086     CALL "CAPI" USING BY VALUE LENGTH OF MYTAB.
000087     DISPLAY "Now BY VALUE LENGTH OF MYOCC;".
000088     CALL "CAPI" USING BY VALUE LENGTH OF MYOCC.
000089     STOP RUN.
```

Compile command

==========

```
cobc -debug -fstatic-call -w -x maincapi.cbl capi.c
```

Results from running maincapi

==========

```
maincapi Line 26: CALL with 1 parameters
     P1: X        BY REFERENCE X(6)        'ABCDEF';
maincapi Line 28: CALL with 2 parameters
     P1: COMP-4   LITERAL      S9(9)       2560;
maincapi.cbl:28: warning: cob_put_s64_param: attempt to over-write constant
parameter 2 with 19
     P2: DISPLAY  BY VALUE     9(2)        16;
maincapi Line 29: CALL with 2 parameters
     P1: COMP-5   BY REFERENCE 9(5)        5555;
     P2: EDITED   BY REFERENCE S9(5)V9(2)  -51277  to   511.47 ;
maincapi Line 30: CALL with 1 parameters
     P1: COMP-4   BY REFERENCE S9(5)       4444;
maincapi Line 31: CALL with 1 parameters
     P1: COMP-4   BY REFERENCE 9(9)        6666;
maincapi Line 33: CALL with 2 parameters
     P1: COMP-1   BY REFERENCE COMP-1      3.141593;
     P2: COMP-2   BY REFERENCE COMP-2      3.14159260;
Float: 4.6415925; Double:4.641592599999999
maincapi Line 35: CALL with 2 parameters
     P1: COMP-5   BY CONTENT   9(5)        5558;
     P2: EDITED   BY CONTENT   S9(5)V9(2)  51147  to   511.47 ;
maincapi Line 36: CALL with 2 parameters
     P1: COMP-5   BY CONTENT   9(5)        5558;
     P2: EDITED   BY CONTENT   S9(5)V9(2)  51147  to   511.47 ;
maincapi Line 37: CALL with 1 parameters
     P1: COMP-4   BY CONTENT   S9(5)       4447;
maincapi Line 38: CALL with 1 parameters
     P1: COMP-4   BY CONTENT   S9(5)       4447;
maincapi Line 39: CALL with 1 parameters
     P1: COMP-4   BY CONTENT   9(9)        6669;
maincapi Line 40: CALL with 1 parameters
     P1: COMP-4   BY CONTENT   9(9)        6669;
maincapi Line 41: CALL with 2 parameters
     P1: COMP-4   BY VALUE     9(5)        5558;
     P2: EDITED   BY CONTENT   S9(5)V9(2)  51147  to   511.47 ;
maincapi Line 42: CALL with 2 parameters
     P1: COMP-4   BY VALUE     9(5)        5558;
     P2: EDITED   BY CONTENT   S9(5)V9(2)  51147  to   511.47 ;
maincapi Line 43: CALL with 1 parameters
     P1: COMP-4   BY VALUE     S9(5)       4447;
```

