Chapter 4

COBOL SUBROUTINES
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>SUBPROGRAM LINKAGE CONCEPTS</td>
<td>3</td>
</tr>
<tr>
<td>CONTROL TRANSFERS</td>
<td>3</td>
</tr>
<tr>
<td>COMMON DATA</td>
<td>4</td>
</tr>
<tr>
<td>SUBPROGRAM LINKAGE--DATA DIVISION</td>
<td>4</td>
</tr>
<tr>
<td>LINKAGE SECTION</td>
<td>4</td>
</tr>
<tr>
<td>SUBPROGRAM LINKAGE--PROCEDURE DIVISION</td>
<td>5</td>
</tr>
<tr>
<td>THE PROCEDURE DIVISION HEADER</td>
<td>5</td>
</tr>
<tr>
<td>ENTRY STATEMENT</td>
<td>5</td>
</tr>
<tr>
<td>CALL STATEMENT</td>
<td>6</td>
</tr>
<tr>
<td>USING OPTION</td>
<td>6</td>
</tr>
<tr>
<td>SUBROUTINE TERMINATION &amp; RE-ENTRY</td>
<td>8</td>
</tr>
<tr>
<td>GOBACK</td>
<td>8</td>
</tr>
<tr>
<td>EXIT</td>
<td>8</td>
</tr>
<tr>
<td>STOP RUN</td>
<td>8</td>
</tr>
<tr>
<td>CMS CONSIDERATIONS</td>
<td>9</td>
</tr>
<tr>
<td>JCL CONSIDERATIONS</td>
<td>10</td>
</tr>
<tr>
<td>PRIMARY INPUT</td>
<td>10</td>
</tr>
<tr>
<td>ADDITIONAL INPUT</td>
<td>11</td>
</tr>
<tr>
<td>APPENDIX A_ EXAMPLES</td>
<td>13</td>
</tr>
</tbody>
</table>
COBOL SUBROUTINES

INTRODUCTION

Subroutines are a collection of self-contained statements that may be compiled separately from the main program and other subroutines. You are familiar with the use of internal subroutines within a COBOL program by the use of the PERFORM verb. An external subroutine (subprogram) written in COBOL is just like a regular program with the four divisions: Identification, Environment, Data and Procedure. In order to create the linkage between the main program and the subprogram(s), several commands and a new Data Division Section are needed.

COBOL programmers use too few subprograms. The reason is that the subroutine feature was added rather late in COBOL, it requires more effort to write a COBOL subroutine, and the numeric and character data of COBOL is harder to pass in subroutine calls because all programs calling the subroutine must pass the data in the same precision and character length. The many required COBOL statements by themselves may outnumber the Procedure Division statements in the subroutine, and programmers find it easier to include the statements as a paragraph or section than as a subroutine. Nonetheless, external subroutines can be a very useful device, especially in complex applications. Modules are used commercially to split up workloads, as an aid in updating at a later time and as a debugging tool.

SUBPROGRAM LINKAGE CONCEPTS

When the solution to a problem is divided into more than one program, the constituent programs must be able to communicate with each other through transfers of control and/or through reference to common data.

CONTROL TRANSFERS

In the Procedure Division of a COBOL program, a CALL statement transfers control to a called program, and a called program may itself transfer control to yet another called program. However, a called program must not directly or indirectly call its caller (such as program A calling program B; program B calling program C; and program C then calling program A) or results are unpredictable.

When control is passed to a called program, execution proceeds in the usual way. When called program processing is completed, the program can either transfer control back to the calling program, call another program, or end the run unit.

A called program is in its initial state the first time it is called within a run unit. On all other entries into the called program, the state of the program remains
unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

COMMON DATA

Program interaction may require that both programs have access to the same data.

In a calling program the common data items are described in the same manner as other File or Working-Storage Section items.

In a called program, common data items are described in the Linkage Section.

