CIS COBOL Operating Guide For Use With the CP/M Operating System

Version 4.5

CIS COBOL Operating Guide For Use With the CP/M Operating System: Version 4.5

Acknowledgements

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection herewith.

The authors and copyright holders of the copyrighted material used herein:

FLOW-MATIC (Trademark for Sperry Rand Corporation) Programming for the Univac® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Table of Contents

List of Figures

List of Tables

PREFACE

This manual describes operating procedures for the CP/M resident releases of the CIS COBOL Compiler and run-time libraries. The compiler converts CIS COBOL source code into an intermediate code which is then interpreted by the Run-Time System. The manual describes the steps needed to compile a program and then execute the compiled program, including all necessary run-time requirements. Operation of the run-time Debug package is also included.

MANUAL ORGANIZATION

Chapters 1 through 4 of this manual describe compiler features and general procedures for loading and execution of programs including linkage of assembler programs. Chapter 5 describes the operation of the configuration utility program CONFIG. Chapters 6, 7 and 8 describe the use of the optional additional software products with CIS COBOL.

The appendices provide summarized information for reference purposes and give configuration information for various run-time environments. Some appendices are omitted because they are not pertinent to this version of CIS COBOL.

AUDIENCE

This manual is intended for personnel already familiar with COBOL usage on other equipment.

This manual contains the following chapters and appendices.

"Chapter 1. Introduction", which gives a general description of the CIS COBOL system, its input and output files, and the run-time libraries provided with the compiler, plus the step-by-step outline of compilation, linking, locating and executing of sample interactive programs.

"Chapter 2. Compiler Controls", which describes compiler commands directives and listing formats.

"Chapter 3. "Run Time System Controls", which gives general instructions for running programs, console operation, CRT screen handling and interactive debugging.

"Chapter 4. Program Design Considerations", which describes the facilities available to overlay programs and invoke other COBOL programs or programs written in other languages from a main program.

"Chapter 5. CONFIG Utility", which gives the objectives of the CONFIG Utility, instructions for configuring standard and non-standard CRTs, and instructions for configuring run-time subroutines.

"Chapter 6. Incorporating FORMS-2 Utility Output", which describes the use of the FORMS-2 screen formatting utility programs output.

"Chapter 7. Using the ANIMATOR Utility Program", which enables debugging a COBOL program interactively on the screen at COBOL source code level.

"Appendix A. Summary of Compiler and Run Time Directives", summarizes the compiler directives available in the CIS COBOL compiler.

"Appendix B. Compile-Time Errors", which lists all errors that can be signalled during program compilation.

"Appendix C. Run-Time Errors", which lists all errors that can be signalled during program execution.

"Appendix D. Operating Systems Errors", which is a listing of the error messages issued by the CP/M Operating System.

"Appendix E. Interactive Debug Command Summary", which summarizes the commands that can be used with the CIS COBOL Interactive Debug program.

"Appendix F. CP/M Disk Files", which is a description of file naming conventions and formats used by CIS COBOL under CP/M.

"Appendix H. Example Configuration specifying Tab Stop Modification", which is a typical screen conversation.

"Appendix J. Example Configuration specifying User Subroutines", which is a typical screen conversation.

"Appendix K. Example Configuration in which No CRT Tailoring is Performed", which is a typical screen conversation to configure a system without CRT tailoring.

"Appendix M. Example Run Time Subroutines", which contains assembler listings of typical supplied sample subroutines.

"Appendix N. Example Use of Run Time Subroutines", which is an example of the way in which the supplied CALL code routines can be used.

"Appendix P. Constraints", which summarises constraints to be when programming using this release of CIS COBOL.

NOTATION IN THIS MANUAL

Throughout this manual, the following notation is used to describe the format of data input or output:

1. All words printed in small letters are generic terms representing names which will be devised by the programmer.
2. When material is enclosed in square brackets [ ], it is an indication that the material is an option which may be included or omitted as required.
3. The symbol << after a CRT entry or command format in this manual indicates that the CR (carriage return) or equivalent data input terminator key must be pressed to enter the command.

Headings are presented in this manual in the following order of importance:

```
CHAPTER  N
         Chapter Heading
TITLE

ORDER ONE HEADING
ORDER TWO HEADING
Order Three Heading      Text two lines down
Order Four Heading
Order Five Heading:      Text on same line
```

Numbers one (1) to nine (9) are written in text as letters, e.g. one.

Numbers ten (10) upwards are written in text as numbers, e.g. 12.

The phrase "For documentation purposes only" in the text of this manual means that the associated coding is accepted syntactically by the Compiler, but is ignored when producing the object program.

RELATED PUBLICATIONS

For details of the CIS COBOL Language, refer to the document:

CIS COBOL Language Reference Manual

For details of the CP/M Operating System, Messages, and File Structures refer to the CP/M Operating System User manuals.

The utility programs ANIMATOR and FORMS-2 are supplied with user manuals as follows:

CIS COBOL ANIMATOR Operating Guide

CIS COBOL FORMS-2 Utility Manual

Chapter 1. INTRODUCTION

GENERAL DESCRIPTION

COBOL (COmmon Business Oriented Language) is the most widely and extensively used language for the programming of commercial and administrative data processing.

CIS COBOL is a Compact, Interactive and Standard COBOL language system designed for use on microprocessor based computers and intelligent terminals under control of the CP/M Operating System. It is designed to run on any 48K byte microcomputer system with CRT and floppy diskettes under control of CP/M. Although the minimum system is as specified above, for maximum efficiency a 64K byte microcomputer with double-density diskettes is recommended.

The CIS COBOL compilation system converts CIS COBOL source code into an intermediate code which is then interpreted by a Run Time System (RTS).

CIS COBOL programs can be created using the standard CP/M text editor to create the CIS COBOL source files. The Compiler compiles the source programs from here, or they are entered interactively direct from the CRT. After compilation is finished, the Run Time System is linked with the compiled output to form a running user program. A listing of the CIS COBOL program is provided by the Compiler during compilation. Any error messages are included in this listing.

An interactive development software tool that enables run-time debugging of COBOL programs with the COBOL code simultaneously displayed is available, and is known as ANIMATOR. See Chapter 7.

Supplied with CIS COBOL is an interactive Debug software tool that enables run-time debugging of the run-time program at object code level. See Chapter 3.

Note

The Interactive Debug software supplied with CIS COBOL cannot be used if ANIMATOR is used. If you have ANIMATOR software, a decision must be made at compile time as to which debugging tool is required.

The standard ANSI DEBUG module is also included in CIS COBOL but this cannot be invoked if ANIMATOR is used.

The CIS COBOL System also incorporates a powerful utility program called FORMS-2.

The purpose of FORMS-2 is to allow the user to define the screen layouts to be used in a CIS COBOL application, by simply keying text at the keyboard and so producing model forms on the CRT. The forms can be automatically used to generate a program which will maintain files with the form data in them.

It provides an ideal medium of communication between the programmer and the end user who may know nothing of computers. The minimum storage requirement for FORMS-2 is 56k bytes.

The FORMS-2 Utility program is available from your CIS COBOL Dealer.

GETTING STARTED WITH CIS COBOL

ISSUE DISK

Each user is provided with the software that makes up the COBOL development system described above on a CIS COBOL Issue Disk.

A CIS COBOL Issue Disk contains the software listed in Table 1-1.

Table 1.1. Issue Disk Contents.

COMPILER	RUN-TIME SYSTEM	CONFIGURATOR
COBOL.COM COBOL.I01 COBOL.I02 COBOL.I03 COBOL.I04 COBOL.MSG	RUNA.COM	CONFIG.COM
DEMONSTRATION PROGRAMS	**RUN-TIME SUBROUTINES**	**UTILITY PROGRAMS**
PI.CBL STOCK1.CBL STOCK2.CBL	CALL.ASM CALL.HEX CALL.PRN	FILEMARK.COM

If your issue disk does not include these items, refer to your CIS COBOL Dealer. Note that files required with ANIMATOR are supplied only if ANIMATOR is purchased, see the ANIMATOR Operating Guide.

THE COMPILER

The CIS COBOL Compiler has several overlays and loads each overlay file from the logged-in drive. The root segment is contained in COBOL.COM and the overlays are contained in the other COBOL files.

THE RUNTIME SYSTEM

The Run Time System (RTS) executes the intermediate code output from the compiler. In addition to standard ANSI COBOL statements, CIS COBOL contains many extensions for use with interactive programs. In order to take advantage of these extensions it is necessary to configure the Run Time System for the CRT conventions to be used, if this is not a standard ADM-3.

CONFIGURATOR

The RTS can be configured to include subroutines written in assembler language. The CONFIG utility program is used to reserve an area within the run-time system into which the user may enter assembler or other language subroutines for use by the CALL Statement in a CIS COBOL program.

THE DEMONSTRATION PROGRAMS

PI.CBL, STOCK1.CBL and STOCK2.CBL are simple demonstration programs, supplied in source form, which show many of the facilities present in CIS COBOL, and which can also be used by newcomers to familiarize themselves with the system.

THE RUN-TIME SUBROUTINES

These modules are supplied to provide an example of the use of the COBOL CALL facility to implement run-time sub-routines (See Chapter 4). Copies of the list files can be found in the Run-Time subroutine appendices at the back of this manual.

FIRST STEPS

Initialization

Initialize and format system disks as required (see DISK UTILIZATION below) and COPY THE CONTENTS of the Issue disk to become a working CIS COBOL system.

Disk Utilization

CIS COBOL is designed to take full advantage of two-drive flexible-disk systems, or systems with hard disk facilities.

Where two flexible-disk drives are available for compilation and running, it can be beneficial to copy the compiler to one system disk and the Run Time System (RTS) to another. By default the intermediate code is output to the disk containing the source at compilation and if, therefore, this also contains the RTS, the program can immediately be run. It is the user's responsibility to decide on the most efficient disk allocation for this system.

Compilation

Compile all the demonstration programs. These are source files and have the extension .CBL.

EXAMPLE:

```
A>COBOL STOCK1.CBL<<
**CIS COBOL V4.5 COPYRIGHT (C) 1978, 1982 MICRO FOCUS LTD
**COMPILING STOCK1.CBL
**ERRORS=00000 DATA=00661 CODE=00235 DICT=00409:nnnnn/nnnnn GSA FLAG = OFF
ppppp

A>
```

Note

All the examples in this manual assume that the CIS COBOL software diskette is loaded in drive A. If the diskette was loaded in drive B, the first line in the above example would be:

```
B>COBOL STOCK1.CBL<<
```

After compilation, a directory listing of the disk will show that two new files exist, namely STOCK1.LST which is the list file, and STOCK1.INT which contains the intermediate code. Similar procedures should be followed for STOCK2.CBL and PI.CBL.

Note that STOCK2 has an error in it which is present to show error formats and is for demonstration purposes only. It does not affect the running of the program.

The message produced by the error is:-

```
nnnnnn        MOVE GET-INPUT TO TF-DATE.
**103*******************                                    *******
```

Note

If the file COBOL.MSG is available on the same drive as the compiler, then a textual diagnostic is printed on the listing and also displayed on the console, for each error found by the compiler.

Running The Demonstration Programs

Assume that the Run Time System is now configured and has been renamed RUN. This will typically be the case on a configuration with one CRT. Where there is more than one CRT, it is a good idea to follow RUN with letters to identify the particular CRT (eg RUND for the Apple Datamedia). RUN will be used as the norm in this manual. When these programs have been compiled and run, you have checked out your disk and have mastered the fundamentals of CIS COBOL facilities.

NOTE: In the Appendices G through L the Run Time System is shown with the file-name RUNA.COM which is the file-name on the issue disk.

Calculation of π (PI)

```
A>RUN PI.INT<<

CIS RTS V4.5 COPYRIGHT(C) 1978, 1982 MICRO FOCUS LTD.  URNXX/nnnn/XX
```

This clears the screen, followed by -

```
CALCULATION OF PI

NEXT TERM IS 0.000000000000

   PI IS  3.141592653589

A>
```

During the execution of PI the next term changes as the iteration progresses.

Stock Control Program One (Cursor Control)

```
A>RUN STOCK1.INT<<

CIS RTS V4.5 COPYRIGHT(C) 1978, 1982 MICRO FOCUS LTD.  URNXX/nnnn/XX
```

This clears the screen, followed by -

```
     STOCK CODE     <    >
     DESCRIPTION    <                        >
     UNIT SIZE      <    >
```

This is a skeleton stock data entry program in which stock records are created on a stock file in stock code order. It allows the cursor control functions to be checked out. The operator has the ability to "tab" the cursor forwards and backwards from one data input field to the next. The cursor may be moved backwards and forwards non-destructively one character position at a time in data input fields. It may also be HOME to the first character position in the first data input field. In addition there is a numeric validation on numeric fields which permits only numeric characters to be entered, and an automatic left zero fill on numeric fields. (See CURSOR CONTROLFACILITIES in Chapter 3 for cursor control keys on the standard CRT)

It also creates an indexed sequential file on disk called STOCK.IT together with its index called STOCK.IDX.

To create a record, key the data into the unprotected areas defined by < >. When a record is complete, press the RETURN key and the record will be written to disk. The unprotected areas will then be space filled ready for the next record to be entered, if the record has been correctly entered. If the record remains displayed, the record was incorrectly keyed.

To terminate the run, enter spaces into the STOCK CODE field and press RETURN.

This results in:

```
END OF PROGRAM
```

Stock Control Program Two (Data Input)

```
A>RUN STOCK2.INT<<

CIS RTS V4.5 COPYRIGHT(C) 1978, 1982 MICRO FOCUS LTD.  URNXX/nnnn/XX
```

This clears the screen followed by -

```
            GOODS INWARD

      STOCKCODE       <    >
      ORDER NO      <       >
      DELIVERY DATE MM/DD/YY
      NO OF UNITS    <     >
```

This is a skeleton stock data input program in which the stock records created by STOCK1 can be accessed.

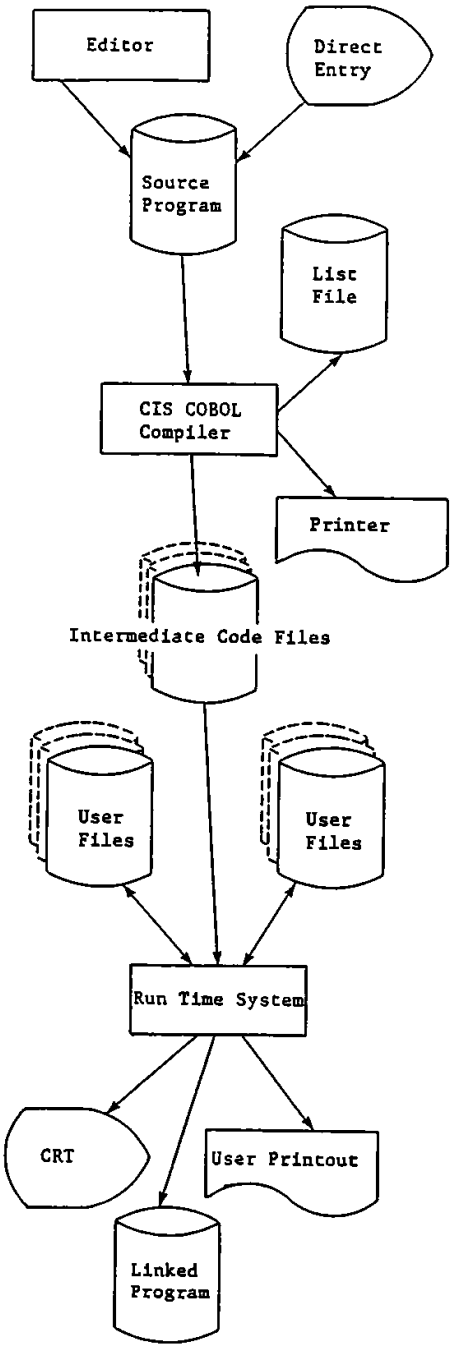
The same cursor control features are present as in STOCK1.INT. Note that the DELIVERY DATE has a different method of prompting than has so far been used.

Terminate in the same way as for STOCK1.

PROGRAM DEVELOPMENT CYCLE

The cycle for development and running of CIS COBOL application programs that must be performed by the programmer is as shown in Figure 1-1.

Figure 1.1. Program Development Cycle.



PREPARATION:

The source programs are created on diskette with the user's own existing editor program, or can be keyed in directly to the compiler using the CRT.

COMPILATION:

COBOL PROG.SRC...

... Loads the single pass compiler to convert a source program (PROG.SRC in this example) into an interpreted object form known as Intermediate Code (PROG.INT). The user may specify the file on which the listing will appear. If this is a disk file, it may be edited to correct errors and used as input for the next run of the compiler.