```
maincapi Line 44: CALL with 1 parameters
     P1: COMP-4   BY VALUE     S9(5)       4447;
maincapi Line 45: CALL with 1 parameters
     P1: COMP-4   BY VALUE     9(9)        6669;
maincapi Line 46: CALL with 1 parameters
     P1: COMP-4   BY VALUE     9(9)        6669;
maincapi Line 48: CALL with 2 parameters
     P1: COMP-3   BY REFERENCE 9(8)        3333 double[3333.0000];
     P2: EDITED   BY REFERENCE S9(5)V9(2)  51277  to   514.07-;
GRPX     was    Hello    World    ;
maincapi Line 50: CALL with 4 parameters
     P1: DISPLAY  BY REFERENCE S9(8)       8888;
     P2: COMP-4   BY REFERENCE S9(5)       4447;
     P3: X        BY REFERENCE X(9)        'Hello';
     P4: Group    BY REFERENCE (18)        'Hello    World    ' --> 'Bye-Bye
Birdie!   ';
maincapi.cbl:50: warning: cob_put_s64_param: parameter 7 is not within range of 4
GRPX     is now Bye-Bye Birdie!   ;
maincapi Line 52: CALL with 3 parameters
     P1: COMP-3   BY REFERENCE 9(8)        3336 double[3336.0000];
     P2: EDITED   BY REFERENCE S9(5)V9(2)  -51407  to   512.77 ;
     P3: X        BY REFERENCE X(9)        'Bye!';
libcob: maincapi.cbl:52: warning: cob_put_s64_param: parameter 7 is not within
range of 3
maincapi Line 53: CALL with 2 parameters
     P1: COMP-5   BY REFERENCE 9(5)        5558;
     P2: EDITED   BY REFERENCE S9(5)V9(2)  51277  to   514.07-;
BIN5FLD BY VALUE & Hello!   ;
maincapi Line 56: CALL with 2 parameters
     P1: COMP-4   BY VALUE     9(5)        5561;
     P2: X        BY CONTENT   X(9)        'Hello!';
maincapi Line 57: CALL with 2 parameters
     P1: COMP-4   BY VALUE     9(5)        5561;
     P2: X        BY CONTENT   X(9)        'Hello!';
maincapi Line 58: CALL with 1 parameters
     P1: COMP-4   LITERAL      S9(9)       18;
GRPX     was    Anyone out there? ;
maincapi Line 61: CALL with 2 parameters
     P1: Group    BY CONTENT   (18)        'Anyone out there? ' --> 'Bye-Bye
Birdie!   ';
maincapi.cbl:61: warning: cob_put_s64_param: attempt to over-write constant
parameter 2 with 21
     P2: DISPLAY  BY VALUE     9(2)        18;
GRPX     is now Anyone out there? ;
maincapi Line 63: CALL with 2 parameters
     P1: X        BY CONTENT   X(9)        'Fred Fish';
     P2: COMP-3   BY REFERENCE 9(8)        3339 double[3339.0000];
maincapi Line 64: CALL with 1 parameters
     P1: COMP-3   BY REFERENCE S9(7)V9(2)  1250 double[12.5000];
maincapi Line 65: CALL with 0 parameters
COMP3    is now 00003342;
COMP4    is now 0000005561;
BINFLD5S is now +04450;
CHRX     is now Hello!   ;
NE       is now   514.07-;
maincapi Line 71: CALL with 1 parameters
     P1: COMP-4   LITERAL      S9(9)       1;
maincapi.cbl:72: warning: cob_put_s64_param: attempt to over-write constant
parameter 1 with 4
maincapi Line 72: CALL with 1 parameters
     P1: DISPLAY  BY VALUE     9(1)        1;
maincapi Line 73: CALL with 1 parameters
     P1: COMP-4   LITERAL      S9(9)       1;
Now BY CONTENT LENGTH OF MYTAB;
maincapi Line 76: CALL with 1 parameters
     P1: COMP-4   LITERAL      9(9)        18;
Now BY CONTENT LENGTH OF MYOCC;
```

```
maincapi Line 78: CALL with 1 parameters
     P1: COMP-4   LITERAL      S9(9)       4;
Now LENGTH OF MYTAB;
maincapi Line 81: CALL with 1 parameters
     P1: COMP-4   LITERAL      9(9)        14;
Now LENGTH OF MYOCC;
maincapi Line 83: CALL with 1 parameters
     P1: COMP-4   LITERAL      S9(9)       4;
Now BY VALUE LENGTH OF MYTAB;
maincapi Line 86: CALL with 1 parameters
     P1: COMP-4   BY VALUE     9(9)        10;
Now BY VALUE LENGTH OF MYOCC;
maincapi.cbl:88: warning: cob_put_s64_param: attempt to over-write constant
parameter 1 with 7
maincapi Line 88: CALL with 1 parameters
     P1: DISPLAY  BY VALUE     9(1)        4;
```

BINARY fields

BINARY fields declared as COMP, COMP-4 or BINARY and are stored in big-endian format by COBOL even if the machine is native little-endian. (GnuCOBOL does have an option to change this but use of that is discouraged because most, if not all, COBOL compilers default to BIG-ENDIAN binary values.) A field of type PIC X(n) COMP-X is n bytes (1 – 8) is used to store a binary value in big-endian format. Fields of type PIC 9(n) COMP-5 are binary values stored in the local native machine format but not necessarily aligned on the appropriate address boundary. C code typically deals with `short, int, long, long long` binary data, always aligned on appropriate address and it is always in the native machine format.

For handling binary fields, the following routines area available. These routines will handle up to 64-bit values. The 'comp5' routines handle the COBOL data field as native machine format (COMP-5, not necessarily aligned). The 'compx' routines handle the COBOL data field as normal COBOL big-endian format (COMP/COMP-4/BINARY/COMP-X).