SUBPROGRAM LINKAGE--DATA DIVISION

Data items referred to in the calling program may be defined in the File, Working-Storage or Linkage Sections. If an identifier in the calling program USING list is a file or is in a file, it must be opened in the calling program preceding transfer of control to the called program.

LINKAGE SECTION

In the Data Division of a called program, the programmer specifies in the Linkage Section those data items that are common with the calling program.

A Linkage Section has meaning if, and only if, this object program functions under control of a CALL statement that contains the USING option.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Items described in the Linkage Section do not have space allocated for them in this program; Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program.

Variables specified in a USING clause of an ENTRY statement or on the Procedure Division header must be defined in the Linkage Section in the called program using level numbers 01 or 77. Since no values can be assigned to these items in the Linkage Section as no space is allocated for them in memory, they should be initialized in the main program where space is allocated.

Data-names in the Linkage Section of the called program need not be the same as the data-names in the calling program; indeed, if the two programs were written by different people at different times, it is unlikely that they would be identical. However, variables in the called program and the calling program should describe data that are of the same length (and preferably of the same type) because the
called program will actually access the data in the calling program's Data Division. It is also important that the two programs agree on the order (or sequence) of the data elements. The order of the data elements is established by the USING clause in the Procedure Division header of the called program.

**SUBPROGRAM LINKAGE--PROCEDURE DIVISION**

In the Procedure Division, control is transferred between COBOL object programs by means of the CALL statement. Reference to common data is provided through the USING option, which can be specified in the CALL statement and in the called program's Procedure Division header.

An ENTRY statement makes provision for alternative entry points into a called program; the USING option can also be specified in the ENTRY statement of a called program.

The GOBACK statement allows termination of called program processing or of main COBOL program processing.

An EXIT PROGRAM statement allows termination of called program processing. STOP RUN terminates run unit execution.

**THE PROCEDURE DIVISION HEADER**

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION [USING data-name-1 [, data-name-2] ...].
```

A USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

Each of the data-names (identifiers) in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

**ENTRY STATEMENT**

The ENTRY Statement establishes an additional starting point in a called program other than the first statement. The format is:

```
ENTRY literal-1 [USING identifier-1 [identifier-2]...]
```
Literal-1 in this case is not the name of the program (PROGRAM-ID), as when the
entry is the first statement, but is any other name conforming to the same rules as
specified for the CALL statement. Again the USING option is available. Control
passes to the next statement following the ENTRY statement specified by the
invoking CALL.

CALL STATEMENT

The CALL statement functions just like the PERFORM statement, except that it
is used to connect two programs: the one containing the CALL, and the one named
in the CALL statement. To invoke or transfer control to a subprogram a CALL
statement is placed in the Procedure Division of the invoking (calling) program
using the following format:

            CALL literal-1 [USING identifier-1 [identifier-2] ...]

Literal-1 is the program-id or ENTRY name to which control is to be given at
execution time. Literal-1 is a nonnumeric literal of 8 characters or less, the first of
which is alphabetic, and is either a program name (as denoted in the
PROGRAM-ID paragraph in the called program) or the name of an entry point
(denoted by an ENTRY statement or the name of an entry point in the called
program). Identifier-1 and identifier-2 are data items located in the File Section or
Working-Storage Section of the calling program. Identifier-1 and identifier-2
correspond to data-name-1 and data-name-2, respectively, of the Procedure Division
header of the called program. (Data-name and identifier can be used
interchangeably, but were both used in this illustration to show their
correspondence).

Called subprograms may contain CALL statements, however, a called subprogram
must not contain a CALL statement that directly or indirectly calls the calling
program.

A CALL statement in the calling program transfers control to the first
nondeclarative procedural statement in the called program. (Refer to the use of
Declaratives in a COBOL manual if this is unclear.)

USING OPTION

The USING option enables data values to be passed to the subprogram from the
main program. USING is available on the CALL statement, ENTRY statement and
Procedure Division header in the formats shown below:

(Within a Calling Program)
CALL literal-1 [USING identifier-1 [identifier-2]...]

(Within a Called Program)

ENTRY literal-1 [USING identifier-1 [identifier-2]...]
or

PROCEDURE DIVISION [USING identifier-1 [identifier-2]...]

Identifier-1, etc. specified in the USING clause indicates data items available to a calling program that may be referred to in a called program. The order of appearance of USING identifiers in both calling and called programs determines the correspondence of single sets of data available to both programs. Note again that the correspondence is positional and not by name. Corresponding identifiers must contain the same number of characters, although their data descriptions need not be the same. The number of identifiers should be the same in each list, but it is permissible for the calling program to specify more, in which case only those specified in the CALL statement will be linked. The data descriptions must be aligned (i.e. PIC clauses must be equivalent). Proper word-boundary alignment is essential when a data-item is specified as COMP, COMP-1, or COMP-2, so use the synchronized (SYNC) option with these mode clauses.

An example of an entry statement and a subprogram PROCEDURE DIVISION are below, with its corresponding call statement:

(Within a Calling Program)

CALL 'TOTALING' USING EMP-CARD WORKING-TOTALS WORKING-AMTS.

(Within a Called Program)