RUNNING:

RUN PROG.INT...

... Loads the run-time system which in turn loads the Intermediate Code. To aid debugging, the CIS COBOL interactive debugging facility is available. This allows the user to set break points, examine and modify locations and then continue execution. Once loaded the programs run to process the user files as required by the application and controlled by the Operator through the CRT.

Once the user program is fully tested it may be permanently linked to the run-time system by use of the "=" option. See Chapter 3.

PROGRAM PREPARATION CONSIDERATIONS

The CIS COBOL compiler normally accepts source input from a standard source file (specified on the compiler command line) as produced by the CP/M "ED" Editor or compatible proprietary editor software.

The CIS COBOL program format is as specified for standard COBOL and is detailed in the *CIS COBOL Language Reference Manual*.

NOTES:

1. Each line of source code must be terminated by a Carriage Return/Line Feed character pair, including the last line.

2. The compiler will reject most non-alphanumeric characters within the input file, e.g. the Tab character, unless embedded in literal strings.

PROGRAM DESIGN CONSIDERATIONS

CIS COBOL provides the full COBOL facilities for overlaying in memory and for invoking programs (dynamically) or subroutines whether written in COBOL or assembler languages, as specified in the COBOL modules Segmentation and Inter-Program Communication. Chapter 4 contains more information on the use of these features.

Chapter 2. COMPILER CONTROLS

COMMAND LINE SYNTAX

The command line format is:

```
COBOL filename [directives]<<
```

COBOL is the name of the file which contains the compiler

filename is the optional name of the program which contains the CIS COBOL source statements. If the filename is not given, the console is taken as the input file.

directive is an optional sequence of CIS COBOL directives that can be specified in any order. Each directive must be separated by one or more spaces. If the sequence is too long to fit on one line of the screen then it may be continued on a subsequent line by typing an ampersand sign "&" followed by carriage return. A particular directive may be on one line only. Where directives have brackets the left-hand bracket may occur zero, one or more spaces after the body of the directive. To terminate the sequence, press return.

COMPILER DIRECTIVES

A description of each of the available compiler directives follows:

FLAG (level)

This directive specifies the output of validation flags at compile time. The parameter "level" is specified to indicate flagging as follows:

LOW Produces validation flags for all features higher than the Low Level of compiler certification of the General Services Administration (GSA).

L-I Produces validation flags for all features higher than the Low-Intermediate level of compiler certification of the GSA.

H-I Produces validation flags for all features higher than the High-Intermediate level of compiler certification of the GSA.

HIGH Produces validation flags for all features higher than the High Level of compiler certification of the GSA.

CIS Produces validation flags for only the CIS COBOL extensions to standard COBOL as it is specified in the ANSI COBOL Standard X.23 1974. (See the *CIS COBOL Language Reference Manual*)[1].

NOFLAG

No flags are listed by the compiler. This is the default if the FLAG directive is ommitted.

RESEQ

If specified, the compiler generates COBOL sequence numbers, re-numbering each line in increments of 10. The default is that sequence numbers are ignored and used for documentation purposes only, i.e., NORESEQ.

[1] Up to version 4.4, the *FLAG (level)* directive was called the *ANS switch*. On older versions of the compiler, use `ANS` as substitute for `FLAG CIS`.

NOINT

No intermediate code file is output. The compiler is in effect used for syntax checking only. The default is that intermediate code is output, i.e., INT (sourcefile.INT).

NOLIST

No list file is produced; used for fast compilation of "clean" programs. The default is a full list, i.e., LIST (sourcefile.LST).

COPYLIST

The contents of the file(s) nominated in COPY statements are listed. The list file page headings will contain the name of any COPY file open at the time a page heading is output. The default is NOCOPYLIST.

NOFORM

No form feed or page headings are to be output by the compiler in the list file. The default is headings are output, i.e., FORM(60).

ERRLIST

The listing is limited to those COBOL lines containing any syntax errors or flags together with the associated error message(s). The default is NOERRLIST.

INT (external-file-name)

Specifies the file to which the intermediate code is to be directed. The default is: source-file.INT.

LIST (external-file-name)

Specifies the file to which the listing is to be directed (this may be a printing device, ie. console or printer or a disk file) The default is: source-file.LST.

For list to console use: LIST(CON:) or LIST (:CO:)

For list to line printer use: LIST(LST:) or LIST (:LP:)

FORM (integer)

Specifies the number of COBOL lines per page of listing (minimum 5). The default is 60.

NOECHO

Error lines are echoed on the console unless this directive is specified. The default is ECHO.

NOREF

Suppresses output of the 4-digit location addresses on the right hand side of the listing file. REF is the default.

Note

Using the directive combination

NOREF NOFORM RESEQ

is a useful way of numbering your CIS COBOL source program. Running the compiler with this combination results in a list file that is an exact duplicate of your source file, with the sequence number field in columns 1 - 6 in numerical order from 000010 in upward increments of 10. An extra three lines are appended at the end of the source code but these are ignored by the compiler if represented in the source. The user can, of course, delete these extra lines using the system editor software.

DATE (string)

The comment-entry in the DATE-COMPILED paragraph, if present in the program undergoing compilation, is replaced in its entirety by the character string as entered between parentheses in the DATE compiler directive. This date is then printed at the top of every listing page under the filename.

QUIET

This directive causes the error text diagnostic messages not to be produced - leaving only the error number messages in the listing. The default is NOQUIET, which allows error text messages to be listed.

PAGETHROW (character-code)

Specifies the ASCII character code for physical page throw on the printing device. The character code is expressed in decimal, and the default is PAGETHROW (12).

ANIM

The ANIM directive compiles the program in such a way as to enable run-time debugging with the ANIMATOR product and should not be specified if you do not have this product. See Chapter 7. Note that the compiler produces three new ANIMATOR files for your program in addition to the intermediate code file (.INT) and any listing (.LST) with the extensions .SDB, .SCP and .DDC respectively. Default is obviously NOANIM. This directive is only for use when compiling programs for later debugging with the ANIMATOR product.

The remaining Compiler directives are only for use when compiling programs to run under the FILESHARE file management system product.

FILESHARE

This directive informs the compiler that the program being compiled contains extended syntax statements that can be used only with the optional FILESHARE product. (See the FILESHARE Users Guide). Without the directive, FILESHARE syntax will be flagged as being in error, and further FILESHARE compile directives (see below) will not be accepted.

RESTRICT (organization)

Categorises all files with the organization specified - "INDEXED" or "RELATIVE" - declared within the program being compiled, as being of type Exclusive access. The default file type is Unrestricted, but not Committable, (See FILESHARE above).

COMMIT (organization)

Categorises all files with the organization specified - "INDEXED" or "RELATIVE" - declared within the program being compiled, as being of type Committable, but not Resettable, (See FILESHARE above).

DERESTRICT (organization)

Categorises all files with the organization specified - "INDEXED" or "RELATIVE" - declared within the program being compiled, as being of type Unrestricted, but not Committable, (See FILESHARE above).

Note

A program containing FILESHARE syntax statements may be compiled using the FILESHARE directive and will run and can be tested in isolation using a single-user RTS.

EXCLUDED COMBINATIONS

Certain of these directives may not be used in combination. Table 2-1 shows the directives that are excluded if the directive shown adjacent in the left hand column is specified

Table 2.1. Excluded Combinations of Directives

DIRECTIVE	EXCLUDED DIRECTIVES
NOLIST	LIST NOFORM FORM RESEQ COPYLIST ERRLIST NOREF
ERRLIST	RESEQ COPYLIST NOREF

SUMMARY INFORMATION ON CRT

The general format of the basic command line is:

```
COBOL filename [directives]<<
```

and the Compiler will reply with:

```
**CIS COBOL V4.5 COPYRIGHT (C) 1978, 1982 MICRO FOCUS LTD.
```

where 4 is the version number and 5 is the release number.

Each directive is then acknowledged by the Compiler on a separate line, and is either ACCEPTED or REJECTED. After all the directives have been acknowledged, the Compiler opens its files and starts to compile. At this point it will display the message:

```
filename COMPILING
```

If any file fails to open correctly, the Compiler will display:

```
filename FAILED TO OPEN
```

The compilation will be aborted, returning control to the operating system.

When the compilation is complete the Compiler displays the message:

```
**ERRORS=nnnnn DATA=nnnnn CODE=nnnnn DICT=mmmnun:nnnnn/ppppp  GSA FLAGS=nnnnn
```

where:

ERRORS	denotes the number of errors found
DATA	denotes the size of RAM required i.e. data area of the generated program
CODE	denotes the size of ROM required i.e. code area of the generated program
DICT	mmmmm denotes the number of bytes used in the data dictionary. nnnnn denotes the number of bytes remaining in the data dictionary. ppppp denotes the total number of bytes remaining in the data dictionary.
GSA FLAGS	denotes the number of compiler validation flags encountered or 'OFF' if the directive NOFLAG was entered or assumed.

LISTING FORMATS

The general layout of the list file is as follows:

```
**CIS COBOL V4.5 filename      PAGE: nnnn
**
** OPTION SELECTED
** - optional directives as entered in compile command line -
**
statement 1                                                          HHHH
     .                                                             .
     .                                                             .
     .                                                             .
statement n                                                          HHHH

**CIS COBOL V4.5 REVISION n                                 URN AA/0300/BA
**COMPILER COPYRIGHT (C) 1978, 1982 MICRO FOCUS LTD
**
**ERRORS=nnnnn DATA=nnnnn CODE=nnnnn DICT=mmmrnm:nnnnn/ppppp GSA FLAGS=nnnnn

END OF LIST
```

The first two lines of title information are repeated for each page. The final line is the same as on the CRT display. The value denoted by HHHH is a hexadecimal value denoting the address of each dataname or procedure statement. Addresses of datanames are relative to the start of the data area, while addresses of procedure statements are relative to the start of the code area (There is an overhead at the start of the data area, and a few bytes of initialization code at the start of the procedure area for each SELECT statement).

A syntax error is marked in the listing by an error line with the following format:

```
nnnnnn illegal statement
** nnn ***            ... ***                      *******
```

where

nnnnnn is the sequence number of the erroneous line

nnn denotes the error number

The asterisks following the error number indicate the character position of the error in the preceding erroneous source line. The asterisks at the end of the line simply highlight the error line.

Note

The sample program STOCK2 compiled as described under Compilation in Chapter 1 contains a sample error line.

A flag is marked in the listing by a flagging line with the following format:

```
nnnnnn                  flagged feature
** level ---           ...      ... ----    -------
```

where

nnnnnn is the sequence number of the flagged line.

'level' represents the level at which the feature is flagged using the same acronyms as can be entered in the command line (when setting the lowest required flagging level):

LOW - Low level

L-I - Low-Intermediate level

H-I - High-Intermediate level

HIGH - High level

CIS - CIS COBOL extensions

The flagged feature is pinpointed at the position of the end of the line of characters beneath the flagged line. The dashes at the end of the line simply highlight the flagging line.

Note

A program in which flags are indicated can still be run. Errors should always be corrected, however, and the program recompiled before the object program is run.

Chapter 3. RUN-TIME SYSTEM CONTROLS

RUN-TIME DIRECTIVES

COMMAND LINE SYNTAX

The command line syntax for running a CIS COBOL object program is as follows:

RUN [-V] [load param] [switch param] [link param] filename [program params]

filename is the name of the intermediate code file. File and device conventions for CP/M are given in Appendix F. RUN must have at least one space keyed after it, and filename must have either a space or RETURN keyed after it. The parameters need not have spaces keyed after them. An example of the whole RUN command line is given later in this Chapter.

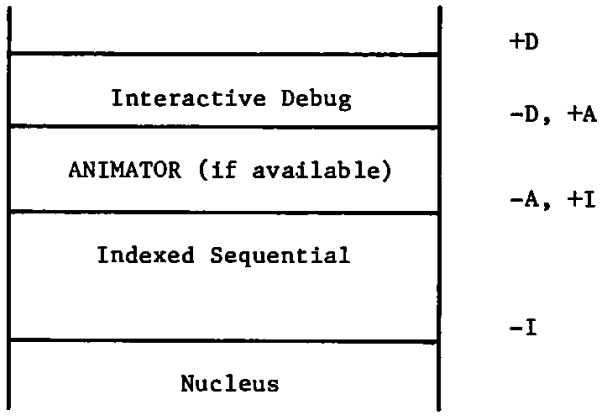
-V (Version) Parameter

The -V parameter inhibits version compatibility checking between the object code (intermediate code) being run and the V4.5 run-time system. (By default, only intermediate codes produced by the V4.5 compiler may be run by the V4.5 run-time system.) Error 165 will result if -V is not included, and the int. code was not the product of the V4.5 compiler. Intermediate code produced by the CIS COBOL compilers V4.3 or V4.4 can be run on the V4.5 run-time system using this directive.

Load Parameter

The optional load parameter provides the Run Time System loader with the load point for the intermediate code in memory. The user has the option to overlay optional modules to conserve program space. Additionally the CIS COBOL Interactive Debug may be invoked. The memory layout of the Run Time System (RTS) is as shown in Figure 3-1.

Figure 3.1. Run Time System Memory Layout.



The default load position excludes the Debug and ANIMATOR modules but implies that Indexed Sequential is included. The Debug module may be included and invoked by using the parameter “+D”.

The +A parameter invokes the ANIMATOR product and can be used only if you have the ANIMATOR product. See Chapter 7.

To exclude the Indexed Sequential package and the optional modules above it (see Figure 3-1), the parameter “-I” should be given.

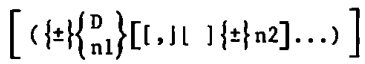
Table 3-1 shows which optional modules will be loaded for the available parameters.

Table 3.1. Optional Modules by Load Parameter.

Load Parameter	Optional Module Included			
	Debug	ANIMATOR	Indexed Seq.	RTS only
+D	Yes	Yes	Yes	Yes
-D or +A	No	Yes	Yes	Yes
-A or +I	No	No	Yes	Yes
-I	No	No	No	Yes

Switch Parameter

CIS COBOL includes the facility of controlling events in a program at run time depending on whether or not programmable switches are set by the operator. See the description of the SPECIAL-NAMES paragraph in the *CIS COBOL Language Reference Manual*. The operator sets these switches at run time by use of the Switch Parameter to the RUN command. The general format of the Switch Parameter is:



where:

[ ] denotes an optional item

{ } denotes a choice

n1 and n2 are any numbers in the range 07. They can be specified in any order and the last appearance of any specific number takes precedence.

D see Standard ANSI COBOL Debug Switch Parameter below

+ or - set the switch n1, n2, etc. on or off respectively. The default is that all switches are off initially.

... denotes that the preceding options enclosed in the outermost brackets can be repeated.

See EXAMPLES later in this Chapter.

Standard ANSI COBOL Debug Switch Parameter

Users may also include a parameter to invoke the standard ANSI COBOL Debug module, whether or not the CIS COBOL Interactive Debug extension to ANSI COBOL is invoked. (See the *CIS COBOL Language Reference Manual* for a description of the Debug facilities).

To include the standard ANSI Debug facility a Run Time switch is required. The format is as for a normal switch parameter (see Switch Parameter above), but the numeric switch character is replaced by D. See also EXAMPLES later in this chapter.

Note

This facility cannot be invoked if ANIMATOR is in use, i.e., the +A parameter has been entered.

Link Parameter

When the program is fully tested it may be linked with the Run Time System to produce an executable program that can be directly loaded. This is achieved by including the parameter "=" to the Run Time System (see the EXAMPLE overleaf). When the intermediate code file has been loaded (following the lines above) a binary file with the filename SAVE is produced from the current store image. It is essential to rename the SAVE file, from which to load directly, to prevent it being overwritten on the next use of '=' parameter. The RENname command is used for this, and the new file-name must be of the form:

```
filename.COM
```

See the CP/M operating documentation for the RENname command.

Note

Programs cannot be linked if the ANIMATOR is in use (ie., parameters +a and = are mutually exclusive).

Program Parameters

These are any parameters required by the program, they can be read in on the console file device :CI: or CON:.

COMMAND LINE EXAMPLES

1. The directive

   ```
   RUN B:PROG.INT 1 2<<
   ```

   loads the program PROG from the intermediate file produced by the compiler and passes the user program parameters 1 and 2 to the program PROG, where they are accessable to the ACCEPT statement (See the *CIS COBOL Language Reference Manual*).

2. The directive

   ```
   PROG<<
   ```

   loads the PROG program but omits those options omitted when PROG was linked (PROG must have been previously linked by the "=" link parameter.)

   If it is required to load the sample program STOCK1 in future, instead of the RUN command given in Chapter 1 (A>RUN STOCK1.INT), the following command could be entered:

   ```
   RUN = STOCK1.INT<<
   ```

   followed by the RENname command:

   ```
   REN STOCK1.COM=SAVE<<
   ```

   In subsequent loads only the command STOCK1<<would then be required.