```
cob_s64_t cob_get_s64_comp5 ( void *cbldata, int len )
cob_u64_t cob_get_u64_comp5 ( void *cbldata, int len )
void      cob_put_s64_comp5 ( cob_s64_t value, void *cbldata, int len )
void      cob_put_u64_comp5 ( cob_u64_t value, void *cbldata, int len )
cob_s64_t cob_get_s64_compx ( void *cbldata, int len )
cob_u64_t cob_get_u64_compx ( void *cbldata, int len )
void      cob_put_s64_compx ( cob_s64_t value, void *cbldata, int len )
void      cob_put_u64_compx ( cob_u64_t value, void *cbldata, int len )
```

The 'get' functions collect the data from the memory address and return a 64-bit value. 's64' indicates signed and 'u64' indicates unsigned.

The 'put' functions store the 64-bit value into the memory location for the given length.

cbldata	memory address of the COBOL data field
len	length of the COBOL data field in bytes
value	64-bit value, typedef cob_s64_t is signed, cob_u64_t is unsigned The C programmer is responsible for making sure the value is suitable considering COBOL field digits and scale.

PACKED Numeric Fields

COMP-3/PACKED-DECIMAL are stored like an IBM packed decimal numeric value. Each digit takes up one hexadecimal position (so 2 digits in 1 byte) with the last hex position being the sign. C is positive, D is negative and F is un-signed. The value is right justified with leading ZEROS. The application C code can use an integer value and be responsible for any needed decimal alignment issues.

```
cob_s64_t cob_get_s64_comp3 ( void *cbldata, int len )
cob_u64_t cob_get_u64_comp3 ( void *cbldata, int len )
void      cob_put_s64_comp3 ( cob_s64_t value, void *cbldata, int len )
void      cob_put_u64_comp3 ( cob_u64_t value, void *cbldata, int len )
```

The 'get' functions collect the data from the memory address and return a 64-bit value. 's64' indicates signed and 'u64' indicates unsigned.

The 'put' functions store the 64-bit value into the memory location for the given length.

cbldata	memory address of the COBOL data field
len	length of the COBOL data field in bytes
value	64 bit value, cob_s64_t is signed, cob_u64_t is unsigned

DISPLAY Numeric Fields

PIC S9 USAGE DISPLAY fields are stored like an IBM zoned decimal numeric value. Each digit takes up one byte (in ASCII), value is right justified with leading ZEROS. The last digit is used to indicate a sign. If the last digit has the 0x40 bit turned on then the value is negative, otherwise it is positive (or unsigned).

If EBCDIC signed is wanted then if the last digit is '0' thru '9' it is unsigned, '{', 'A' thru 'I' is positive, '}', 'J' thru 'R' is negative.

The following routines will be written and will use EBCDIC sign values if the COBOL module was compiled with the –fsign=EBCDIC.

```
cob_s64_t cob_get_s64_pic9 ( void *cbldata, int len )
cob_u64_t cob_get_u64_pic9 ( void *cbldata, int len )
void      cob_put_s64_pic9 ( cob_s64_t value, void *cbldata, int len )
void      cob_put_u64_pic9 ( cob_u64_t value, void *cbldata, int len )
```

The 'get' functions collect the data from the memory address and return a 64 bit value. 's64' indicates signed and 'u64' indicates unsigned.

The 'put' functions store the 64-bit value into the memory location for the given length.

cbldata	memory address of the COBOL data field
len	length of the COBOL data field in bytes
value	64 bit value, cob_s64_t is signed, cob_u64_t is unsigned

Alpha Fields

In COBOL PIC X fields are fixed size and padded out with SPACES. In C, NUL terminated strings are often wanted. Some routines will be written to convert between fixed size COBOL Fields and C strings.

If '`charfld`' is NULL, then memory will be malloc'd and should use `cob_free(string)` to release the memory. If '`charfld`' is not NULL, then '`charlen`' is the maximum length of the C field. The return value is the address of the C string.