ENTRY 'TOTALING' USING EMPLOYEES TOTALS AMOUNTS.
or

PROGRAM-ID TOTALING.

: PROCEDURE DIVISION USING EMPLOYEES TOTAL AMOUNTS.

NOTE: EMP-CARD corresponds to EMPLOYEES
      WORKING-TOTALS corresponds to TOTALS
      WORKING-AMTS corresponds to AMOUNTS

Page 4 - 7
SUBROUTINE TERMINATION & RE-ENTRY

There are three ways to effect termination of a program or subroutine.

GOBACK

The GOBACK marks the end of a called program. It must be the only statement or the last of a series of imperative statements in a sentence (i.e. it must be followed by a period). The general format of a GOBACK statement is:

(Within a Called Program)

GOBACK

If control reaches a GOBACK statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

EXIT

The EXIT PROGRAM statement specifies the logical end of a called program. The format is:

(Within a Called Program)

paragraph-name. EXIT PROGRAM.

The EXIT statement must be preceded by a paragraph-name, and must be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches an EXIT PROGRAM statement, and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph.

STOP RUN

A STOP RUN is also possible and will return control directly to the system.

The effects of these statements are shown in the following table:
### TABLE 4.1

**EFFECT ON PROGRAM TERMINATION IN MAIN AND SUBPROGRAMS**

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>MAIN PROGRAM</th>
<th>SUBPROGRAM</th>
</tr>
</thead>
</table>
| GOBACK.     | Return to invoker - system.  
Causes end of job step. | Return to invoking program.                           |
| EXIT PROGRAM| Non-Operational. Execution continues with next statement.                    | Return to invoking program.                           |
| STOP RUN.   | Return to invoker - system.  
Causes end of job step. | Return directly to invoker of main program-system.  
Causes end of job step. |

Note: The first time a program is entered its state is that of a fresh program and data values are those of the called program when control was passed. On each subsequent entry, however, its state is as it was upon the last exit and data values are those of the last exit unless changed in the main program before re-entry. Reinitialization of the following items is thus essential:

- **GO TO** statements that have been **ALTERED TALLY**
- data items
- **ON** statements
- **PERFORM** statements
- **EXHIBIT CHANGED** or **EXHIBIT CHANGED NAMED**

The range of a **PERFORM** is still in effect upon subsequent entry, therefore, great care must be exercised.

**CMS CONSIDERATIONS**

After you have written the COBOL main and subprograms and logically linked them, you must get the computer to physically link them. Within CMS the linkage Edit operation is done during the LOAD step. In this step the object modules are physically linked together so that they are able to interact during execution. This is accomplished through the following sequence:
1) use COBOL2 to compile both programs,
2) LOAD the main program
3) INCLUDE the subprogram
4) START */ NOSTAE

JCL CONSIDERATIONS

Once the COBOL main and subprograms are written and logically linked, you must get the computer to physically link them using Job Control Language (JCL). The linkage edit step is the crucial step in this process. The object modules must be physically linked together in order for them to be able to interact during execution. This can be done in two ways, either concatenating as primary input or as additional input. Complete JCL examples are provided in the appendix as an illustration of each approach.

PRIMARY INPUT