3. The directive

```
RUN +D (+1+2,+3) = PROG.INT<<
```

loads the program PROG with interactive CIS COBOL Debug and the Indexed Sequential module. Programmable switches 1, 2 and 3 are set, and a binary file of the program PROG is created, which can subsequently be loaded directly. A SAVE file is created and the Interactive CIS COBOL Debug initial display will appear on the CRT when the saved binary PROG is run.

4. The directive

```
RUN (-2 +5-7+7) PROG.INT<<
```

loads the program PROG from the intermediate file produced by the compiler, without Interactive Debug and with programmable switches 5 and 7 on and 2 off. Note that the last setting of switch 7 is accepted. Switches 1, 3, 4 and 6 are off by default.

Note

An overlayed program always expects the overlays to be in the logged-in drive. Disks in other drives are not searched for overlays.

5. The directive

```
RUN (+D) PROG.INT<<
```

loads the program PROG from the intermediate code file produced by the compiler with the standard COBOL ANSI DEBUG module invoked, but omitting CIS COBOL Interactive Debug.

6. The directive

```
RUN +D (+2,+4 +D) PROG.INT<<
```

loads the program PROG with Interactive CIS COBOL Debug and with programmable switches 2 and 4 set, and with the standard ANSI COBOL DEBUG module invoked.

Warning

NEVER TERMINATE A PROGRAM RUN BY POWERING DOWN OF THE COMPUTER SYSTEM, PARTICULARLY IF THE PROGRAM CONTAINS DISK FILE PROCESSING.

INTERACTION IN APPLICATION PROGRAMS

CRT SCREEN HANDLING

COBOL is traditionally a batch processing language; CIS COBOL extends the language to make it interactive. CIS COBOL offers many facilities for automatic formatting of a CRT screen and facilitates keying of input.

The CIS COBOL programmer can specify areas of the screen into which the operator is able to key data, and also whether such data is numeric or alphanumeric. This is achieved by defining the screen as a record in the DATA DIVISION in which the data fields correspond to the input area and FILLER's correspond to the rest of the screen.

An ACCEPT statement nominates a record description, which permits input to the character positions corresponding to variables identifed by data-names. Conversely, a DISPLAY statement outputs only

from non-FILLER fields in the record description which it nominates. The programmer can thus easily build up complex conversations for data entry and transaction processing.

While data is being keyed, the operator has full cursor manipulation facilities, each variable acting as a tab stop. Non-numeric digits may not be entered into fields defined as numeric. Finally, when the operator has checked that the data is correct, the RETURN key is depressed and the data becomes available to the program. Because all characters are transferred to the appropriate area as they are keyed in there is no transmission delay.

Screen Layout and Format Facilities

The following facilities are available for screen layout and formatting:

- Screen as a record description
- FILLER
- REDEFINES
- AT line column
- CURSOR addressing
- Character highlighting (if available on the CRT in use)
- Clear screen
- Numeric validation of PIC 9(n) fields
- Automatic editing of numeric edited data-items
- De-editing of numeric edited to numeric data-items

Cursor Control Facilities

During execution of ACCEPT statements the cursor is manipulated on the CRT screen by the cursor control keys on the console keyboard as shown in Table 3-2.

Table 3.2. CRT Cursor Control Keys

Function	**Keys**[1]
Home (referred to as # or HOM in this manual)	Ctrl ↓
Tab forward a field	↓
Tab backward a field	↑
Forward space	→
Backward space	←
Column Tab	TAB
Left Zero[2]	.
Return	RETURN

1 - Where CTL is specified the operator must press the CTL key hold it down and simultaneously press the character key. Back one space for ADM3A is thus both the CTL and the H character keys.

2 - The "." for left zero fill is a “,” when

```
   DECIMAL-POINT IS COMMA
```

Function	Keys[1]
is specified in the user program	

INTERACTIVE DEBUGGING

Two levels of debugging are available to the programmer. The first involves optional "debugging lines" that are included if the "DEBUGGING MODE" switch is present in the "SOURCE-COMPUTER" sentence. The second is the interactive Debug package that is included at run-time under the control of the user (see Switch Parameter in this Chapter).

If Debug is included in the RTS, it will announce its presence when the program is loaded as follows:

```
RUN +D STOCK1.INT<<

CIS RTS V4.5 COPYRIGHT (C) 1978, 1982 MICRO FOCUS LTD URN XX/nnnn/XX

Debug Mark 3.1              title

?                           prompt
```

The user now has the following commands available:

P	Displays the current program counter (p-c).
G	Breakpoint at specified address.
X	Execute one CIS COBOL statement at a time.
D	Display bytes in the Data Division
A	Replace contents of a memory location by a hexadecimal value or ASCII character.
S	Set start of block for correction or display.
/	Display bytes in block above.
.	Change bytes in block above.
T	Trace paragraphs up to breakpoint specified.
L	Output one CR LF on the CRT
M	Define Debug command macro with name specified
$	End macro definition
C	Displays specified character on the CRT
;	Precedes comment to describe a macro just entered.

A description of the use of each of these Debug commands follows.

THE P COMMAND

The P command displays the address at which the program counter (p-c) currently points i. e, where the current instruction is in the Procedure Division code of a program. This hexadecimal address is that printed in the right hand column of a program listing.

EXAMPLE:

At the start of a program the p-c is at 0000 as shown below:

```
?P<<            -command
```

```
0000                -current p-c
?                   -prompt
```

NOTES:

1. The location given by the 'P' command is relative to the start of the PROCEDURE DIVISION. All numbers in the Debug package are expressed as hexadecimal values.

THE G COMMAND

The G command executes from the current p-c until the p-c reaches the value in the parameter to 'G'. If this value is not the address of an executed instruction, the breakpoint is never reached and the program continues.

EXAMPLE:

If a breakpoint is required at PARA-22 in the following code:

```
          .                              .
          .                              .
     PARA-22.                            017A
          ADD 1 TO COUNT.                017B - hex address
          MOVE FIELD-1 TO FIELD-2.       018C
          .                              .
          .                              .
```

the following command is typed:

```
?G 017A<<
?
```

The display of the second question mark above indicates that the G command has executed completely and thus the breakpoint has been reached.

NOTES:

1. Exactly four hexadecimal digits must be keyed for an address value.

   A check on the current address at this point by use of the P command would be as follows:

```
?P<<
 017A                -returns p-c
```

2. The command G 0000 can be used to cause a breakpoint on entry to the next called subroutine.

THE X COMMAND

When a suspected error is reached, single instructions can be stepped through one at a time by use of the 'X' command. After each COBOL instruction is executed, the hexadecimal number in the right-hand column is the address of the first statement on a line. Where COBOL operations are made up of several individual primitive instructions, DEBUG may appear to halt in the middle of a line. If this occurs, the RETURN is pressed again.

EXAMPLE:

If an error occurred in the MOVE instruction the X command sequence would be shown as follows :

```
?X<<
 018C
?
```

To check the contents of "FIELD-2" before and after the move for code in the "DATA DIVISION" the display would be:

```
02   FIELD-1 PIC XXX VALUE "ABC".     0030
02   FIELD-2 PIC XXX VALUE "XYZ".     0033
02   FIELD-3 PIC X(8O) VALUE SPACE.   0036
     .                                .
     .                                .
```

THE D COMMAND

To display bytes in the DATA DIVISION, the 'D' command can be used. This displays 16 bytes from the address specified (again the address is derived from the information on the listing). It displays each byte as a hexadecimal value plus an ASCII equivalent if it is printable.

EXAMPLE:

```
?D 0030<<
41-A 42-B 43-C 58-X 59-Y 5A-Z 20-  20-  20- ........
FIELD-1        FIELD-2        FIELD-3
     ?
```

If the MOVE is then executed and re-examined the following display results:

```
?X<<
 019C
?D 0030<<
41-A 42-B 43-C 41-A 42-B 43-C 20-  20-  20-
```

THE A COMMAND

The "A" command is used to amend data at a specified memory location.

EXAMPLE:

To replace the first character “A” of FIELD-1 by “G”. The value supplied may be a two character hex value or an ASCII character preceded by quote eg "G or 47.

```
?A 0030 47<<                        -amend byte
?D 0030<<
 47-G 42-B 43-C 41-A 42-B 43-C 20-  20-  20- ........
?
```

This correction facility allows continued running even if a bug has produced an erroneous result.

THE S COMMAND

Where a number of corrections are required, DEBUG allows specification of a working register which contains an address. This address can be set or incremented and the contents can be displayed or modified immediately by use of the 'S' command. The address and contents can then be displayed by keying '/'.

EXAMPLE:

To display the first byte of FIELD-1 operation would be as follows:

```
?S 0030<<                    -load address
?/<< -display
 0030 47G
?
```

THE '.' COMMAND

To amend the byte at the current location '.' is used; this also increments the working register.

EXAMPLE:

To change FIELD1 to “DEF” the display would be:

```
?S 0030<<              -load address
?.44.45.46<<           -modify
?D 0030<<
 44-D 45-E 36-F ......
```

To increment only the working register use ','.

THE T COMMAND

An advanced form of the 'G' command is the 'T' command. This also executes up to a breakpoint in the PROCEDURE DIVISION, but also prints the address of each paragraph encountered.

EXAMPLE:

```
?T 017B<<         trace up to 017B
```

Note

The command T 0000 can be used to trace up to the start of the next called sub-program.

DEBUG MACRO COMMANDS

The user will find that some Debug command sequences are used often when debugging. If these sequences are long it can become tiresome typing them in. To overcome this and to allow the development of complex debugging sequences Debug permits the definition of macros comprised both of basic operations and other macros. Macros are given names of one character.

The M Command

Macros are introduced by the 'M' command followed immediately by the macro name.

EXAMPLE:

To define a macro to execute up to 018C, display the value at 0030, then jump by a single instruction and display again; the following would be typed:

```
?MZ G 018C   D0030 L X D 0030 $<<
?
```

To invoke this macro its name is typed as follows:

```
?Z<<
41-A 42-B 43-C 58-X 59-Y 5A-Z ......   First display
0190
41-A 42-B 43-C 41-A 42-B 43-C ......   Second display
```

There are two other commands introduced in this macro: 'L' and '$'.

The L Command

The 'L' command merely forces a carriage return and line feed to be output on the console.

The $ Command

The '$' command ends a macro definition.

The C Command

To allow macro writers to output characters to the console, the command 'C' is provided. This outputs its parameter on the console

EXAMPLE:

```
?C "A<<
A
?
```

The ; Command

To improve readability of macros, comments may be inserted. These are introduced by the character ';' and terminated by carriage return.

EXAMPLE:

```
?MZ D 0030 XL D 0030 $ ; Run macro<<
```

Macro names must be letters only. Lower case letters are converted internally to upper case.

If an error is made in typing in a macro then it may be reentered. However, there is only a finite amount of macro space and space is not released if a macro is reentered. If the space runs out or the maximum nesting of macros is exceeded then the message STACK OVERFLOW will result.

EXAMPLE:

```
?MZ Z$  ; macro to crash system<<
?Z<<
```

After the crash has occurred, the Debug system will return to command mode and will reset the stack to allow the user to continue. However, if more serious crashes occur i.e. those with no message, then the system will not recover.

For full details of Debug commands see Appendix E.

Chapter 4. CIS COBOL APPLICATION DESIGN CONSIDERATIONS

CIS COBOL provides the full COBOL facilities for including programs dynamically and for overlaying in memory and for invoking programs (dynamically) or subroutines whether written in COBOL or assembler languages, as specified in standard COBOL modules Segmentation and Inter-Program Communication.

With these facilities available, large and complex CIS COBOL application programs can be run. System designers in particular should realize that the total size of the application is not constrained by the intrinsic hardware environment. This Chapter describes the use of these facilities.

Details of the CIS COBOL Language elements to include the Inter-Program Communication and Segmentation features are given in the CIS COBOL Language Reference Manual.

CIS COBOL APPLICATION DESIGN FACILITIES

The facilities for Inter-Program Communication, Segmentation and Chaining are summarised below and described in the remainder of this Chapter.

INTER-PROGRAM COMMUNICATION (CALL)

CIS COBOL enables COBOL applications to be divided at source level into separate independent modules. Each module is referred to as a program, in line with ANSI 1974 notation. Programs are called dynamically from a main application program. Programs written in assembler code language can also be called from a main COBOL application program. In both cases control is transferred by the use of the CALL statement which may be used with parameters.

SEGMENTATION (OVERLAYING)

CIS COBOL enables a COBOL program with a large Procedure Division to be broken into a COBOL program with a smaller Procedure Division and multiple overlays providing the remaining Procedure Division. The overlays are known as independent segments. A segmented program can be CALLed as can any program.

CHAINING

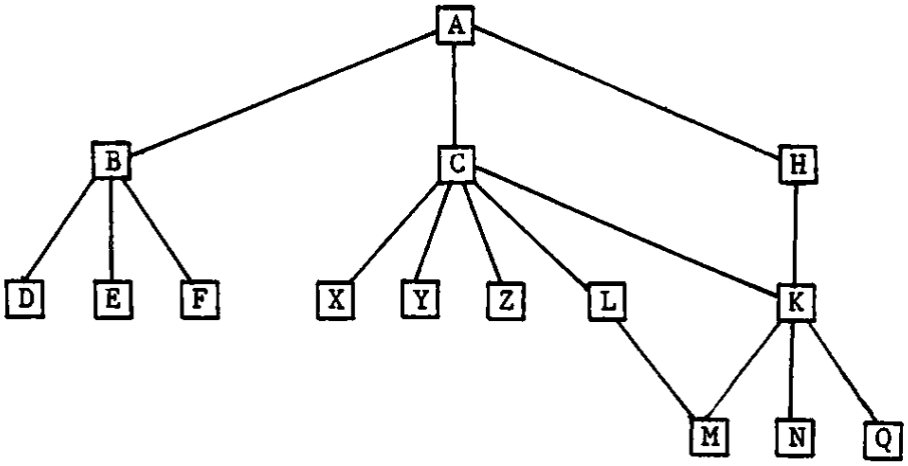
Chaining is a CIS COBOL feature to pass control from a CIS COBOL application to another application or utility. The chained application or utility replaces the original CIS COBOL application in its entirety. The CHAIN facility is a subroutine supplied with the CIS COBOL Run-Time System. See ASSEMBLER SUBROUTINES PROVIDED BY MICRO FOCUS in this Chapter. Control is not returned to the program calling CHAIN.

INTER-PROGRAM COMMUNICATION

By use of the Inter-Program Communication feature, control can be passed from one program to another using the CALL statement and applications can therefore be designed in independent modules or programs.

Figure 4-1 shows a sample application using inter-program communication.

Figure 4.1. Sample CALL Tree Structure.



The main program A which is permanently resident in memory calls B, C, or H which are subsidiary functions and stand-alone functions within the application. These programs call other specific functions as follows:

B calls D, E and F
C calls X Y, or Z conditionally and K or L conditionally.
H calls K.
K calls M N or Q conditionally.
L calls M if it need to.

As the functions B, C and H are stand-alone they do not need to be permanently resident in memory together, and can therefore be called as necessary using the same physical memory when they are called. The same applies to the lower functions at their level in the tree structure.

In the example shown in Figure 4-1, the use of CALL and CANCEL would need to be planned so that a frequently called subroutine such as K would be kept in memory to save load time. On the other hand because it is called by C or H it cannot be initially called without C or H in memory i.e., the largest of C or H should call K initially so as to allow space. It is important also to avoid overflow of programs; see MEMORY LAYOUT in this Chapter, At the "level" of X, Y and Z it does not matter in which order loading takes place because they do not make calls at a lower "level".

Another case for leaving called programs in memory is if they open files. Otherwise these programs would have to re-open the files on every call.

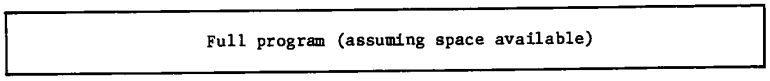
The CIS COBOL Run Unit is an application that is written in CIS COBOL and arranged into a number of separate CIS COBOL programs; these programs communicate with, invoke and cancel each other by use of COBOL "CALL" and "CANCEL" statements.

FORMAT OF CIS COBOL “CALL”

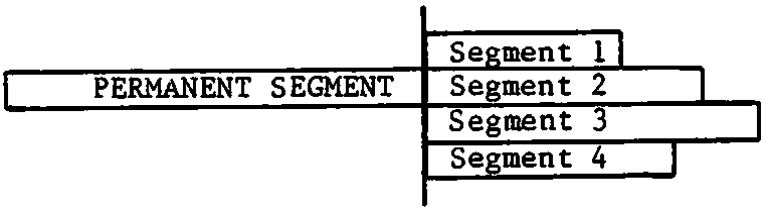
The general format of the CIS COBOL "CALL" and "CANCEL" statements are given in the CIS COBOL Language Reference Manual.