```
char * cob_get_picx ( void *cbldata, size_t len, void *charfld, size_t charlen )
char * cob_put_picx ( void *cbldata, size_t len, void *string )
```

Floating point fields

COMP-1 is a native machine 'float' and COMP-2 is a native machine 'double'. But in COBOL the fields may not be aligned on a suitable address boundary for direct manipulation so there are some routines provided to get/put this data type.

```
float       cob_get_comp1 (void *cbldata);
double      cob_get_comp2 (void *cbldata);
void        cob_put_comp1 (float val, void *cbldata);
void        cob_put_comp2 (double val, void *cbldata);
```

C struct vs COBOL record

A C struct is similar to a COBOL record (i.e. 01 with sub-fields). GnuCOBOL has no utility that will convert between C struct and a COBOL record so you must do that by yourself. The size and alignment of data fields in COBOL depend on the field's USAGE but also compiler directives.

-fibmcomp	sets -fbinary-size=2-4-8 -fsynchronized-clause=ok
-fno-ibmcomp	sets -fbinary-size=1--8 -fsynchronized-clause=ignore
-fbinary-comp-1	COMP-1 is a 16-bit signed integer Normally COMP-1 is a 'float'

GnuCOBOL compile

The GnuCOBOL compiler accepts many command line options including `-L` to specific directory to search for libraries and `-l` to indicate specific libraries to process. You could compile either COBOL or C modules to an object module and then place the object module into an archive library which is later referenced by other programs at compile time.

This is the most practical way to build up a collection of subroutines that may be used by other parts of your application.

Micro Focus compatible functions

Micro Focus has a collection of subroutines callable from C code to get binary values from COBOL fields defined as COMP/COMP-4/BINARY/COMP-X and also COMP-5. See the Micro Focus documentation for details. A collection of `#defines` has been used to define the Micro Focus compatible functions as calls to the GnuCOBOL functions passing a 'len' value when needed. Following is a list of these #defines.

```
typedef char *      cobchar_t;
#define cobs8_t     cob_s8_t
#define cobuns8_t   cob_u8_t
#define cobs16_t    cob_s16_t
#define cobuns16_t  cob_u16_t
#define cobs32_t    cob_s32_t
#define cobuns32_t  cob_u32_t
#define cobs64_t    cob_s64_t
#define cobuns64_t  cob_u64_t

#define cobsetjmp(x)    setjmp (cob_savenv (x))
#define coblongjmp(x)   cob_longjmp (x)
#define cobsavenv(x)    cob_savenv (x)
#define cobsavenv2(x,z) cob_savenv2 (x, z)
#define cobfunc(x,y,z)  cob_func (x, y, z)
#define cobcall(x,y,z)  cob_call (x, y, z)
#define cobcancel(x)    cob_cancel (x)

#define cobgetenv(x)    cob_getenv (x)
#define cobputenv(x)    cob_putenv (x)
#define cobtidy()   cob_tidy ()
#define cobinit()   cob_extern_init ()
#define cobexit(x)  cob_stop_run (x)
#define cobcommandline(v,w,x,y,z)   cob_command_line (v,w,x,y,z)

#define cobclear()  (void) cob_sys_clear_screen ()
#define cobmove(y,x)    cob_set_cursor_pos (y, x)
#define cobcols()   cob_get_scr_cols ()
#define coblines()  cob_get_scr_lines ()
#define cobaddstrc(x)   cob_display_text (x)        /* no limit [MF=255] */
#define cobprintf   cob_display_formatted_text  /* limit of 2047 [MF=255] */
#define cobgetch()  cob_get_char ()