The use of primary input is perhaps the easiest, but may not be as prevalent in industry. Each COBOL program (main or subroutine) is compiled separately using the same load module data set name and a disposition of (MOD,PASS). This forces concatenation of each object module into one. Then, in the linkage edit step, all external references are resolved in this module and in the COBOL libraries.

EXAMPLE 1, in Appendix A of this Chapter, is an illustration of the primary input approach.

In Example 1B only the last source program to be compiled is executed under COB2CLG; all previous source programs in this job have simply been compiled. The catalogued procedures in both cases, COB2C and COB2CLG, use the same data set name (DSN=&&LOADSET) and specify concatenation in the DISP=(MOD,PASS) parameter. The only thing you have to specify is which program, in that module, is the first program to be executed. This is done by an ENTRY statement in the LKED procedure step of the COB2CLG or COB2LG job steps. It is formatted below:

ENTRY external name

The external name is the name specified in the Identification Division PROGRAM-ID statement of the main (calling) program. This is a linkage editor control statement and is placed as an input card to the LKED.SYSIN data definition statement. ENTRY must start in card column 2 or later, be followed by at least one blank and then followed by the external name of the program to be executed first.
In Example 1 the program called COBOL SOURCE A1 is the main program and is therefore specified in the ENTRY statement.

**ADDITIONAL INPUT**

In Example 1 the job setup would only work if you had each program in a source code format. To save compiling each time, a program can be kept on disk in object form. To then link this object program to the other programs requires a new linkage editor control statement. There are two statements that can be used to effect this linkage and they are formatted below:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE</td>
<td>ddname [(member-name\n    [member-name]...)]\n    [ddname[(member-name\n    [member-name]...)]]...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY</td>
<td>ddname (member-name\n    [member-name]...\n    [ddname(member-name\n    [member-name]...)]...</td>
</tr>
</tbody>
</table>

In either statement the ddname indicates the name of a DD statement within the step that specifies a library or sequential data set that contains one or more separately compiled object modules. The member name is a member of the library if it is a partitioned data set. The LIBRARY statement can also contain a library of load modules, but you cannot mix the two kinds in one statement. The input specified in the INCLUDE statement is concatenated directly to the object module specified in the SYSLIN DD statement. The input specified in a LIBRARY statement is searched last after the SYSLIN DD modules and INCLUDE modules, except for those reserved for the automatic library call.

If you should need to continue a linkage editor control statement, stop at a convenient comma before column 72, place a non-blank character in column 72 and start the next card in column 16 (just like UTILITY control statements).
Job setups using additional input are given in Examples 2 and 3, Appendix A of this chapter.

When creating a permanent object module, be sure to place the COB2.SYSLIN DD statement directly following the EXEC statement in order to override the temporary data set that would otherwise be created.