SEGMENTATION

By use of the CIS COBOL Segmentation feature all of the Procedure Division can be loaded into the available memory. Because it cannot, however, be loaded all at once, it is loaded one segment at a time, to achieve the same effect, in the reduced store space as shown below.



In the case of a COBOL segmented program the compiler allows space for the largest segment in that program:



The beginnings of the segments of a Procedure Division of a segmented program are denoted in the CIS COBOL source by a SECTION label, e.g.

```
      .
      .
      .
SECTION 52.
      MOVE A TO B.
      etc.
      .
      .
      .
SECTION 62.
      MOVE X TO Y.
      etc.
      .
      .
      .
```

Segmentation can be applied only to the Procedure Division. The Identification, Environment and Data Divisions are common to all segments; in addition there may be a common Procedure Division. All this common code is known as the Permanent Segment. Control Flow between Permanent and Independent Segments is fully specified in the Language Reference Manual.

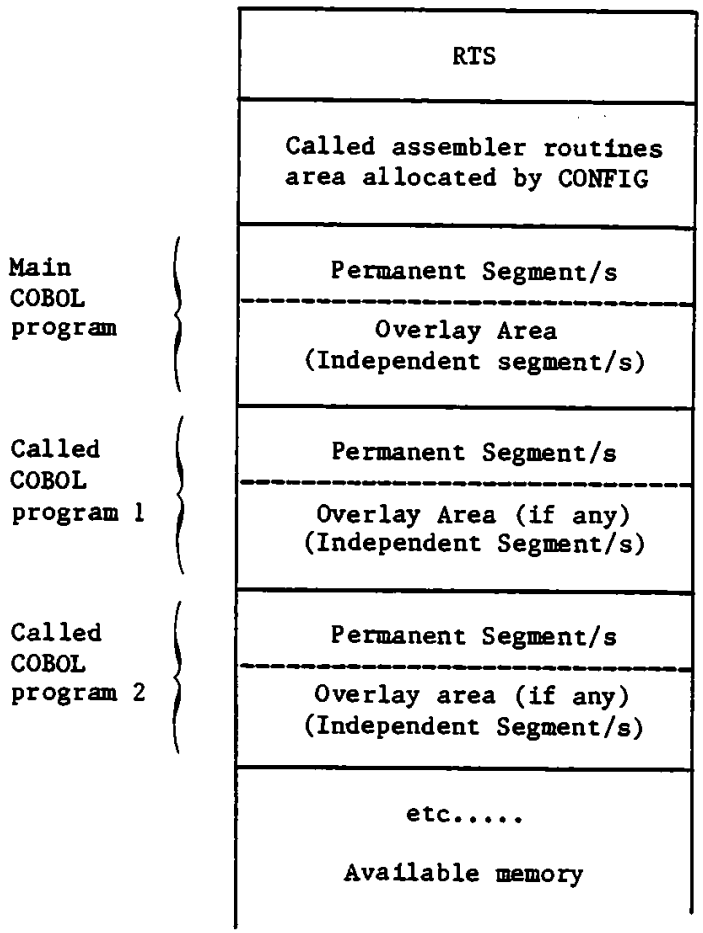
CHAINING

The CIS COBOL program chaining feature can be used to replace an application or utility in memory in its entirety. A CALL is made to the supplied CHAIN utility program which allows another linked program not requiring parameters to be loaded and entered. There is no return to the calling program. The CHAIN routine is described later in this Chapter.

MEMORY LAYOUT

In order to consider the use of overlaying (Segmentation) and/or multilanguage calling of other programs together, it is useful to consider the memory layout. Assuming that both features are in use Figure 4-2 shows the memory layout.

Figure 4.2. Memory Layout using Segmentation and Inter-Program Communication.



It can be seen in figure 4-2 that called programs are loaded contiguously. If however a program is cancelled the memory is made available for another called program. Planning of the use of CALL is therefore required to ensure that space is available. When a program is loaded it is always placed in the largest contiguous area of unused memory. Care is needed in the design of CALL/CANCEL sequences as fragmentation of the total available space in memory for loading into can occur due to inappropriate design.

Figure 4-2 also shows that there is one fixed area of memory allocated by CONFIG for called Assembler subroutines; see Chapter 5.

OPERATIONAL FEATURES

Each COBOL program in a CIS COBOL application suite, with the exception of the main program, should have a Linkage Section in the Data Division through which to communicate with COBOL programs that call them.

All CIS COBOL programs other than the main program must be compiled and their intermediate code placed in disk files which are accessed at run time. The main program may be in intermediate code and named as a parameter to RUN, or it may be linked to RUN in the manner described earlier under RUNTIME DIRECTIVES.

Any number of COBOL programs and assembler code subroutines can be CALLed from a COBOL program. Operational features of CALL are as follows:

1. The CALLed intermediate code program file must be present on disk at the time of the first CALL to the file or fatal error 164 will result.
2. There must be room available in memory for the program to be loaded. The ON OVERFLOW phrase can be used to specify program action if insufficient space is available. Otherwise the CALL statement is ignored and the next calling program instruction is performed.
3. Run-time Subroutines must be preconfigured into the RTS.
4. Disks can be changed during or at run time by suitable user-programmed operator messages and actions. Under CP/M the changed drive will then become READ only (i.e. accessable only for input.)
5. The CANCEL statement reclaims unused storage when executed at run time.
6. No more than seven programs can have been called concurrently.

If a tree structure of called independent programs is used as shown earlier, each segment can call the next dynamically by using the technique shown in the following sample coding:

```
WORKING-STORAGE SECTION.

01   NEXT-PROG      PIC X(20) VALUE SPACES.
01   CURRENT-PROG   PIC X(20) VALUE "1STPROG.INT".
PROCEDURE DIVISION.
LOOP.
     CALL CURRENT-PROG USING NEXT-PROG.
     CANCEL CURRENT-PROG.
     IF NEXT-PROG = SPACES STOP RUN.
     MOVE NEXT-PROG TO CURRENT PROG.
     MOVE SPACESTO NEXT PROG.
     GO TO LOOP.
```

The actual programs to be run can then specify their successors as follows:

```
.
.
.
LINKAGE-SECTION.
01   NEXT-PROG PIC X(20).
.
.
.
PROCEDURE DIVISION USING NEXT-PROG.
.
.
.
.
.
MOVE "SUCCESOR.INT" TO NEXT-PROG.
EXIT PROGRAM,
```

It can be seen that in this way each independent segment or sub-program cancels itself, and changes the name in the CALL statement to call the next one by use of the USING phrase.

RUN TIME COBOL PROGRAM LINKAGE

Run-time execution of the COBOL verb CALL depends on the argument used by the CALL.

When the subroutine or subprogram is in COBOL, the parameter is an alphanumeric quantity whose value is interpreted as a file-name and the appropriate file of intermediate code is loaded from disk into memory and executed.

When the subroutine is configured into the RTS for the main program (See RUN-TIME SUBROUTINES - CALL in this Chapter), the CALL parameter is a numeric quantity, its value is interpreted as the linkage number to the Run Time subroutine table and the corresponding machine code subroutine is executed.

EXAMPLE LINKAGE

```
PROCEDURE DIVISION
      .
      .
      .
CALL "A:SUBITM.INT" USING ...
      .
      .
      .
CALL "10" USING ...
      .
      .
      .
```

For the first CALL in this example to perform correctly the file SUBITM.INT must be present on disk unit A and must contain a compiled COBOL program. For the second CALL to -perform correctly the RTS must contain an assembler subroutine (Run-Time subroutine) arranged as subroutine 10. A description of run-time subroutine inclusion follows.

RUN-TIME SUBROUTINES (IN ASSEMBLER OR NON-COBOL LANGUAGES)

The run-time system is designed in such a way that the user may write and include assembled or other language subroutines that can be accessed using the COBOL "CALL" verb. (See the Appendix on example use of this facility at the back of this manual).

RESERVING SPACE FOR RUN-TIME SUBROUTINES

To reserve space in the run-time system for User Subroutines, it is necessary first of all to run the CONFIG program (see Chapter 5) to direct it to reserve the space and, from it, to obtain the absolute address at which the code is to be placed, (See also Appendix K).

FORMAT OF RUN-TIME SUBROUTINE AREA

The code is now created, ensuring that an 'ORG' is placed at its head to position the code at the correct place in store as specified by the configuration utility. The code is entered using any CP/M editor software, then assembled and finally linked at this address using the CP/M DDT linker facility.

Each Subroutine is identified by an integer as in the example in Appendix M (CALTOP).

The first part of the Subroutine area must consist of a table of addresses as follows:-

BYTE 0	Highest subroutine number which is available
BYTE 1+2	Address of routine to satisfy CALL "0"
BYTE 3+4	Address of routine to satisfy CALL "1"
BYTE 5+6	Address of routine to satisfy CALL "2"

If byte 0 contains n, the user need not include all numbers in the range 0 to n, in which case an unused integer has address 0. Thus if the user wishes to support CALL "0" and CALL "2" only, the table would be as follows:-

```
ORG       NNNNH           ;PROG ADDRESS FROM CONFIGURATOR
DB        2D              ;3 ROUTINES AVAILABLE
DW        ADDR0           ;ADDRESS OF CALL "0" ROUTINE
DW        0               ;CALL "1" NOT IMPLEMENTED
DW        ADDR2           ;ADDRESS OF CALL "2" ROUTINE
```

PARAMETER PASSING TO RUN-TIME SUBROUTINES

Parameter passing in run time subroutines is as follows:

1. If one parameter is passed, its address will be found in register pair B,C.
2. If two parameters are passed, the first parameter address will be passed in B,C the second address in D,E.
3. If three or more are passed, the last two will be passed as in 2 above, and the rest will be stacked, in such a way that the first parameter will be the last to be POPped from the stack.
4. The return address to the Run Time System will be found at the top of the stack on entry to the CALL code.
5. The user need not clear all parameters from the stack, since this will be automatically reset by the Run Time System, provided the address on the top of the stack on entry is returned to.
6. If register B,C and/or D,E are not used for parameter passing, they will contain 'FFFF' on entry to the CALL code.
7. After the last parameter has been POPped from the stack, the next POP will return the value FFFF.
8. If only one parameter is passed the entry following the return address on the stack will be FFFF as will registers D,E.
9. If no parameters are passed, then conditions will be as in 8 above with B,C set to FFFF also.

The use of terminator FFFF allows the user programmer to pass a variable number of parameters to the subroutine.

PLACEMENT OF THE SUBROUTINES IN THE SUBROUTINE AREA

The subroutines will typically be written completely independently of the COBOL program in any language which generates microprocessor order code. They will be assembled or compiled into absolute modules located at the addresses specified in the table. at the front of the subroutine area. During development these addresses will typically change with each new compilation, as the sizes of the various subroutines change.

The subroutine object code will then be patched into the subroutine area using the CP/M DDT utility.

This utility is described in detail in the CP/M Manual describing DDT.

SAMPLE RUN WITH RUN-TIME SUBROUTINES

The following series of operations show a typical CIS COBOL object program run where a CALL is made to user subroutines.

1. Place CP/M system disc in drive A.
2. Place object pack containing your HEX file in drive B.
3. Key B: to log in drive B.
4. Key A:DDT RUNA.COM - where RUNA.COM is the configured RTS.
5. The system will respond with:-

```
NEXT      PC
6100      0100H
```

6. Key Ixxxxxx.HEX - the HEX file identity
7. Key R
8. The system will respond with:-

```
NEXT      PC
XXYY      0000H
```

   At this point take a note of the first two digits of NEXT i.e. "XX" in this example - convert them to decimal from hexadecimal and subtract 1.

   EXAMPLE:

```
NEXT      PC
6216      0000H
```

   XX= 62H i.e., 98D-1 = 97D

   Make a note of this decimal value.

9. Press the Control and C keys simultaneously.
10. System responds with B
11. Key SAVE NN RUNZ.COM

    Where NN is the decimal number noted in (8), and RUNZ .COM is the of your new Run Time System.

ASSEMBLER SUBROUTINES PROVIDED BY MICRO FOCUS

The following standard CALL codes are available in the Run Time System.

CHAIN	CALL code "260"
PEEK	CALL code "261"
POKE	CALL code "262"
GET	CALL code "263"
PUT	CALL code "264"
ABSCAL	CALL code "265"

The user may call these routines without making any alteration to the Run Time System.

The CHAIN Subroutine

The CHAIN call allows another linked CIS COBOL program or any program not requiring parameters to be loaded and entered. There is no return to the calling program.

A parameter list of one variable must be passed with CALL CHAIN:

- The data-name containing the file-name of the program to chain to.
- The file-name must be terminated by at least one space character.

EXAMPLE:

```
WORKING-STORAGE SECTION.
                    .
                    .
                    .
                    .
    03    NEXT-PROG PIC X(10) VALUE "PRIN2.COM".
                    .
                    .
                    .
                    .
    03    CHAIN     PIC X(3) VALUE "260".
                    .
                    .
                    .
                    .
PROCEDURE DIVISION.
                    .
                    .
                    .
                    .
          CALL CHAIN USING NEXT-PROG.
                    .
                    .
                    .
```

The PEEK Subroutine

The PEEK call allows an absolute address location to be examined from a user program. The CALL returns into the user area a copy of the 8 bit value at the absolute address.

A parameter list of two variables must be passed with CALL PEEK:

- The five-character data-name containing the absolute address to be read from.
- The one-character data-name where the value is to be read to.

EXAMPLE:

```
WORKING-STORAGE SECTION.
                    .
                    .
                    .
                    .
    03    PEEK      PIC X(3)  VALUE "261".
                    .
```

```
                    .
                    .
                    .
    03    ADDRESS   PIC 9(5)  VALUE  1234.
                    .
                    .
                    .
                    .
    03    DATA-VAL  PIC X.
                    .
                    .
                    .
                    .
PROCEDURE DIVISION.
                    .
                    .
                    .
                    .
         CALL PEEK USING ADDRESS, DATA-VAL.
                    .
                    .
                    .
                    .
```

The POKE Subroutine

The POKE CALL allows an absolute address location to be set from a user program. The CALL transfers a copy of an 8-bit value in the user program to an absolute address.

A parameter list of two variables must be passed with CALL POKE:

- The five-character data-name containing the absolute address to be written to.
- The one-character data-name whose value is to be written.

EXAMPLE:

```
WORKING-STORAGE SECTION.
               .
               .
               .
               .
     03   POKE      PIC X(3)  VALUE "262".
               .
               .
               .
               .
     03   ADDRESS   PIC 9(5)  VALUE  2345.
               .
               .
               .
               .
     03   DATA-VAL  PIC X  VALUE "V".
               .
               .
               .
               .
PROCEDURE DIVISION.
```

```
                .
                .
                .
                .
     CALL POKE USING ADDRESS, DATA-VAL.
```

The GET Subroutine

The GET call allows a hardware port to be input from a user program. The CALL inputs the port and returns the 8 bit value to a user area.

A parameter list of two variables must be passed with CALL GET:

- The three-character data-name containing the port to be input from.
- The one-character data-name to be input to.

EXAMPLE:

```
WORKING-STORAGE SECTION.
                .
                .
                .
                .
     03   GET       PIC X(3)  VALUE "263".
                .
                .
                .
                .
     03   PORT      PIC 9(3)  VALUE 129.
                .
                .
                .
                .
     03   DATA-VAL  PIC X.
                .
                .
                .
                .
PROCEDURE DIVISION.
                .
                .
                .
                .
    CALL GET USING PORT, DATA-VAL.
                .
                .
                .
                .
```

The PUT Subroutine

The PUT call allows a hardware port to be output from a user program. The CALL outputs an 8 bit value to the port from a user area.

A parameter list of two variables must be passed with CALL PUT:

- The three-character data-name containing the port to be written to.

- The one-character data-name to be written.

EXAMPLE:

```
WORKING-STORAGE SECTION.
                .
                .
                .
                .
     03   PUT        PIC X(3)  VALUE "264".
                .
                .
                .
                .
     03   PORT       PIC 9(3)  VALUE 131.
                .
                .
                .
                .
     03   DATA-VAL   PIC X     VALUE X"2F".
                .
                .
                .
                .
PROCEDURE DIVISION.
                .
                .
                .
                .
    CALL PUT USING PORT, DATA-VAL.
                .
                .
                .
                .
```

The ABSCAL Subroutine

The ABSCAL call allows a subroutine CALL to an absolute location. No parameters are passed to the subroutine at the absolute address.