#define cobget_x1_compx(d)  (cobuns8_t)     cob_get_u64_compx(d, 1)
#define cobget_x2_compx(d)  (cobuns16_t)    cob_get_u64_compx(d, 2)
#define cobget_x4_compx(d)  (cobuns32_t)    cob_get_u64_compx(d, 4)
#define cobget_x8_compx(d)  (cobuns64_t)    cob_get_u64_compx(d, 8)
#define cobget_sx1_compx(d) (cobs8_t)   cob_get_s64_compx(d, 1)
#define cobget_sx2_compx(d) (cobs16_t)  cob_get_s64_compx(d, 2)
#define cobget_sx4_compx(d) (cobs32_t)  cob_get_s64_compx(d, 4)
#define cobget_sx8_compx(d) (cobs64_t)  cob_get_s64_compx(d, 8)
#define cobget_x1_comp5(d)  (cobuns8_t)     cob_get_u64_comp5(d, 1)
#define cobget_x2_comp5(d)  (cobuns16_t)    cob_get_u64_comp5(d, 2)
#define cobget_x4_comp5(d)  (cobuns32_t)    cob_get_u64_comp5(d, 4)
#define cobget_x8_comp5(d)  (cobuns64_t)    cob_get_u64_comp5(d, 8)
#define cobget_sx1_comp5(d) (cobs8_t)   cob_get_s64_comp5(d, 1)
#define cobget_sx2_comp5(d) (cobs16_t)  cob_get_s64_comp5(d, 2)
#define cobget_sx4_comp5(d) (cobs32_t)  cob_get_s64_comp5(d, 4)
#define cobget_sx8_comp5(d) (cobs64_t)  cob_get_s64_comp5(d, 8)
#define cobget_xn_comp5(d,n)    (cobuns64_t)    cob_get_u64_comp5(d, n)
#define cobget_xn_compx(d,n)    (cobuns64_t)    cob_get_u64_compx(d, n)
#define cobget_sxn_comp5(d,n)   (cobs64_t)  cob_get_s64_comp5(d, n)
#define cobget_sxn_compx(d,n)   (cobs64_t)  cob_get_s64_compx(d, n)

#define cobput_x1_compx(d,v)    (void)  cob_put_u64_compx((cob_u64_t)v,d,1)
#define cobput_x2_compx(d,v)    (void)  cob_put_u64_compx((cob_u64_t)v,d,2)
#define cobput_x4_compx(d,v)    (void)  cob_put_u64_compx((cob_u64_t)v,d,4)
#define cobput_x8_compx(d,v)    (void)  cob_put_u64_compx((cob_u64_t)v,d,8)
```

```
#define cobput_x1_comp5(d,v)     (void)  cob_put_u64_comp5((cob_u64_t)v,d,1)
#define cobput_x2_comp5(d,v)     (void)  cob_put_u64_comp5((cob_u64_t)v,d,2)
#define cobput_x4_comp5(d,v)     (void)  cob_put_u64_comp5((cob_u64_t)v,d,4)
#define cobput_x8_comp5(d,v)     (void)  cob_put_u64_comp5((cob_u64_t)v,d,8)
#define cobput_sx1_comp5(d,v)    (void)  cob_put_s64_comp5((cob_s64_t)v,d,1)
#define cobput_sx2_comp5(d,v)    (void)  cob_put_s64_comp5((cob_s64_t)v,d,2)
#define cobput_sx4_comp5(d,v)    (void)  cob_put_s64_comp5((cob_s64_t)v,d,4)
#define cobput_sx8_comp5(d,v)    (void)  cob_put_s64_comp5((cob_s64_t)v,d,8)
#define cobput_xn_comp5(d,n,v)   (void)  cob_put_u64_comp5(v, d, n)
#define cobput_xn_compx(d,n,v)   (void)  cob_put_u64_compx(v, d, n)
#define cobput_sxn_comp5(d,n,v)  (void)  cob_put_s64_comp5(v, d, n)
#define cobput_sxn_compx(d,n,v)  (void)  cob_put_s64_compx(v, d, n)
```

EXTFH – External File Interface

GnuCOBOL supports the EXTFH interface used by Microfocus and IBM COBOL compilers. GnuCOBOL specifically uses the FCD3 structure as defined by the COPY book called **`xfhfcd3.cpy`**. A C module written to use this EXTFH interface can be compiled and used by either GnuCOBOL or Microfocus Visual COBOL on the same platform.

The C header **`libcob.h`** includes **`common.h`** which has **`FCD3`** defined as a `typedef` plus all of the EXTFH operation codes.

Sample C module using EXTFH

```
==========================
    /*********************************************************************************
    *                                                                                *
    * For GnuCOBOL add  -fcallfh=TSTFH              as a compile option              *
    *                                                                                *
    * This is a sample module for GnuCOBOL, but it does not do very much             *
    *********************************************************************************/

    #include <string.h>
    #include <stdlib.h>
    #include <libcob.h>

    static char *txtOpCode(int opCode);