When linking a new object module with the old object module be sure to place the LKED override cards after the COB cards. If the cards do not follow the input stream they will be ignored, therefore, place them in COMPIL, LINKAGE EDIT, and GO order. Note that in each example an ENTRY statement is still needed.
APPENDIX A. EXAMPLES

This appendix contains a sample COBOL main program (COBOL SOURCE A1) and a sample COBOL subprogram (COBOL SUB A1) with three different examples of compilation, link editing, and execution.

EXAMPLE 1:
1. Compile the main program, saving the object module in a temporary data set.
2. Then compile the subprogram and concatenate this object module onto the end of the main.
3. In the next step, load and go, link editing the combined object modules.

EXAMPLE 2:
1. Compile the subprogram first and save the object module in a temporary data set.
2. Then compile, link, and go, compiling the main and linking the two object modules which are concatenated through JCL.

EXAMPLE 3:
1. Compile the sub and save it in a partitioned data set.
2. Compile the main saving it as a new member on the same partitioned data set.
3. The next step then links the two object modules together and saves the load module in a different partitioned data set.
4. The last step in this job does a load and go using the saved load module.

Note: Refer to the Programmer’s Guide for explanation re: the use of the dynam/nodynam option.
IDENTIFICATION DIVISION.
PROGRAM-ID. COBMAIN.
INSTALLATION. UNT.
DATE-Written. JULY 17, 1992.
DATE-COMPiled. 07/18/92.
*
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT INPUT-FILE ASSIGN TO DA-S-INPUT.
   SELECT PRINT-FILE ASSIGN TO UT-S-OUTPUT.
*
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE
   RECORDING MODE IS F
   LABEL RECORDS ARE OMITTED
   RECORD CONTAINS 80 CHARACTERS
   DATA RECORD IS INPUT-REC.
   01 INPUT-REC PIC X(80).
*
FD PRINT-FILE
   RECORDING MODE IS F
   LABEL RECORDS ARE OMITTED
   RECORD CONTAINS 132 CHARACTERS
   DATA RECORD IS PRINT-REC.
   01 PRINT-REC PIC X(132).
*
WORKING-STORAGE SECTION.
  01 SYSTEM-FLAGS.
     05 EOD PIC 9 VALUE ZERO.
*
  01 AMT PIC 9(3) VALUE 1.
  01 TOT PIC 9(4) VALUE ZERO.
  01 TOT-LINE.
     05 FILLER PIC X(30) VALUE 'TOTAL RECORDS PROCESSED = '.
     05 PTOT PIC ZZZ9.
     05 FILLER PIC X(35) VALUE 'THIS VALUE ACCUMULATED IN COBSUB'.
  01 DATA-IN.
     05 FILLER PIC X VALUE SPACES.
     05 CARD-IN PIC X(80).
*
PROCEDURE DIVISION.
000-Main-Loop SECTION.
  PERFORM 100-INITIALIZE-SYS.
  PERFORM 200-PROCESS-DATA UNTIL EOD = 1.
  MOVE TOT TO PTOT.
WRITE PRINT-REC FROM TOT-LINE AFTER 2 LINES.
PERFORM 300-CLOSE-FILES.
STOP RUN.

* 100-INITIALIZE-SYS SECTION.
110-OPEN-FILES.
OPEN INPUT INPUT-FILE
OUTPUT PRINT-FILE.
190-INITIALIZE-SYS-EXIT.
EXIT.

* 200-PROCESS-DATA SECTION.
210-READ-RECORD.
READ INPUT-FILE INTO CARD-IN
AT END
MOVE 1 TO EOD
GO TO 290-PROCESS-DATA-EXIT
END-READ.
220-WRITE-RECORD.
WRITE PRINT-REC FROM DATA-IN AFTER 1 LINES.
290-PROCESS-DATA-EXIT.
EXIT.

* 300-CLOSE-FILES SECTION.
310-CLOSE-FILES.
CLOSE INPUT-FILE
PRINT-FILE.
390-CLOSE-FILES-EXIT.
EXIT.

COBOL SUB A1 (Subprogram)

000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID. COBSUB.
000003 INSTALLATION. UNT.
000004 DATE-WRITTEN. JULY 17, 1992.
000005 DATE-COMPILED. 07/18/92.
000006 *
000007 ENVIRONMENT DIVISION.
000008 *
000009 DATA DIVISION.
000010 *
000011 WORKING-STORAGE SECTION.
000012 01 SYSTEM-FLAGS.
000013 05 DUMMY PIC 9 VALUE ZERO.
000014 *
000015 LINKAGE SECTION.
000016 01 X PIC 9(3).
000017 01 Y PIC 9(4).
*  PROCEDURE
  000-MAIN-LOOP
    ADD X TO Y.
  001-EXIT
    GOBACK.

*  DIVISION USING X Y.
  SECTION.

  SECTION.
EXAMPLE 1A

//FE16EX1A JOB (FE16,10,2),HOEKE,PASSWORD=