A parameter list of one variable must be passed with CALL ABSCAL:

- The five-character data-name containing the decimal absolute address to be called.

EXAMPLE:

```
WORKING-STORAGE SECTION.
                          .
                          .
                          .
     03   ABSCAL          PIC X(3)  VALUE "265".
                          .
                          .
                          .
     03   ADDRESS         PIC 9(5)  VALUE 5.
                          .
                          .
```

```
                              .
PROCEDURE DIVISION.
                              .
                              .
                              .
    CALL ABSCAL USING ADDRESS.
                              .
                              .
                              .
```

The File Name Manipulation Routines SPLIT and JOIN

CP/M names can be decomposed into a device code, file name and file extension, and the supplied sub-programs SPLIT and JOIN can be used by system programmers to decompose and reconstitute names in this way. Usually SPLIT is called first and then JOIN is used to produce a file name string with a modified extension.

Important use of these subroutines is made by the CIS COBOL software as follows:

- The compiler to produce default listing and intermediate code filenames from the source file name
- The compiler to produce the file names of its overlays
- Segmented programs to produce the file names for the various segments and the inter-segment reference file
- The standard CIS COBOL indexed sequential file package t o produce the name of the index file

SPLIT and JOIN can also prove of use to an application programmer where there is a requirement to process filenames partially specified, and when writing portable software.

A parameter list of four variables must be passed with CALL SPLIT or CALL JOIN:

1. Identifier of the complete name string (minimum length 20 bytes)
2. Identifier of the device substring (minimum length 6 bytes)
3. Identifier of the file name substring (minimum length 10 bytes)
4. Identifier of the file extension substring (minimum length 5 bytes)

SPLIT separates the string found at 1 storing its resultant substrings at 2. 3. and 4. separately; JOIN takes the substrings found at 2 . 3. and 4. and combines them storing the resulting complete string at 1.

The file name strings are subject to the CP/M maximum length and may be terminated earlier by a space character. This means that the parameters (1 - 4) specified above must be the identifiers of areas of WORKING STORAGE each at least as large as their respective minimum length.

The order of the parameters passed to SPLIT and JOIN is of course important:

CALL SPLIT using	filename-to-be-split, device-substring, name-substring, extension-substring.
CALL JOIN using	concatenated-substrings, device-substring, name-substring, extension-substring.

EXAMPLE:

```
 WORKING-STORAGE SECTION.
      01   Keyed-filename PIC X(20).
      01   Namstr PIC X(lO).
      01   Devstr PIC X(6).
      01   Extstr PIC X(S).

      01   DISK-filename PIC X(20) value spaces.

      01   Default-device PIC X(2) value "B:".

      01   SPLIT PIC X(3) value "268".

      01   JOIN PIC X(3) value "269".

      .
      .
      .
 Procedure Division.

           ACCEPT keyed-filename.

           Call SPLIT using keyed-filename, devstr, namstr, extstr.

*
* Now put default device into device string if user
* did not specify a particular device.

*

           IF devstr - spaces move default-device to devstr.

*

           call JOIN using disk-filename, devstr, namstr, extstr.

*
* Now perform file processing on filename specified by
* the user, and now concatenated in 'disk-filename'
*
            .
            .
            .
```

Chapter 5. CONFIGURATION UTILITY

OBJECTIVES

The Configuration Utility Program (CONFIG) can be used as follows:

1. To reserve an area within the RTS into which the user may enter assembler or other language subroutines for use by the CALL statement in a CIS COBOL program. This function may only be performed once and it is therefore essential to copy the RTS before running CONFIG. (The subroutine code is written by the user as an absolute segment which he then patches into the area reserved in the RTS using the CP/M DDT Utility).

2. To modify the default tabbing positions used when ACCEPTing data from the screen.

Note

CONFIG does not provide a capability for the inclusion of user subroutines into linked programs or programs that already contain user subroutines.

USING CONFIG

A CP/M System disc is loaded in drive A, and the CIS COBOL Issue Disk in drive B. CP/M is bootstrap loaded and the system responds as follows:

A>B:

B>

To load CONFIG the following entry is typed:

B>CONFIG [filename]

At this point, CONFIG signs on, as shown in the listings in the Appendices. It should be noted here that whenever CONFIG is waiting for the operator to key something, it will output the" >" sign as a prompt character. The first request from CONFIG is the file name of the run-time system to be configured if this has not been entered in the command line. In the appendices the reply RUNA.COM was made.

Once the configuration utility has been given the RTS file name, there will be a short pause during which it attempts to access the file. Should it fail to find the file (e.g . wrong file name or no .COM extension) it will display:

FILE OPEN FAILURE, PLEASE ENTER A NEW NAME

and request the file name to be entered again.

NOTES:

1. If the disk identifier is omitted, the configuration utility accesses the logged in disk which is in drive B.

2. A version check is carried out after successful opening of the RTS file. ONLY Version 4.5 programs can be configured using CONFIG Version 3.

The RTS allows the use of a 'TAB' character. This allows the user to jump eight characters at a time on input, as the default.

Users have the opportunity to vary this default, by replying Y (Yes). The configuration utility then asks the operator to key in the character positions at which the tabs should be placed (See Appendix H).

The RTS also provides the ability to supply Assembler code that will service the COBOL "CALL" verb. A reply of N at this point results in the end of run. The effects of replying Y are described under RUNTIME SUBROUTINES in this Chapter. See also RUN-TIME SUBROUTINES - CALL in Chapter 4.

Note

CONFIG does not allow for inclusion of user subroutines in a linked program.

At this point the RTS is ready to be stored on disk and there will be a short delay while this takes place.

RUN TIME SUBROUTINES

The user may include his own subroutines in the RTS, which can be CALLed from a CIS COBOL program. These may be written in assembler or other languages such as PL/M which generate 8080 or Z80 machine code. If such subroutines are required, then the configuration utility must be used to determine at what address they should be held.

The standard RTS supplied allows parameters to be used at run time to control the position at which the COBOL Intermediate Code is to be loaded. Parameters must not be entered if the ANIMATOR package is in use (+A was entered). Once the configuration utility has configured the Run Time System to allow run time subroutines to be included, this facility is withdrawn, and the Intermediate Code will always load at the address determined by the configuration utility . The actual address is dependent on the answers to questions posed by CONFIG requesting details of the facilities wanted in the RTS being configured.

The configuration utility will allow the following options:

1. To add the subroutines to the end of the RTS allowing all facilities to be used.

2. To remove the possibility of using the Interactive Debug package, overwrite this with the subroutines and load the intermediate code beyond this.

3. To overwrite the Indexed Sequential package and the Debug and ANIMATOR package.

4. To overwrite the Indexed Sequential, Debug and ANIMATOR packages.

Having ascertained where the run time subroutines should be located the user is asked to specify the length of the subroutines in order that the load point for the intermediate code may be determined. It is important to ensure that the figures input for the length of the subroutines is the maximum that is likely to be used, as any excess will be overwritten by the intermediate code.

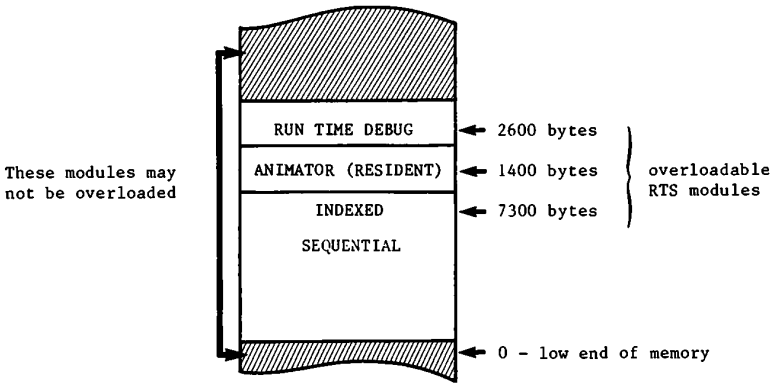
The configuration utility will advise the address at which the subroutines are to be located.

MEMORY MANAGEMENT CONSIDERATIONS

If the modules established by CONFIG as overloadable (based on user replies during the CONFIG run) have a total contiguous length exceeding that of the assembler routines, the routines can reside in this free space; otherwise they must be appended at the high-address end of the RTS.

It can therefore be seen that the total length of the RTS, once assembler subroutines are included, may or may not have increased depending on the two factors above.

The diagram below gives an idea of the length (in decimal) of the RTS overloadable modules in CIS V4.5.



From the above diagram it can be seen that the maximum length of assembler subroutines that can be embedded in the RTS is of the order of 11,000 bytes - only possible in the case where all of the three modules DEBUG, ANIMATOR, INDEXED are specified as excludable.

Note that the size of the RTS will NEVER decrease as a result of assembler subroutine inclusion, because of the fixed module at the top of the RTS.

Chapter 6. INCORPORATING FORMS-2 UTILITY PROGRAM OUTPUT

INTRODUCTION

The FORMS-2 Utility program offers two major facilities to CIS COBOL users:

1. The user can define screen layouts to be used in a CIS COBOL application by simply keying the text at the keyboard, and so producing a model form on the CRT.

2. The user can automatically generate programs to manipulate data input using the created form. In particular, indexed sequential files can be generated and maintained automatically, and these files can, of course, be used with CIS COBOL programs.

The FORMS-2 Utility is available as a separate software package, and is supported by the FORMS-2 Utility Program Users Guide.

SCREEN LAYOUT FACILITY

The FORMS-2 Screen Layout facility generates source COBOL Record Descriptions for screen layouts.

MAJOR FACILITIES

Users have three major facilities available to them:

1. They may store an image copy on disk of the form they have just defined for subsequent use in this or another FORMS-2 run. The image can be printed to obtain a hard copy, using the O/S standard file print utility program.

2. They may generate CIS COBOL source code for the data descriptions required to define the form just created. This may then be included into a CIS COBOL program by use of the COPY verb.

3. They may choose to generate a Check Out program which allows duplication of many machine conversations which would take place during a run of the application which is being designed.

CIS COBOL PROGRAMMING FOR FORMS-2 SCREEN LAYOUTS

All that the user has to do to incorporate FORMS-2 Screen layout output in a program is to specify the FORMS-2 output file name (filename.DDS) in a COBOL COPY statement. Obviously data item names in the user program must be specified to correspond with those generated from a user-specified base name by FORMS-2. Details of FORMS-2 name generation are given in the FORMS-2 Utility Program Users Guide.

EXAMPLE:

```
000000     COPY          "DEMO.DDS".
```

GENERATED PROGRAMS

The FORMS-2 Utility generates a COBOL program which maintains data stored in the created forms in an indexed sequential file automatically, with automatic generation of file names from a user-supplied base name. These files comply with the standards in use by the operating system under which CIS COBOL is being used.

CIS COBOL PROGRAMMING FOR FORMS-2 GENERATED FILES

No special programming is required to use FORMS-2 generated program files in a CIS COBOL application program. The files are processed as normal indexed sequential files. It is worth noting that the files can be fully maintained interactively by use of only the FORMS-2 Utility. In addition to establishing or deleting files, this includes the following facilities:

- Insertion of new records
- Insertion of the same data in records with different keys
- Display of any selected record/s (Full inquiry facility)
- Insertion or amendment of records dependent on their key
- Deletion of records
- Read and display next record or a message if end of file detected
- Terminate run

Details of the FORMS-2 Indexed Sequential File handling facilities are given in the FORMS-2 Utility Program Users Guide.

Chapter 7. USING THE ANIMATOR UTILITY PROGRAM

ANIMATOR is a COBOL oriented debugging tool that is available for use with CIS COBOL. The main aim of ANIMATOR is to free the COBOL programmer from the need to be aware of the internal representations of either data or procedural code, so that even a trainee programmer already has the knowledge necessary to debug his programs effectively.

This is achieved by using the screen as a "window" into the source COBOL program and "animating" execution by moving the cursor from statement to statement as execution proceeds. Speed of execution can be varied; the user may also switch off animation thus allowing rapid execution up to the area of interest.

The user can interrupt execution at any point, either by defining break-points or dynamically simply by pressing the space-bar on the keyboard. Whilst execution is suspended the user can easily examine any part of the source code by means of simple commands to refresh the screen display. This means that it is not even necessary to have a printed compilation listing in order to debug a program.

Various other debugging functions are available, invoked by pressing a key. Only the top 20 lines of the screen are used for the display of source code, the bottom area being used to display menus of available commands, some of which invoke subordinate command menus.

Where debugging functions require reference to either data items or procedural statements this is achieved by the user moving the cursor to “point” at the appropriate place in the source code. Alternatively data items can be referenced by actually typing the COBOL data-name.

Where control of ANIMATOR requires more keyboard input than simply pointing with the cursor or pressing one of the displayed command characters, COBOL syntax is used. For instance, replacement of data item values is achieved by typing that value in COBOL literal format (i.e. non-numeric literals are enclosed in quotes).

The facilities provided in ANIMATOR make it much more than simply a COBOL-oriented debugger. It can be a valuable training aid, and also provides the ideal means for a programmer to attain understanding of an unfamiliar program.

ANIMATOR is supplied as a separate product complete with dccumentation. This Chapter describes CIS COBOL operating considerations in order to use the ANIMATOR utility.

COMPILATION

In order to be able to use ANIMATOR with a CIS COBOL program, a specific directive must be included in the CIS COBOL compiler command line.

THE ANIM COMPILER DIRECTIVE

The inclusion of directives in the compiler command line is described in Chapter 2 of this manual. If the ANIM directive is included the compiler will compile the source input in such a way as to allow run time animation. The compiler generates in addition to the ".INT" file, three other files with extension identifiers as follows:

.DOC
.SCP
.SCB

These files will be directed to the same drive as the intermediate file produced by the compiler.

Note

The intermediate code file includes data specifying whether or not it was produced by compilation with the ANIM directive specified. An intermediate file produced by compiling without ANIM cannot be run with animation even if the three extra files mentioned above are available from a previous compilation when ANIM was specified.

RUNNING PROGRAMS WITH ANIMATOR

To run a CIS COBOL program that has been compiled with the ANIM compiler directive specified, it is necessary to enter the run command line parameter +A. Chapter 3 of this manual describes the CIS COBOL Run Command line.

THE +A RUN COMMAND PARAMETER

In addition to specifying a particular load point for a user program (see Chapter 3) the +A parameter is the animation run time switch, and causes ANIMATOR to be loaded and run providing dynamic control of the user program. The following files must be present at run time in order to use ANIMATOR:

File	Disk Drive
$ANIM.V45	The logged in drive
filename.CSL filename.SCP filename.SCB filename.DDC	The drive containing the int. code (Note: the file containing the COBOL source must have the extension .CBL)

If $ANIM.V45 is not present on the logged-in drive, a message is displayed on the VDU and ANIMATOR is permanently switched off. If any of the other files is not present, then the message

Animation of root programs inhibited - missing files

is displayed and ANIMATOR is not activated for the root program, but still may be invoked for called subprograms.

Note

Deletion/Renaming of files (except $ANIM.V45) can be used to switch off animation for selected programs within a suite. This facility can be used as an alternative to recompiling without the ANIM switch.

EXAMPLES:

The directive

```
RUN +A PROG.INT<<
```

loads and runs the program PROG with animation. The program must have been compiled with ANIM and all necessary files (see LOAD Parameter in Chapter 3) must be present. Also, the RTS must be capable of initiating ANIMATOR (i.e. this facility is available and has not been omitted at configuration time - see Chapter 5).

The directive

```
RUN +A = PROG.INT<<
```

is invalid, and results in the message

"=" and "+A" not allowed in conjunction

being displayed on the screen, followed immediately by a return to CP/M.

The directive

```
RUN +A +I = PROG.INT<<
```

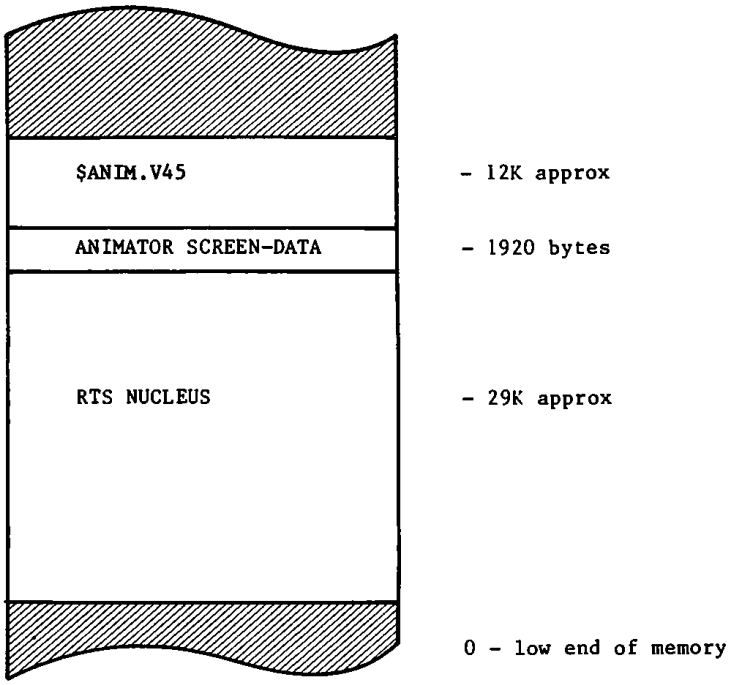
is invalid (only 1 load parameter allowed) and results in the message

Command line processing error

being displayed on the screen, followed immediately by a return to CP/M.