    /*********************************************************************
       Replace filename with environment variable value, then open the file
       This is required as MF Cobol seems to have pre-read the ENV Variables
    *********************************************************************/
    static int
    doOpenFile(
       unsigned char  *opCodep,
       FCD3  *fcd,
       char  *opmsg)
    {
       int      sts,oldlen,j,k;
       char  *oldFptr,*env,wrk[64];
       unsigned char  svOther;
       unsigned int   opCode;

       oldFptr = fcd->fnamePtr;            /* Save values */
       oldlen  = LDCOMPX2(fcd->fnameLen);
       fcd->otherFlags &= ~OTH_DOLSREAD;
       svOther = fcd->otherFlags;

       sts = EXTFH( opCodep, fcd );          /* No DD_, so use normal MF File Open */
       printf("EXFTH did %s; File now %s\n",opmsg,
                        (fcd->openMode & OPEN_NOT_OPEN)?"Closed":"Open");
       return sts;
    }

    /*********************************************************
     *  TSTFH - External File Handler entry point.
    *********************************************************/
    int
    TSTFH( unsigned char *opCodep, FCD3 *fcd)
    {
       unsigned int   opCode;
       char  *fname;
       int      sts, ky, j, k;

       if(*opCodep == 0xfa)
          opCode = 0xfa00 + opCodep[1];
       else
```

```
        opCode = opCodep[1];

    if(fcd->fileOrg == ORG_LINE_SEQ
    || fcd->fileOrg == ORG_SEQ
    || fcd->fileOrg == ORG_INDEXED
    || fcd->fileOrg == ORG_RELATIVE) {
        switch (opCode) {
        case OP_OPEN_OUTPUT:
        case OP_OPEN_IO:
        case OP_OPEN_EXTEND:
        case OP_OPEN_OUTPUT_NOREWIND:
            return doOpenFile( opCodep, fcd, txtOpCode(opCode));
            break;

        case OP_OPEN_INPUT:
        case OP_OPEN_INPUT_NOREWIND:
        case OP_OPEN_INPUT_REVERSED:
            return doOpenFile( opCodep, fcd, txtOpCode(opCode));
            break;

        case OP_CLOSE:
            return doOpenFile( opCodep, fcd, txtOpCode(opCode));
            break;

        default:
            break;
        }

    }

    if(opCode == OP_CLOSE
    && (fcd->openMode & OPEN_NOT_OPEN) ) {
        return 0;
    }

    sts = EXTFH( opCodep, fcd );
    return sts;
}
static char *           /* Return Text name of function */
txtOpCode(int opCode)
{
    static char tmp[32];
    switch (opCode) {
    case OP_OPEN_INPUT:     return "OPEN_IN";
    case OP_OPEN_OUTPUT:        return "OPEN_OUT";
    case OP_OPEN_IO:     return "OPEN_IO";
    case OP_OPEN_EXTEND:        return "OPEN_EXT";
    case OP_OPEN_INPUT_NOREWIND:  return "OPEN_IN_NOREW";
    case OP_OPEN_OUTPUT_NOREWIND: return "OPEN_OUT_NOREW";
    case OP_OPEN_INPUT_REVERSED:  return "OPEN_IN_REV";
    case OP_CLOSE:          return "CLOSE";
    case OP_CLOSE_LOCK:     return "CLOSE_LOCK";
    case OP_CLOSE_NOREWIND:     return "CLOSE_NORED";
    case OP_CLOSE_REEL:     return "CLOSE_REEL";
    case OP_CLOSE_REMOVE:       return "CLOSE_REMOVE";
    case OP_CLOSE_NO_REWIND:    return "CLOSE_NO_REW";
    case OP_START_EQ:       return "START_EQ";
    case OP_START_EQ_ANY:       return "START_EQ_ANY";
    case OP_START_GT:       return "START_GT";
    case OP_START_GE:       return "START_GE";
    case OP_START_LT:       return "START_LT";
    case OP_START_LE:       return "START_LE";
    case OP_READ_SEQ_NO_LOCK:  return "READ_SEQ_NO_LK";
    case OP_READ_SEQ:       return "READ_SEQ";
    case OP_READ_SEQ_LOCK:      return "READ_SEQ_LK";
    case OP_READ_SEQ_KEPT_LOCK:    return "READ_SEQ_KEPT_LK";
    case OP_READ_PREV_NO_LOCK:     return "READ_PREV_NO_LK";
```