/*/ROUTE PRINT UNTVM1.FE16
/*/ROUTE PUNCH UNTVM1.FE16
/*/**************************************************************************
/*/ STEP1 COMPILE THE MAIN PROGRAM AND SAVES THE OBJECT MODULE
/*/ ON &LOADSET
***************************************************************************/
/*/STEP1 EXEC COB2C,PARM.COB2=(LIST,MAP,NOLIB,VBREF,XREF,
  Fdump,_DUMP,NODYNAM)
  /COB2.SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
  SPACE=(TRK,(3,3)),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
  /COB2.SYIN DD *
  /INC COBOL SOURCE A1
/*/**************************************************************************
/*/***************************************************************************/
/*/ STEP2 COMPILE THE COBOL SUBPROGRAM AND MODS THE OBJECT MODULE
/*/ ONTO THE END OF &LOADSET
***************************************************************************/
/*/STEP2 EXEC COB2C,PARM.COB2=(LIST,MAP,NOLIB,VBREF,XREF,
  Fdump,_DUMP,NODYNAM)
  /COB2.SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
  SPACE=(TRK,(3,3)),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
  /COB2.SYIN DD *
  /INC COBOL SUB A1
/*/**************************************************************************
/*/***************************************************************************/
/*/ STEP3 LINKEDITS THE TWO OBJECT MODULES TOGETHER AND DOES
/*/ A LOAD AND GO
***************************************************************************/
/*/STEP3 EXEC COB2LG
  /LKED.SYSLIN DD DSN=&LOADSET,DISP=(OLD,DELETE)
  /LKED.SYIN DD *
    ENTRY COBMAIN
/*/GO.SYSABOUT DD SYSOUT=A
  /GO.SYSDOUT DD SYSOUT=A
  /GO.SYSDUMP DD SYSOUT=A
  /GO.SYSOUT DD SYSOUT=A
  /GO.OUTPUT DD SYSOUT=A
  /GO.INPUT DD *
  RECORD NUMBER 1
  RECORD NUMBER 2
  RECORD NUMBER 3
RECORD NUMBER 4
RECORD NUMBER 5
/
//
//FE16EX1B JOB (FE16,10,2),HOEKE,PASSWORD=
/*ROUTE PRINT UNTVM1.FE16
/*ROUTE PUNCH UNTVM1.FE16
/*/ 
/************************************************
/*/ STEP1 COMPILES THE SUBPROGRAM AND SAVES THE OBJECT MODULE
/*/ ON &LOADSET 
/************************************************
/*/ STEP1 EXEC COB2C,PARM.COB2=(LIST,MAP,NOLIB,VBREF,XREF,
//FDUMP,DUMP,NODYNAM)
//COB2.SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//SPACE=(TRK,(3,3)),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
//COB2.SYSIN DD *
/INC COBOL SUB A1
/*/ 
/*/ 
/************************************************
/*/ STEP2 COMPILES THE COBOL MAIN PROGRAM AND MODS THE OBJECT MODULE
/*/ ON TO THE END OF &LOADSET 
/************************************************
/*/ STEP2 EXEC COB2CLG,PARM.COB2=(LIST,MAP,NOLIB,VBREF,XREF,
//FDUMP,DUMP,NODYNAM)
//COB2.SYSLIN DD *
/INC COBOL SOURCE A1
/*/ 
/*/ 
/************************************************
/*/ NEXT STEP2 LNKEDITS THE TWO OBJECT MODULES TOGETHER AND DOES
/*/ A LOAD AND GO 
/************************************************
/*/ LKED.SYSLIN DD DSN=&LOADSET,DISP=(OLD,DELETE)
//LKED.SYSIN DD *
ENTRY COBMAS
/*/ 
//GO.SYSABOUT DD SYSOUT=A
//GO.SYSDBOUT DD SYSOUT=A
//GO.SYSDUMP DD SYSOUT=A
//GO.SYSOUT DD SYSOUT=A
//GO.OUTPUT DD SYSOUT=A
//GO.INPUT DD *
RECORD NUMBER 1
RECORD NUMBER 2
RECORD NUMBER 3
RECORD NUMBER 4
RECORD NUMBER 5
RECORD NUMBER 6

Page 4 - 19
OUTPUT from Example 1B:

TOTAL RECORDS PROCESSED = 10  THIS VALUE ACCUMULATED IN COBSUB

Note that in Example 1B, the subprogram is saved in the object module before the main program, therefore, the order in which they are compiled and saved is not important.

EXAMPLE 2

//FE16EX2  JOB (FE16,:10,2),HOEKE,PASSWORD=
/*/ROUTE PRINT UNTVM1.FE16
/*/ROUTE PUNCH UNTVM1.FE16
/*/  
/*/STEP1 COMPILES THE SUBPROGRAM AND SAVES THE OBJECT MODULE