MEMORY MANAGEMENT CONSIDERATIONS

The size of the RTS with ANIMATOR included is larger by 1920 (decimal) bytes, than it will be when not included. Additionally, the program $ANIM.V45 will be loaded as and when it is necessary to animate a program, and will remain in memory thereafter. The diagram that follows gives an idea of the memory usage by CIS systems components when running with ANIMATOR:



ANIMATOR attempts to load the complete Data Division of the program to be animated into memory; it then loads as much of the Procedure Division as can be fitted in (ANIMATOR maintains a 'window' onto the procedure division code). If the entire Data Division of the program cannot be accommodated in available memory, then the program cannot be animated.

Note

1. In addition to memory usage by CIS COBOL system components, memory may be reserved by a resident operating system at the top end of memory.

2. ANIMATOR and Interactive Debug (see Chapter 3) are mutually exclusive facilities and cannot be used concurrently .

3. If the RTS has been configured for user subroutines, and at the time of configuration the ANIMATOR or Interactive Debug modules were excluded (as described under MEMORY MANAGEMENT CONSIDERATIONS in Chapter 3) it is invalid to supply a load parameter of "+A" or "+D", since the RTS no longer contains these modules. In general, attempts to activate a facility which has been omitted in this way will result in the message:

   Pre-assigned Load Point Used

Appendix A. SUMMARY OF COMPILER AND RUN-TIME DIRECTIVES

COMPILER DIRECTIVES

The general format of the command line for compilation is:

```
A> COBOL filename [directives]
```

filename is the name of the file that contains the CIS COBOL source program.

A description of the available compiler directives follows:

FLAG (level)

This directive specifies the output of validation flags at compile time. The parameter "level" is specified to indicate flagging as follows:

LOW Produces validation flags for all features higher than the Low Level of compiler certification of the General Services Administration (GSA).

L-I Produces validation flags for all features higher than the Low-Intermediate level of compiler certification of the GSA.

H-I Produces validation flags for all features higher than the High-Intermediate level of compiler certification of the GSA.

HIGH Produces validation flags for all features higher than the High Level of compiler certification of the GSA.

CIS Produces validation flags for only the CIS COBOL extensions to standard COBOL as it is specified in the ANSI COBOL Standard X.23 1974. (See the *CIS COBOL Language Reference Manual*)[1].

NOFLAG

No flags are listed by the compiler. This is the default if the FLAG directive is ommitted.

RESEQ

If specified, the compiler generates COBOL sequence numbers, re-numbering each line in increments of 10. The default is that sequence numbers are ignored and used for documentation purposes only, i.e., NORESEQ.

NOINT

No intermediate code file is output. The compiler is in effect used for syntax checking only. The default is that intermediate code is output, i.e., INT (sourcefile.INT).

NOLIST

No list file is produced; used for fast compilation of "clean" programs. The default is a full list, i.e., LIST (sourcefile.LST).

[1] Up to version 4.4, the *FLAG (level)* directive was called the *ANS switch*. On older versions of the compiler, use `ANS` as substitute for `FLAG CIS`.

COPYLIST

The contents of the file(s) nominated in COPY statements are listed. The list file page headings will contain the name of any COPY file open at the time a page heading is output. The default is NOCOPYLIST.

NOFORM

No form feed or page headings are to be output by the compiler in the list file. The default is headings are output, i.e., FORM(60).

ERRLIST

The listing is limited to those COBOL lines containing any syntax errors or flags together with the associated error message(s). The default is NOERRLIST.

INT (external-file-name)

Specifies the file to which the intermediate code is to be directed. The default is: source-file.INT.

LIST (external-file-name)

Specifies the file to which the listing is to be directed (this may be a printing device, ie. console or printer or a disk file) The default is: source-file.LST.

For list to console use: LIST(CON:) or LIST (:CO:)

For list to line printer use: LIST(LST:) or LIST (:LP:)

FORM (integer)

Specifies the number of COBOL lines per page of listing (minimum 5). The default is 60.

NOECHO

Error lines are echoed on the console unless this directive is specified. The default is ECHO.

NOREF

Suppresses output of the 4-digit location addresses on the right hand side of the listing file. REF is the default.

DATE (string)

The comment-entry in the DATE-COMPILED paragraph, if present in the program undergoing compilation, is replaced in its entirety by the character string as entered between parentheses in the DATE compiler directive. This date is then printed at the top of every listing page except the first.

QUIET

The full text of error messages is suppressed, only the numbers are produced. The default is NOQUIET.

PAGETHROW (character-code)

Specifies the ASCII character code for physical printer page throw. Default is PAGETHROW (12).

ANIM

The program is compiled for run-tim e debugging with the optional ANIMATOR product, (See Chapter 7). Default is NOANIM.

FILESHARE

The program to be compiled contains additional FILESHARE syntax that can be read only if you have the optional FILESHARE product.

RESTRICT (organization), COMMIT (organization), DERESTRICT (organization)

Specifies the shared access mode for all files with the organization entered. Can only be used with the optional FILESHARE product. See FILESHARE directive above and also Chapter 8.

RUN TIME DIRECTIVES

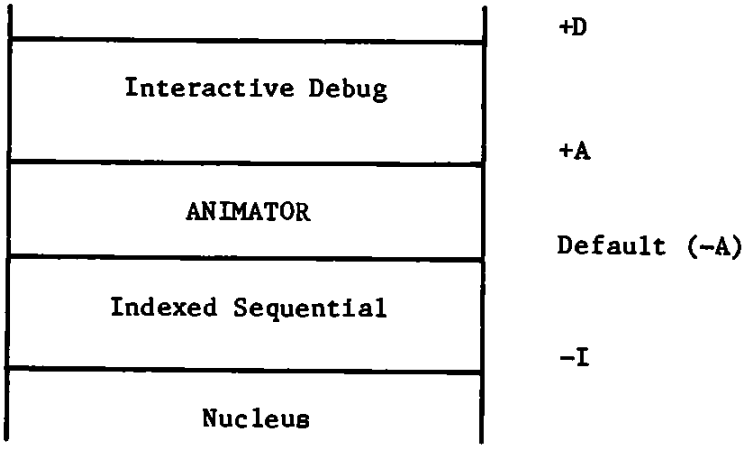
The command line syntax for running a CIS COBOL object program is as follows:

```
RUN [-V] [load param] [switch param] [link param] filename
[program params]
```

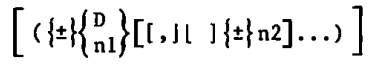
where:

-V inhibits the compatibility check between the compiler and RTS versions.

load param loads modules as follows:



switch param is of general format:



n1 and n2 are any program switch numbers (See Language Reference Manual) in the range 0-7

D invokes the standard ANSI COBOL Debug module

+ or - sets the associated switch on or off

link param is the = (equal sign) symbol which is us ed to link the program with the Run Time System so that it can be directly loaded. Note that it is important to rename the SAVE file generated to avoid it being overwritten at the next use of the = parameter. (Cannot be used with +A).

filename is the name of the file in which the intermediate code of the program to be loaded is stored

program params are any formats required to be passed to the program from the Operator at load time. These are user specific.

Appendix B. COMPILE-TIME ERRORS

The error descriptions that correspond to error numbers as printed on listings produced by the CIS COBOL compiler are as follows:

ERROR	DESCRIPTION
01	Compiler Error; consult your Technical Support Service
02	Illegal format of data-name
03	Illegal format of literal or invalid use of 'ALL'
04	Illegal format of character
05	Data-name declared twice
06	Too many data or procedure names have been declared - compilation abandoned
07	Illegal character in column 7, or continuation line error
08	Nested COPY statement or unknown file specified
09	'.' missing
10	The statement starts in the wrong area of the source line
22	'DIVISION' missing
23	'SECTION' missing
24	'IDENTIFICATION' missing
25	'PROGRAM-ID' missing
26	'AUTHOR' missing
27	'INSTALLATION' missing
28	'DATE-WRITTEN' missing
29	'SECURITY' missing
30	'ENVIRONMENT' missing
31	'CONFIGURATION' missing
32	'SOURCE-COMPUTER' missing
33	OBJECT-COMPUTER or SPECIAL-NAMES clause in error
34	'OBJECT-COMPUTER' missing
36	'SPECIAL-NAMES' missing
37	SWITCH Clause in error
38	DECIMAL-POINT Clause in error
39	CONSOLE Clause in error
40	Illegal currency symbol
42	'DIVISION' missing
43	'SECTION' missing
44	'INPUT-OUTPUT' missing
45	'FILE-CONTROL' missing
46	'ASSIGN' missing
47	'SEQUENTIAL' or 'RELATIVE' or 'INDEXED' missing
48	'ACCESS' missing on indexed or relative file
49	'SEQUENTIAL' or 'DYNAMIC' missing
50	Illegal combination ORGANIZATION/ACCESS/KEY

ERROR	DESCRIPTION
51	Unrecognised clause in SELECT statement
52	RERUN clause contains syntax error
53	SAME AREA clause contains syntax error
54	File-name missing or illegal
55	'DATA DIVISION' missing
56	'PROCEDURE DIVISION' missing or unknown statement
57 *	'EXCLUSIVE', 'AUTOMATIC' or 'MANUAL' missing
58 *	Non-exclusive lock mode specified for restricted file
62	'DIVISION' missing
63	'SECTION' missing
64	File-name not specified in SELECT statement
65	RECORD SIZE integer missing
66	Illegal level number or level 01 required
67	FD qualification contains syntax error
68	'WORKING-STORAGE' missing
69	'PROCEDURE DIVISION' missing or unknown statement
70	Unrecognized clause in Data Description or previous'.' missing
71	Incompatible clauses in Data Description
72	BLANK is illegal with non-numeric data-item
73	PICTURE clause too long
74	VALUE with non-elementary item, wrong data-type or value truncated
75	VALUE clause in error or illegal for PICTURE type
76	FILLER/SYNCHRONIZED/JUSTIFIED/BLANK clause for non-elementary item
77	Preceding item at this level has 0 or more than 8192 bytes
78	REDEFINES of different levels or unequal field lengths.
79	Data Division exceeds 32K and data-item has address above 7FFF
81	Data Description clause inappropriate or repeated
82	REDEFINES data-name not declared
83	USAGE must be COMP, DISPLAY or INDEX
84	SIGN must be LEADING or TRAILING
85	SYNCHRONIZED must be LEFT or RIGHT
86	JUSTIFIED must be RIGHT
87	BLANK must be ZERO
88	OCCURS must be numeric, non-zero and unsigned
89	VALUE must be a literal, numeric literal or figurative constant
90	PICTURE string has illegal precedence or illegal character
91	INDEXED data-name missing or already declared
92	Numeric edited PICTURE string is too large
101	Unrecognised verb
102	IF ELSE mismatch
103	Data-item has wrong data-type or is not declared

ERROR	DESCRIPTION
104	Procedure name has been declared twice
10S	Procedure name is the same as a data-name
106	Name required
107	Wrong combination of data-types
108	Conditional statement not allowed; imperative statement expected
109	Malformed subscript
110	ACCEPT or DISPLAY wrong
111	Illegal syntax used with I-O verb
112 *	LOCK clause specified for file with lock mode EXCLUSIVE
113 *	KEPT specified for uncommittable file
115 *	KEPT omitted for comittable file
116	IF statements nested too deep (maximum 8)
117	Structure of Procedure Division wrong (e.g. DECLARATIVES not first)
118	Reserved Word missing or incorrectly used
119	Too many subscripts in one statement
120	Too many operands in one statement
141	Inter-segment procedure name declared twice
142	IF ELSE mismatch at the end of source input
143	Data-Item has wrong data-type or is not declared
144	Procedure name undeclared
145	INDEX name declared twice
146	Cursor address field not declared or not 4 bytes long
147	KEY declaration missing or FD missing
148	STATUS declaration missing
149	FILE STATUS data-item has the wrong format
150	Paragraph to be ALTERed is not declared
151	PROCEDURE DIVISION in error
152	USING parameter is not declared in the linkage section
153	USING parameter is not level 01 or 77
154	USING parameter is used twice in the parameter list
157	Structure of Procedure Division wrong (e.g. DECLARATIVES not first)
160	Too many operands in one statement

* The error codes marked by an asterisk apply only when the optional FILESHARE product is in use.

In addition to these numbered error messages, the following message can be displayed with subsequent termination of the compilation:

```
FATAL I-O ERROR:  filename
```

where filename is the erroneous file.

Any intermediate code file produced is not usable.

The following conditions will cause this error:

Disk overflow
File directory overflow
File full
Impossible I-O device usage

Other operating system dependent conditions can also cause this error.

Note

You will notice that the numbers of the numbered error messages listed above are not continuous, i.e., there are gaps in the numbering. The compiler should never have cause to generate an error message with a number not listed above. If you ever encounter such a number, consult your Micro Focus Product Technical Support office.

Appendix C. RUN-TIME ERRORS

Run-time error messages are preceded by the name and segment number of the currently executing intermediate code file.

There are two types of runtime errors: Recoverable and Fatal.

(a) Recoverable errors

If the programmer has specified the STATUS clause in the FILE-CONTROL paragraph of a program error handling is the programmer's responsibility. This will generally only apply to errors that are not considered fatal by the operating system.

(b) Fatal errors

All errors except those above are fatal. They may come from the operating system or from the run-time system. Fatal errors cause a message to be output to the console which includes a 3-digit error code and reference to the COBOL statement subsequent to that in which the error occurred. These fall into two classes:

(i) Exceptions — These cover arithmetic overflow, subscript out of range, too many levels of perform nesting.

(ii) I-O errors — These exclude those for which STATUS is not selected as above.

Error	**Description**
151	Random read on sequential file
152	REWRITE on file not open I-O
153	Subscript out of range
154	Perform nesting exceeds 22 levels
156	Invalid file operation
157	Object file too large
158	REWRITE on line-sequential file
159	Malformed line-sequential file
161	Illegal intermediate code
162	Arithmetic overflow or underflow
164	Specified CALL code not supplied or Attempt to call a COBOL module recursively (i.e when is already active)
165	Incompatible releases of compiler and run-time system
168	Memory arrangement failure
169	Invalid indirect sequential file key length (>32 characters)
170	Illegal operation in Indexed Sequential
171	Attempt to read I-S record in output/extend mode
172	Attempt to delete I-S record in non I-O mode
173	Attempt to write I-S record in input mode
174	Attempt to CALL/CANCEL on active program
176	Illegal inter-segment reference
180	COBOL file malformed
181	Fatal file malformation

Error	Description
194	File size too large (>0.5MB)
195	DELETE/REWRTTE not preceded by a READ
196	Relative (or Indexed) - Record number too large (>65535)
197	File save failure
198	Program load failure (using CHAIN)
199	Indexed sequential file name too long (>20 characters)
200	Insufficient space to load Animator

Appendix D. OPERATING SYSTEM ERRORS

These errors appear in the same format as CIS COBOL Run-Time errors; conventionally error numbers 1-99 are reserved for the operating system. In the following list fatal errors are marked with an asterisk.

ERROR	DESCRIPTION
0	No error
* 1	Insufficient buffer space
2	File not open when access attempted
3	Attempt to open more than 12 files simultaneously
4	Illegal file name
5	Illegal device specification
6	Attempt to write to input file
* 9	No room in diskette directory
12	Attempt to open file already open
13	Attempt to open for input a non-existant file
22	Illegal or impossible access mode to OPEN
* 24	Disk input-output error[a]

[a] Could be caused by physical surface damage, incorrect format or invalid address marker.