```
    case OP_READ_PREV:      return "READ_PREV";
    case OP_READ_PREV_LOCK:    return "READ_PREV_LK";
    case OP_READ_PREV_KEPT_LOCK:  return "READ_PREV_KEPT_LK";
    case OP_READ_RAN:       return "READ_RAN";
    case OP_READ_RAN_NO_LOCK:  return "READ_RAN_NO_LK";
    case OP_READ_RAN_KEPT_LOCK:   return "READ_RAN_KEPT_LK";
    case OP_READ_RAN_LOCK:     return "READ_RAN_LK";
    case OP_READ_DIR:       return "READ_DIR";
    case OP_READ_DIR_NO_LOCK:  return "READ_DIR_NO_LK";
    case OP_READ_DIR_KEPT_LOCK:   return "READ_DIR_KEPT_LK";
    case OP_READ_DIR_LOCK:     return "READ_DIR_LK";
    case OP_READ_POSITION:     return "READ_POSITION";
    case OP_WRITE:          return "WRITE";
    case OP_REWRITE:     return "REWRITE";
    case OP_DELETE:      return "DELETE";
    case OP_DELETE_FILE:       return "DELETE_FILE";
    case OP_UNLOCK:      return "UNLOCK";
    case OP_ROLLBACK:       return "ROLLBACK";
    case OP_COMMIT:      return "COMMIT";
    case OP_WRITE_BEFORE:      return "WRITE_BEFORE";
    case OP_WRITE_BEFORE_TAB:  return "WRITE_BEFORE_TAB";
    case OP_WRITE_BEFORE_PAGE:    return "WRITE_BEFORE_PAGE";
    case OP_WRITE_AFTER:       return "WRITE_AFTER";
    case OP_WRITE_AFTER_TAB:   return "WRITE_AFTER_TAB";
    case OP_WRITE_AFTER_PAGE:  return "WRITE_AFTER_PAGE";
    }
    sprintf(tmp,"Func 0x%02X:",opCode);
    return tmp;
}
```

Sample COBOL program

```
==========================
           IDENTIFICATION DIVISION.
           PROGRAM-ID.    SEQFIX.

           ENVIRONMENT DIVISION.
           CONFIGURATION SECTION.

           INPUT-OUTPUT SECTION.
           FILE-CONTROL.
               SELECT FLATFILE ASSIGN EXTERNAL SEQFIX
               ORGANIZATION SEQUENTIAL
               FILE STATUS IS CUST-STAT .

           DATA  DIVISION.
           FILE SECTION.
           FD  FLATFILE
               BLOCK CONTAINS 5 RECORDS.

           01  TSPFL-RECORD.
               10  CM-CUST-NUM                    PICTURE X(8).
               10  CM-COMPANY                     PICTURE X(25).
               10  CM-DISK                        PICTURE X(8).
               10  CM-NO-TERMINALS                PICTURE 9(4) COMP-4.
               10  CM-PK-DATE                     PICTURE S9(14) COMP-3.
               10  CM-TRAILER                     PICTURE X(8).

           WORKING-STORAGE SECTION.

           77  MAX-SUB           VALUE  6         PICTURE 9(4) COMP SYNC.
           77  CUST-STAT                          PICTURE X(2).
           01  TEST-DATA.

             02  DATA-CUST-NUM-TBL.

               05  FILLER PIC X(8) VALUE "ALP00000".
               05  FILLER PIC X(8) VALUE "BET00000".
               05  FILLER PIC X(8) VALUE "GAM00000".
               05  FILLER PIC X(8) VALUE "DEL00000".
               05  FILLER PIC X(8) VALUE "EPS00000".
               05  FILLER PIC X(8) VALUE "FOR00000".

             02  DATA-CUST-NUM REDEFINES DATA-CUST-NUM-TBL
                                       PIC X(8) OCCURS 6.
             02  DATA-COMPANY-TBL.