/*/ON &SUBOBJ
/*/STEP1  EXEC COB2C,PARM.COB2=(LIST,MAP,NOLIB,VBREF,XREF,  
//FDUMP,DUMP,NODYNAM)
//COB2.SYSLIN DD DSN=&SUBOBJ,DISP=(MOD,PASS),UNIT=SYSDA,  
//SPACE=(TRK,(3,3)),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
//COB2.SYSIN DD *
//INC COBOL SUB A1
/*/  
/*/STEP2 COMPILES THE COBOL MAIN PROGRAM AND THEN DOES A LINK AND GO
/*/USING THE MAIN OBJECT MODULE FROM STEP2 AND THE SUBPROGRAM
/*/OBJECT MODULE FROM THE PREVIOUS STEP
/*/  

Page 4 - 20
**STEP2** EXEC COB2CLG, PARM.COB2=(LIST, MAP, NOLIB, VBREF, XREF, FDUMP, DUMP, NODYNAM)

/*
// COB2.SYSIN DD *
/INC COBOL SOURCE A1
/
// LKED.SYSLIN DD DSN=&amp;LOADSET, DISP=(OLD,DELETE)
// DD DSN=&amp;SUBOBJ, DISP=(OLD,DELETE)
// LKED.SYSIN DD *
ENTRY COBMAIN
/
*/

GO.SYSABOUT DD SYSOUT=A
GO.SYSDBOUT DD SYSOUT=A
GO.SYSUDUMP DD SYSOUT=A
GO.SYSOUT DD SYSOUT=A
GO.OUTPUT DD SYSOUT=A
GO.INPUT DD *
RECORD NUMBER 1
RECORD NUMBER 2
RECORD NUMBER 3
RECORD NUMBER 4
RECORD NUMBER 5
*/

**EXAMPLE 3**

FE16EX3 JOB (FE16, :10,5), HEOKE, PASSWORD=
*/ROUTE PRINT UNTVM1.FE16
*/ROUTE PUNCH UNTVM1.FE16
/"

************************************************************************************
// STEP1 DELETES THE PARTITIONED DATA SETS
************************************************************************************

// STEP1 EXEC PGM=IEFBR14
// DELE
DD DSN=USER.FE16.OBJ.MOD, DISP=(OLD,DELETE),
// UNIT=SYSDA, VOL=SER=ACAD01
// DD DSN=USER.FE16.LOAD.MOD, DISP=(OLD,DELETE),
// UNIT=SYSDA, VOL=SER=ACAD01
/

************************************************************************************
// STEP2 COMPILES THE SUBPROGRAM AND STORES THE OBJECT MODULE
// AS "SUBOBJ" MEMBER ON THE OBJ.MOD DATA SET
************************************************************************************

// STEP2 EXEC COB2C, PARM.COB2=(LIST, MAP, NOLIB, VBREF, XREF, FDUMP, DUMP, NODYNAM)
// COB2.SYSLIN DD DSN=USER.FE16.OBJ.MOD(SUBOBJ),
// DISP=NEW,KEEP), UNIT=SYSDA, VOL=SER=ACAD01,
// DCB=(BLKSIZE=80, LRECL=80, RECFM=FB, DSOFG=PO),

Page 4 - 21
SPACE=(80,(1000,100,5))
/INC COBOL SUB A1
/*

STEP3 Compiles the main program and stores the object module
as "MAINOBJ" member on the OBJ.MOD data set

STEP3 EXEC COB2C, PARM.COB2=(LIST, MAP, NOLIB, VREF, XREF,
FDUMP, DUMP, NODYNAM)
/COB2.SYSLIN DD DSN=USER.FE16.OBJ.MOD(MAINOBJ),
DISP=(OLD, KEEP), UNIT=SYSDA, VOL=SER=ACAD01,
DCB=(BLKSIZE=80, LRECL=80, RECFM=FB, DSORG=PO),
SPACE=(80, (1000, 100, 5))
/COB2.SYSIN DD *
/INC COBOL SOURCE A1
/

STEP4 Links the two object modules from step2 and step3
TOGETHER AND PRODUCES A LOAD MODULE (COBLOAD) WHICH IS
STORED AS A MEMBER ON THE LOAD.MOD DATA SET

STEP4 EXEC PGM=IEWL, PARM=(XREF, LET, LIST), REGION=512K
/SYSLIN DD DSN=USER.FE16.OBJ.MOD(MAINOBJ), DISP=OLD,
UNIT=SYSDA, VOL=SER=ACAD01
/ SYSLIB DD DSN=USER.FE16.OBJ.MOD(SUBOBJ), DISP=OLD,
UNIT=SYSDA, VOL=SER=ACAD01
/ SYSLIB DD DSN=USER.FE16.OBJ.MOD(COLOAD),
UNIT=SYSDA, VOL=SER=ACAD01,
SPACE=(1024, (50, 20, 5))
/ SYSLIB DD DSN=SYS1.COB2LIB, DISP=SHR
/ SYSLIB DD DSN=SYS2.RSRCH.SUBLIB, DISP=SHR
/ SYSPRT DD UNIT=SYSDA, SPACE=(1024, (50, 20, 1))
/ SYSPRT DD SYSOUT=A
/ SYSPRT DD *
ENTRY COBMAIN
INCLUDE MAINOBJ, SUBOBJ
/*

STEP5 Loads and executes the load module from the previous
step

STEP5 EXEC PGM=COBLOAD, REGION=500K
/STEPLIB DD DSN=SYS1.COB2LIB, DISP=SHR
/ DD DSN=SYS2.RSRCH.SUBLIB, DISP=SHR
*/
DD DSN=USER.FE16_LOAD.MOD(COBLOAD),DISP=(OLD,PASS),UNIT=SYSDA, VOL=SER=ACAD01

//GO.SYSABOUT DD DUMMY
//GO.SYSDBOUT DD DUMMY
//GO.SYSUDUMP DD DUMMY
//GO.SYSLOUT DD SYSOUT=A
//GO.OUTPUT DD SYSOUT=A
//GO.INPUT DD *
RECORD NUMBER 1
RECORD NUMBER 2
RECORD NUMBER 3
RECORD NUMBER 4
RECORD NUMBER 5
/*
*/