Appendix E. INTERACTIVE DEBUG COMMAND SUMMARY

	COMMAND	EFFECT	
	A data-ref val	Change value at address given to val (data division)	
	B	Execute until specified location changes	
	C val	Display ASCII character corresponding to val	
	D data-ref	Display 16 bytes from address given	
	E	Execute until specific location changes to specified contents	
	G proc-ref	Execute from current position until given address is reached	
	L	Output carriage return/line feed to console	
	M name	Start definition of macro	
	N	Set relative addressing default to start of user area	
	O	Set relative addressing default to start of segment	
	P	Display current program counter	
	S data-ref	Set work register to address given	
	T proc-ref	Trace all paragraphs executed up to address (Procedure Division)	
	X	Execute one instruction	
	$	End macro definition	
	/	Display byte at address in work register	
	. val	Change byte at address in work register to val and increment register	
	,	Increment work register	
	;	Start comment - line up to carriage return is ignored	
where:		data-ref	16 bit hex value (4 digits) in data area
		proc-ref	16 bit hex value (4 digits) in code area
		val	8 bit value (2 hex digits or inverted commas and ASCII char eg "A")
		name	single ASCII character

Appendix F. CP/M DISK FILES

GENERAL

The disk file system used in CIS COBOL is the diskette based CP /M system described in the INTRODUCTION TO CP/M FEATURES AND FACILITIES Manual. A description of file creation and management is available in that Introduction.

CIS COBOL offers sequential, relative and indexed organizations.

All file processing information is defined within an interactive CIS COBOL program. File organization, access method, device assignment and allocation of disk space are defined by the SELECT statement in the INPUT-DUPUT SECTION of the ENVIRONMENT DIVISION and an FD entry in the FILE SECTION of the DATA DIVISION.

SPECIFYING FILES

CIS COBOL offers fixed (compile time) file assignment and dynamic (run time) file assignment facilities.

FIXED FILE ASSIGNMENT

The CP/M file name is assigned to the internal user file-name at compile time as shown in the specifications that follow.

Environment Division

In the FILE-CONTROL paragraph the general format of the SELECT and ASSIGN TO statements is as follows:

General Format

SELECT file-name ASSIGN TO { external-file-name-literal | file-identifier }
[, external-file-name-literal | file-identifier]

Parameters

filename - Can be any user-defined CIS COBOL word (see User Defined COBOL Words in Chapter 2)

file-identifier - See Run-Time File Assignment later in this Appendix

external-file-name-literal - Is a standard CP/M file name of the following general format:

{ [drive] filename [extension] | device }

where:

drive - The pre-established CP/M disk drive identifier A: through P:

device - Devices other than disk as follows:[1]

LPT:	Line Printer	PUN:	Punch Device
LST:		:TP:	

[1] - The availability of any of these devices is dependent upon the availability of the driver software for the device in your version of CP/M.

:LP:		:HP:	High Speed Punch
:CI:	Keyboard Input	:RDR:	Reader Device
:CC:	Screen Output	:TR:	
CON:	Console I-O	:HR:	High Speed Reader
		:BB:	Byte Bucket

filename - One through eight alphabetic or numeric characters (no spaces)

extension - One through three alphabetic or numeric characters (no spaces)

Examples or Fixed File Assignment

```
SELECT     STOCKFILE
    ASSIGN TO "B:WAREHS.BUY".
SELECT    STOCKFILE
     ASSIGN TO ":F1: WAREHS.BUY".
```

Data Division

The file-name specified as above is then used in the File Description for that program (see File File Description - Complete Entry Skeleton in Chapters 5, 6 and 7 of the *CIS COBOL Language Reference Manual*).

Procedure Division

The file-name specified as above is then also used in the OPEN and CLOSE statements when the file is required for use in the program. (See THE OPEN STATEMENT and THE CLOSE STATEMENT in Chapters 5, 6 and 7 of the *CIS COBOL Language Reference Manual*),

RUN-TIME FILE ASSIGNMENT

The internal user file-name is assigned to a file-identifier (an alphanumeric user-defined COBOL Word), which automatically sets up a PIC X(15) data area in which to store the external CP/M file name. The external CP/M file name can then be stored in this data area in the Procedure Division by the user, and can be altered during the run as required.

The following specifications are required run-time assignment:

Environment Division

In the FILE-CONTROL paragraph the general format of the SELECT and ASSIGN TO statements is as follows:

General Format

SELECT filename ASSIGN TO fileidentifier

Parameters

file-name - Can be any user-defined CIS COBOL word. (See User defined COBOL Words in Chapter 2 of the *CIS COBOL Language Reference Manual*).

file-identifier - Is any user-defined CIS COBOL word (See User Defined COBOL Words in Chapter 2 of the *CIS COBOL Language Reference Manual*),

Example of Run-Time File Assignment

```
    SELECT STOCKFILE

         ASSIGN STOCKNAME.
```

Data Division

The file-name specified as above is then used in the File Description for that program (see THE FILE DESCIPTION - COMPLETE ENTRY SKELETON in Chapters 5, 6 and 7 of the *CIS COBOL Language Reference Manual*).

Procedure Division

The external CP/M file name of the required file (see under FIXED FILE ASSIGNMENT above for format) is then stored as required in the file-identifier location specified above by the user program before the file is OPENed for use.

EXAMPLE:

```
MOVE     "B:WAREHS.BUY" TO STOCK-NAME.
OPEN     INPUT STOCK-FILE.
.
.
.
CLOSE     STOCK-FILE.
.
.
.
MOVE     "B:WAREHS.SEL" TO STOCK-NAME.
OPEN     INPUT STOCK-FILE.
.
.
.
CLOSE     STOCK-FILE.
.
.
.
MOVE "B:PROGA.SRC" TO file-identifier.
OPEN      INPUT     file-name.
```

The CP/M file name could have been entered via an ACCEPT statement i.e. by an operator, or stored as any other variable data.

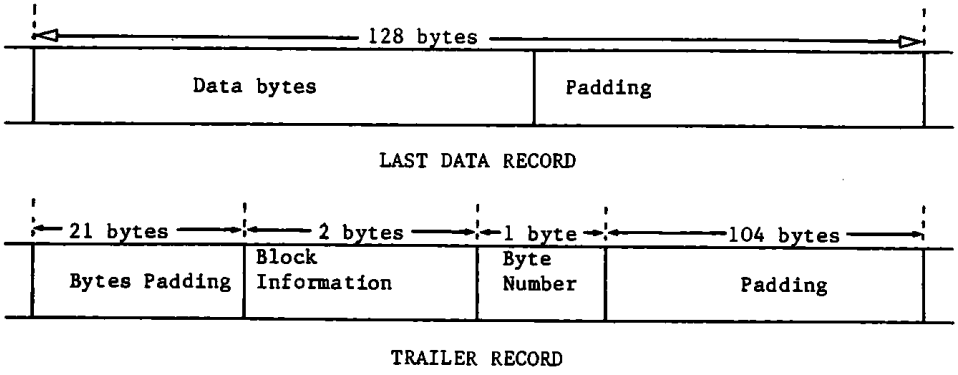
In this way different external files can be used as a common internal user file during any run of a program, but care is required to ensure that the correct file is allocated at any given time.

Note

The device assignment B: in the file name above can be replaced by the format :F1: for compatability with other operating systems.

BLOCK LENGTHS

CP/M uses fixed-length 'CP/M records' (blocks) on disk of 128 bytes per block. Since CIS COBOL permits block lengths other than 128 bytes, a trailer block is appended by CIS COBOL to CP/M files. The last two blocks in a file appear as follows:



The last data block is padded beyond the last data byte with EOF characters (1AH) up to 128 bytes. If the last data block is full i.e. 128 bytes long, then no padding is inserted. The trailer block contains the position at which the next data byte would be inserted in 'byte number' and 'block number Within file' format.

An important corollary of this is that if the CP/M utility PIP is used to move CIS COBOL files it must treat them as binary files. This means either renaming them to have the extension .COM, or using the "[0]" parameter (alpha 0).

Files read as line-sequential need not possess the trailer block and need only be terminated by using the standard CP/M EOF convention. This allows source programs to be prepared using the CP/M editor.

CIS COBOL DISK FILE STRUCTURES UNDER CP/M

CIS COBOL offers four types of file organization for use by the COBOL programmer - Sequential, line sequential, relative and indexed sequential (ISAM). A file is a set of records. A record is a set of contiguous data bytes which are mapped into hardware sectors with which they need not coincide, i.e. a record can start anywhere within a sector and can span hardware sector boundaries. The data is held as follows:

SEQUENTIAL

Sequential files are read and written using fixed length records, the length used being that of the longest record defined in the COBOL program's FD.

Normally the space occupied per record is the same as the program record length and data of any type may be held on the file: this does not however apply if WRITES are done using BEFORE or AFTER ADVANCING, as extra control characters are inserted and the data cannot then be read back correctly.

The RTS writes a trailer block to an output file to mark the precise position of the end of data, and expects to find one on an input file. There are no limits on file size beyond those imposed by the operating system and/or hardware.

LINE SEQUENTIAL

Line sequential file format is intended to cater for text (ASCII) files as generated by editors and other similar utilities. This is the only type of CIS COBOL file format in which variable length records are supported: the two-byte combination 0D0AH (carriage return, line feed) is used as a record delimiter, and any single byte 1AH (control-Z) as an unconditional file terminator. On input the CR-LF is removed and the record area padded out with spaces as necessary: on output any trailing spaces in the program's record area are ignored. Use of ADVANCING phrases other than BEFORE 1 causes the output of additional device control characters. A file created in this way can still be read by a program, but the additional control characters are not filtered out and will appear in the record area.

RELATIVE

Relative file organization provides a means of accessing data randomly by specifying its position in the file. Records are of fixed length, the length used being that of the longest record defined in the program's FD. To designate whether or not a record logically exists, two bytes are added to the end of each record: these contain 0D0AH if the record logically exists on the file and 0000H if it does not. The total length of a file is determined by the highest relative record number used; CIS COBOL imposes a limit of 65535 on this value independently of operating system and/or hardware constraints. Data of any type may be held on the file; the RTS uses a trailer block to determine the precise position of the end of data.

INDEXED SEQUENTIAL

An indexed sequential (ISAM) file occupies two CP/M files on disk: both are in a relative file format, one containing the data and the other all indexing and free space information - the index (.IDX) file.

The name for the index file is derived from the name supplied for the ISAM file by substituting the extension '.IDX' in place of any supplied in the ISAM file name. The name for the Data file is the same as that supplied for the ISAM file. This means that different ISAM files cannot be distinguished purely by a change in the file-name extension and also that it is advisable to refrain from using the extension '.IDX' in other contexts. e.g. 'CLOCK.FLE' as an ISAM file-name produces an index 'CLOCK.IDX' in addition to the CLOCK.FLE data file.

The index is built up as an inverted tree structure which grows in height as records are added: the number of index file accesses required to locate a randomly selected record depends principally on the number of records on the file and the 'keylengths'. An approximate guide to the number of levels in the tree (and hence the number of accesses required) is

index levels $\log_k$ (number of records)

where k = 150 / keylength + 2

but will vary slightly on the order in which records are added and deleted.

Faster response times are obtainable when reading a file sequentially, but only if other ISAM operations do not intervene.

The size (in bytes) of an ISAM file is approximately related to the maximum number of records it contains as follows:

data = (record length + 2) * max. no of records

index = no of records / k - 1 * 256 where k is as defined above

Note

The necessity of taking regular back-up copies of all types of files cannot be emphasised too strongly and this should always be regarded as the main safeguard. There are however situations with indexed files (e.g. media corruption) that can lead to only one of the two files becoming unuseable. If the index file is lost in this way, it is normally possible to recover data records from just the data file (although not in key sequence) and cut down on the time lost due to a failure. As an aid to this, all unused data records are marked as deleted at the relative file level by appending two bytes to each record which contain LOW-VALUES. For undeleted records these bytes contain the characters Carriage Return and Line Feed. The recovery operation may therefore be done with a simple COBOL program by defining the data file as ORGANIZATION SEQUENTIAL ACCESS SEQUENTIAL with records defined as two bytes longer than in the ISAM file description. The records are then read sequentially, the data MOVEd from the sequential file record. area into the indexed (ISAM) file record area, and written to a new version of the indexed file; except for those records with LOW-

VALUES in the last two (extra) bytes which records should be discarded. Note that these two bytes (containing carriage-return and line-feed characters in a required record) are not written to the ISAM file on recovery, by virtue of the record length discrepancy of 2 bytes in the record definitions.

FILE ERROR STATUS

If a programmer has specified the STATUS clause in the FILE-CONTROL paragraph in a program the operating system error number as returned by CP/M is available in the Status Key 2 byte in the event of a file error (See the *CIS COBOL Language Reference Manual*). If it is required to display this status with its correct decimal value, careful redefinition of data-items is required in order to avoid truncation of the value. This is because the facility that enables the storage of a nonnumeric value greater than decimal 99 as a hexadecimal value is an extension to the ANSI COBOL standard X3.23 (1974) but the rules for moving or manipulating such data are retricted by the standard to a maximum of decimal 99.

The example that follows illustrates one method of retrieving the value of status key 2 for display purposes.

Note how truncation has been avoided by redefining the two status bytes as one numeric data item (length two bytes) capable of storing up to four decimal digits.

```
** CIS COBOL V4.4                  B:STATUS.CBL                    PAGE: 0001
**
** OPTIONS SELECTED
**     RESEQ
**
000010 ENVIRONMENT DIVISION.                                          0118
000020 INPUT-OUTPUT SECTION.                                          0118
000030 FILE-CONTROL.                                                  0118
000040 SELECT FILE1 ASSIGN "TST.FIL"                                  0184
000050  STATUS IS FILE1-STAT.                                         0186
000060 DATA DIVISION.                                                 01BD
000070 FILE SECTION.                                                  01BD
000080 FD FILE1.                                                      01BD
000090 01  F1-REC PIC X(80).                                         01BD
000100 WORKING-STORAGE SECTION.                                       020F
000110 01  FILE1-STAT.                                               020F 00
000120  02 S1 PIC X.                                                 020F 00
000130  02 S2 PIC X.                                                 0210 01
000140 01  STAT-BIN REDEFINES FILE1-STAT PIC 9(4) COMP.              020F 00
000150 01  DISPLY-STAT.                                              0211 02
000160  02 S1-DISPL PIC X.                                          0211 02
000170  02 FILLER PIC X(3).                                         0212 03
000180  02 S2-DISPL PIC 9999.                                       0215 06
000190 PROCEDURE DIVISION.                                            0000
000200      OPEN INPUT FILE1.                                        001A
000210      IF S1 NOT = 9 GO TO PARA1.                               001E
000220                                                               0030
000230      MOVE S1 TO S1-DISPL.                                     0030
000240      MOVE LOW-VALUES TO S1.                                   0035
000250      MOVE STAT-BIN TO S2-DISPL.                               003A
000260      DISPLAY DISPLY-STAT.                                     0041
000270 PARA1.                                                        004C 00
000280      STOP RUN.                                                004D
000290                                                               004E
000300                                                               004E
** CIS COBOL V4.4 REVISION 0                              URN MB/1178/BL
```

```
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=00537 CODE=00231 DICT=00206:17445/17651 GSA FLAGS=  OFF
```

FILEMARK UTILITY PROGRAM

The FILEMARK Utility program is used to write the trailer block that is required by CIS COBOL, in situations where it is not present. The program writes the trailer block on to the end of any specified file, without checking the internal format of that file. It is possible, therefore, to append a CIS COBOL trailer block to any CP/M file.

The program checks whether a CIS COBOL trailer block is already present, and if so, advises the operator by a displayed message (see ERROR CONDITIONS below), otherwise it appends a trailer block. FILEMARK can therefore be used to check for the presence of a trailer block.

OPERATING INSTRUCTIONS

Loading

FILEMARK is supplied as a directly loadable program to run under CP/M. It is loaded and run as follows:

```
FILEMARK [drive:] filename<<
```

where:

drive is a CP/M disk drive identifier i.e. A thru P.
filename is a standard CP/M filename in the format: name.ext

Running

The FILEMARK program is interactive in operation and displays messages during successful running as follows:

FILE FOUND; PROCESS BEGUN

CIS COBOL EOF RECORD SUCCESSFULLY ADDED TO FILE

FILE CLOSED; PROCESSING SUCCESSFULLY COMPLETED

Error Conditions

Any error condition that occurs during running of FILEMARK is conveyed to the user by a self-explanatory message. Error messages are as follows:

FILE NOT FOUND; RUN ABANDONED

indicates that the specified filename does not exist on the specified drive.

FILE IS MAX. SIZE THUS NO FURTHER RECORDSCAN BE ADDED; RUN ABANDONED

indicates that the addition of a trailer record would cause the file to exceed the maximum size allowed by CP/M.

ERROR DURING DISK READ; RUN ABANDONED

indicates that a read failure has occurred during the scan of the file.

ERROR WHEN WRITING CIS COBOL EOF RECORD; RUN ABANDONED

indicates that a write failure has occurred while attempting to write the trailer record.

ERROR DURING FILE CLOSURE; RUN ABANDONED

indicates that a CP/M file closure procedure has failed and the file is not usable.