               05  FILLER PIC X(25) VALUE "ALPHA ELECTRICAL CO. LTD.".
               05  FILLER PIC X(25) VALUE "BETA SHOE MFG. INC.      ".
               05  FILLER PIC X(25) VALUE "GAMMA X-RAY TECHNOLOGY   ".
               05  FILLER PIC X(25) VALUE "DELTA LUGGAGE REPAIRS    ".
               05  FILLER PIC X(25) VALUE "EPSILON EQUIPMENT SUPPLY ".
               05  FILLER PIC X(25) VALUE "FORTUNE COOKIE COMPANY   ".
             02  DATA-COMPANY  REDEFINES DATA-COMPANY-TBL
                                       PIC X(25) OCCURS 6.
             02  DATA-ADDRESS-2-TBL.

               05  FILLER PIC X(10) VALUE "NEW YORK  ".
               05  FILLER PIC X(10) VALUE "ATLANTA   ".
               05  FILLER PIC X(10) VALUE "WASHINGTON".
               05  FILLER PIC X(10) VALUE "TORONTO   ".
               05  FILLER PIC X(10) VALUE "CALGARY   ".
               05  FILLER PIC X(10) VALUE "WHITEPLAIN".

             02  DATA-ADDRESS   REDEFINES DATA-ADDRESS-2-TBL
                                       PIC X(10) OCCURS 6.
```

```
  02  DATA-NO-TERMINALS-TBL.

    05  FILLER PIC 9(3) COMP-3 VALUE 10.
    05  FILLER PIC 9(3) COMP-3 VALUE 13.
    05  FILLER PIC 9(3) COMP-3 VALUE 75.
    05  FILLER PIC 9(3) COMP-3 VALUE 10.
    05  FILLER PIC 9(3) COMP-3 VALUE 90.
    05  FILLER PIC 9(3) COMP-3 VALUE 254.

  02  DATA-NO-TERMINALS REDEFINES DATA-NO-TERMINALS-TBL
                              PIC 9(3) COMP-3 OCCURS 6.
01  WORK-AREA.
    05  SUB                            PICTURE 9(4) COMP SYNC.
        88  ODD-RECORD                 VALUE 1 3 5.
PROCEDURE DIVISION.
    PERFORM LOADFILE.
    OPEN I-O FLATFILE.
    READ FLATFILE
    DISPLAY "Read " CM-CUST-NUM " Sts:" CUST-STAT.
    ADD 1 TO CM-NO-TERMINALS
    REWRITE TSPFL-RECORD
    DISPLAY "REWRITE " CM-CUST-NUM " Sts " CUST-STAT
            " Trms:" CM-NO-TERMINALS.
    CLOSE FLATFILE.
    STOP RUN RETURNING 0.

READ-RECORD.
    MOVE SPACES                       TO TSPFL-RECORD.
    READ FLATFILE
    IF CUST-STAT NOT = "00"
      DISPLAY "Read Status: " CUST-STAT
    ELSE
      DISPLAY "Read  " CM-CUST-NUM
              " Trms:" CM-NO-TERMINALS
    END-IF.

LOADFILE.
    OPEN OUTPUT FLATFILE.
    PERFORM LOAD-RECORD
                VARYING SUB FROM 1 BY 1
                  UNTIL SUB > MAX-SUB.
    CLOSE FLATFILE.

LOAD-RECORD.
    MOVE SPACES                       TO TSPFL-RECORD.
    MOVE DATA-CUST-NUM      (SUB)     TO CM-CUST-NUM.
    MOVE DATA-COMPANY       (SUB)     TO CM-COMPANY.
    MOVE DATA-NO-TERMINALS  (SUB)     TO CM-NO-TERMINALS.
    MOVE 20070319                     TO CM-PK-DATE.
    IF SUB = 1 OR 4 OR 6
        MOVE -20070319                 TO CM-PK-DATE.
    IF  ODD-RECORD
        MOVE "8417"                   TO CM-DISK
    ELSE
        MOVE "8470"                   TO CM-DISK.
    WRITE TSPFL-RECORD.
```

Compile Command

==========

cobc -debug -fstatic-call -fcallfh=TSTFH -w -x seqfix.cbl tstfh.c

Output from program

```
=================
    EXFTH did OPEN_OUT; File now Open
    EXFTH did CLOSE; File now Closed
    EXFTH did OPEN_IO; File now Open
    Read ALP00000 Sts:00
    REWRITE ALP00000 Sts 00 Trms:0011
    EXFTH did CLOSE; File now Closed
```

Although this sample of using EXTFH does not do much of importance, it does demonstrate how a C module can intercept I/O requests and either do something special or pass on to the built-in EXTFH function.