OPEN FAILURE; RUN ABANDONED

indicates that a CP/M file opening procedure has failed and the file cannot be opened for processing.

CIS COBOL EOF RECORD ALREADY EXISTS.

indicates that the FILEMARK program has detected a standard CIS COBOL trailer label already present. The program terminates without writing anything to disk.

Note

The presence of more than one CIS COBOL trailer label at the end of a file can cause problems during processing. For normal use of the file, only one trailer label record is required.

Appendix G. EXAMPLE CONFIGURATION OF A HYPOTHETICAL CRT SPECIFYING TAB STOP MODIFICATION

```
B>CONFIG RUND.COM
************************************************
CIS COBOL RUN TIME SYSTEM (RTS) CONFIGURATOR V3.00
COPYRIGHT(C) 1978, 1982            MICRO FOCUS LTD
************************************************
VERSION n.n REVISION nnn USER REFERENCE NUMBER XX/nnnn/XX

THE RTS IS SUPPLIED WITH COLUMN TAB STOPS IN COLUMNS:-
08,16,24,32,40,48,56,64,72 DO YOU WISH TO MODIFY THESE? INPUT ONE OF THE
FOLLOWING: 'YES' 'Y' 'NO' 'N' >N

THE RTS PROVIDES THE FACILITY TO INCORPORATE ASSEMBLER CODE THAT MAY
BE ENTERED BY YOU FROM THE COBOL "CALL" VERB.
DO YOU WISH TO INCLUDE SUCH CODE?
INPUT ONE OF THE FOLLOWING:'YES' 'Y' 'NO' 'N'
>N
YOUR RUNTIME SYSTEM HAS BEEN CONFIGURED
```

Appendix H. EXAMPLE CONFIGURATION SPECIFYING USER SUBROUTINES

```
B>CONFIG RUNR.COM
************************************************
CIS COBOL RUN TIME SYSTEM (RTS) CONFIGURATOR V3.00
COPYRIGHT(C) 1978, 1982            MICRO FOCUS LTD
************************************************
VERSION n.n REVISION nnn USER REFERENCE NUMBER XX/nnnn/XX

THE RTS IS SUPPLIED WITH COLUMN TAB STOPS IN COLUMNS:-
08,16,24,32,40,48,56,64,72 DO YOU WISH TO MODIFY THESE? INPUT ONE OF THE
FOLLOWING: 'YES' 'Y' 'NO' 'N' >N

THE RTS PROVIDES THE FACILITY TO INCORPORATE ASSEMBLER CODE THAT MAY
BE USED BY YOU IN THE COBOL "CALL" VERB.
DO YOU WISH TO INCLUDE SUCH CODE?
INPUT ONE OF THE FOLLOWING:- 'YES'  'Y'  'NO'  'N'
>Y

IN THAT CASE WE MUST DECIDE WHERE IT IS TO GO.
DO YOU WISH TO USE THE DYNAMIC DEBUG FACILITY WITH THIS RTS,
INPUT ONE OF THE FOLLOWING:- 'YES'  'Y'   'NO'  'N'
>N

DO YOU WISH TO USE ANIMATOR?
INPUT ONE OF THE FOLLOWING:-  'YES'  'Y'  'NO'  'N'
>N

DO YOU WISH TO USE THE INDEXED SEQUENTIAL PACKAGE,
INPUT ONE OF THE FOLLOWING:-  'YES'  'Y'  'NO'  'N'
>N

HOW MANY BYTES DOES YOUR ASSEMBLER CODE USE, (ENTER A DECIMAL NUMERIC
STRING)
>264

PLEASE ARRANGE TO LOCATE YOUR CODE AT 419CH.

YOUR RUNTIME SYSTEM HAS NOW BEEN CONFIGURED
```

Appendix I. EXAMPLE CONFIGURATION IN WHICH NO CRT TAILORING IS PERFORMED

```
B CONFIG
************************************************
CIS COBOL RUN TIME SYSTEM (RTS) CONFIGURATOR V3.00
COPYRIGHT(C) 1978, 1982            MICRO FOCUS LTD
************************************************

ENTER THE FILE-NAME CONTAINING THE RTS TO BE CONFIGURED.

>RUNA.COM
VERSION n.n REVISION nnn USER REFERENCE NUMBER XX/nnnn/XX

THE RTS IS SUPPLIED WITH COLUMN TAB STOPS IN COLUMNS:-
08,16,24,32,40,56,64,72
DO YOU WISH TO MODIFYTHESE,
INPUT ONE OF THE FOLLOWING: 'YES' ' Y' 'NO' 'N'
>N
THE RTS PROVIDES IHE FACILITY TO INCORPORATE ASSEMBLER CODE THAT MAY
BE ENTERED BY YOU FROM THE COBOL "CALL" VERB.
DO YOU WISH TO INCLUDE SUCH CODE?
INPUTONE OF THE FOLLOWING:- ' YES' 'Y' 'NO' 'N'
>N

YOUR RUN TIME SYSTEM HAS BEEN CONFIGURED
```

Appendix J. EXAMPLE OF USER RUN TIME SUBROUTINES

```
;*************************************************************************
;*
;*
;* THIS IS AN EXAMPLE OF USER CALL CODE SUPPLIED PURELY FOR GUIDANCE OF THE
;* USER TO ENABLE THE MECHANICS OF CALL CODE INSERTION TO BE BETTER
;* UNDERSTOOD.
;* THE CODE IS DESIGNED TO BE A USEFUL EXAMPLE OF CALL, AND IF IMPLEMENTED
;* WILL ALLOW THE COBOL PROGRAMMER TO CREATE 16 BIT BINARY QUANTITIES FROM
;* UP TO 5 ASCII DIGITS, AND VICE VERSA. THE USE IS EXPLAINED IN MORE DETAIL
;* AT THE HEAD OF EACH ROUTINE.
;*
;* MICRO FOCUS LTD. HAS TAKEN EVERY PRECAUTION TO ENSURE THE ACCURACY OF
;* THESE ROUTINES, BUT CANNOT BE HELD LIABLE IN ANY WAY FOR ANY ERRORS OR
;* OMISSIONS IN THEM.
;*
;*************************************************************************
;* THE MODULE MUST BE LOCATED AT THE ADDRESS SPECIFIED BY CONFIGURATOR
;* WHEN THE RTS IN WHICH THE CODE IS TO RESIDE WAS CONFIGURED. (SEE
;* OPERATING GUIDE, SECTION 5).
;*
BASE:   EQU     04404H              ;REPLACE 04404H BY THE ADDRESS
                                    ;GIVEN BY CONFIGURATOR.
;*
;*
        ORG     BASE                ;SET THE BASE ADDRESS

;*
;*
;* NOW FOLLOWS THE CALL CODE IDENTIFICATION TABLE. THIS IS A TABLE OF
;* ADDRESSES OF THE ENTRY-POINTS TO THE ROUTINES. PRECEDED BY A BINARY
;* 8 BIT ITEM SPECIFYING THE HIGHEST AVAILABLE ROUTINE NUMBER
;*
;*
CALTOP: DB      MAXNO                   ;HIGHEST AVAILABLE CALL ROUTINE.
        DW      0                       ;CALL "00" (DOES NOT EXIST)
        DW      DECBIN                  ;CALL "01" - DECIMAL ASCII TO BINARY
        DW      BINDEC                  ;CALL "02" - BINARY TO DECIMAL ASCII
MAXNO:  EQU     ($-CALTOP-3)/2          ;LET THE ASSEMBLER DO THE WORK
;*
;*
;* NB. ALTHOUGH THE USE OF CALL "00" IN THE ABOVE EXAMPLE WOULD CAUSE
;* THE RTS TO ISSUE THE FOLLOWING ERROR:-
;*                   164 - CALL CODE DOES NOT EXIST
;* THE USER IS AT LIBERTY TO PROVIDE HIS OWN CODE. BY PLUGGING IN
;* THE APPROPRIATE ROUTINE ADDRESS.
;*
;* SIMILARY, OTHER ROUTINES MAY BE ADDED BY INCREASING THE NUMBER
;* OF ADDRESSES SPECIFIED.  IF THESE ARE ADDED BEFORE THE MAXNO EQUATE.
;* THEN THE BYTE AT CALTOP WILL ALWAYS BE CORRECT
;*
;*ROUTINE:        DECBIN
```

```
;*
;*CALLING SEQUENCE:
;*               CALL "01" USING PARA1 PARA2 PARA3.
;*
;*FUNCTION:      THIS ROUTINE CONVERTS A STRING OF DECIMAL (ASCII)
;*               DIGITS INTO A 16 BIT BINARY QUANTITY. IT IS VERY LOW LEVEL
;*               IN THAT IT EXPECTS A POSITIVE DECIMAL VALUE
;*
;*PARAMETERS:    PARA1 - ADDRESS OF LENGTH OF DECIMAL STRING
;*                       HELD AS 1 BYTE ASCII DIGIT (NOT CHECKED)
;*                       THIS ADDRESS WILL BE NO. 2 ON STACK
;*
;*               PARA2 - ADDRESS OF DECIMAL STRING
;*                       THIS ADDRESS WILL BE IN B,C ON ENTRY
;*
;*               PARA3 - ADDRESS OF RESULT AREA.
;*                       SPECIFIES A 2 BYTE AREA
;*                       THIS ADDRESS WILL BE IN D,F ON ENTRY
;*
;*VALUES RETURNED:       16 BIT RESULT IN PARA3
;*
;*
;*
DECBIN:
        POP     H               ;GET RETURN ADDRES OFF STACK
        XHTL                    ;GET ADDRESS OF PARA1
                                ;PUTTING RETURN ADDRESS BACK.
;*
        MOV     A,M             ;PUT IT IN ACCUMULATOR
        ANI     0FH             ;CONVERT TO BINARY

        PUSH    D               ;SAVE ADDRESS OF RESULT
        PUSH    B               ;MOVE STRING REF
        POP     D               ;  INTO D,E
        LXI     H,0             ;HL 1 BINARY ACCUMULATOR
DEC10:
        PUSH    PSW             ;SAVE THE COUNT
        DAD     H               ;BINARY ACCUMULATOR *2
        MOV     B,H             ;  AND MOVE IT INTO B,C
        MOV     C,L             ;

        DAD     H               ;BINARY ACCUMULATOR *4
        DAD     H               ;                   *4
        DAD     B               ;         *8 + *2 1 *10
                                ; (IE. 8X + 2X 1 10X)
                                ;------------------------

        LDAX    D               ;GET THE DECIMAL CHAR
        INX     D
        ANI     0FH             ;CONVERT TO BINARY CHAR
        MVI     B,0H
        MOV     C,A
        DAD     B               ;ACC + CHAR
        POP     PSW
        DCR     A               ;KEEP COUNT
        JNZ     DEC10
;*
;*              NOW STORE RESULT IN USER'S AREA
```

```
;*
        XCHG                    ;PUT RESULT IN D,F
        POP     H               ;GET ADDRESS OF RESULT AREA
        MOV     M,D             ;STORE MS BYTE
        INX     H
        MOV     M,E             ;STORE LS BYTE
        RET
;*
;*ROUTINE:        BINDEC
;*
;*CALLING SEQUENCE:
;*               CALL "02" USING PARA1 PARA2.
;*
;*FUNCTION:      TAKES THE BINARY QUANTITY ADDRESSED BY PARA1 AND CONVERTS
;*               IT INTO A 5 DIGIT DECIMAL (ASCII) NO. THE RESULT IS PLACED
;*               IN THE AREA SPECIFIED BY PARA2.
;*
;*PARAMETERS:    PARA1 - ADDRESS OF 16 BIT (2 BYTE) QUANTITY.
;*                       WILL BE IN REG B,C ON ENTRY
;*
;*               PARA2 - ADDRESS OF 5 BYTE RESULT AREA.
;*                       WILL BE IN REG D,E ON ENTRY
;*
;*VALUES RETURNED:
;*               5 DIGIT ASCII VALUE IN PARA2.
;*

BINDEC:
        PUSH    B               ;GET VALUE ADDR
        POP     H               ;  IN H,L
        MOV     B,M             ;VALUE
        INX     H               ;  IN
        MOV     C,M             ; B,C
        LXI     H,0             ;PUSH CONSTANTS
        PUSH    H               ;   ON TO
        LXI     H,-10           ;   STACK
        PUSH    H               ;   FOR USE
        LXI     H,-100          ;   DURING
;*
        PUSH    H               ;   BINARY TO DECIMAL CONVERSION
        LXI     H,-1000
        PUSH    H
        LXI     H,-10000
        PUSH    H
;*                              ;D,E - ADDRESS OF RESULT FIELD
CN25:
        MVI     A,30H           ;SET TALLY TO ASCII ZERO
CN30:
        POP     H               ;GET THE CONSTANT
        PUSH    H               ;RESTORE IT
        DAD     B               ;SUBTRACT FROM SOURCE OP
        JNC     CN40            ;ITS GONE NEGATIVE
        INR     A               ;INC TALLY

        PUSH    H               ;REPLACE B,C WITH
        POP     B               ;  NEW RESULT
        JMP     CN30
CN40:
```

```
        POP     H               ;CLEAR CONSTANT OFF STACK
        STAX    D               ;STORE TALLY IN RESULT FIELD
        INX     D               ;INC RESULT ADDR POINTER
        POP     H               ;ANY MORE CONSTANTS ?
        MOV     A,L
        ORA     H
        JZ      CN50            ;NO - FINISH OFF
        PUSH    H               ;YES - RESTORE IT
        JMP     CN25
CN50:
        MOV     A,C             ;INSERT UNITS
        ADI     B0H             ;CONVERT TO ASCII
        STAX    D
        RET                     ;RETURN
;*
;*
;*
```

Appendix K. EXAMPLE USE OF RUN-TIME SUBROUTINES

```
**CIS COBOL V3.3               CALLEX.CBL                          PAGE: 0001
**
000010  IDENTIFICATION DIVISION                                          000F
000020  PROGRAM-ID            CALL-EXAMPLE.                             000F
000030*                                                                  000F
000040*  This dummy program has been produced by Micro Focus             000F
000050*  as an example of the way in which the supplied CALL             000F
000060*  code routines may be used.                                      000F
000070*                                                                  000F
000080 DATA DIVISION.                                                    000F
000090 WORKING-STORAGE SECTION.                                          000F
000100 01  ROUTINE-NAMES                                                000F
000110     02  DECIMAL-BINARY          PIC X(2)  VALUE "01".            000F
000120     02  BINARY-DECIMAL          PIC X(2)  VALUE "02".            0011
000130*                                                                  0013
000140 01  PARAMETER-FIELDS                                             0013
000150     02 DECIMAL-NUMBER-LENGTH    PIC 9 VALUE 4.                   0013
000160     02 DECIMAL-NUMBER           PIC 9(4) VALUE 1234.             0014
000170     02 BINARY-RESULT            PIC X(2).                        0018
000180*                                                                  001A
000190     02 BINARY-NUMBER            PIC X(2) VALUE X"04D2".          001A
000200     02 DECIMAL-RESULT           PIC 9(5).                        001C
000210*                                                                  0021
000220 PROCEDURE DIVISION.                                               0000
000230* The following CALL will convert the 4 digit numeric field       0000
000240* DECIMAL-NUMBER to a 16 bit binary quantity in BINARY-RESULT.     0000
000250************************************************************      0000
000260    CALL DECIMAL-BINARY USING DECIMAL-NUMBER-LENGTH               0000
000270 DECIMAL-NUMBER BINARY-RESULT.                                     0000
000280************************************************************      000A
000290* BINARY-RESULT now contains the binary number 0402.              000A
000300*                                                                  000A
000310* The following CALL will convert the 16 bit binary field         000A
000320* BINARY-NUMBER to a 5 digit DECIMAL-RESULT                        000A
000330************************************************************      000A
000340    CALL BINARY-DECIMAL USING BINARY-NUMBER DECIMAL-RESULT.       0012
000350************************************************************      0012
000360* DECIMAL-RESULT now contains the value 01234.                    0012
**CIS COBOL V4.2  COMPILER COPYRIGHT (C) 1978 MICRO FOCUS LTD URN AA/3999/AB
**
**ERRORS=00000  DATA=00033  CODE=00043  DICT=00188:29624         END OF LIST
```

Appendix L. CONSTRAINTS

1. LINE SEQUENTIAL ORGANIZATION

   1.1 Any file that is intended ever to be dumped to a line printer should be given Line Sequential organisation, or sequential organization with BEFORE/AFTER clauses subsidiary to every WRITE statement.

   1.2 The Carriage Return (CR) and Line Feed (LF) characters that terminate a record (i.e . line) are exchanged by the Run Time System for padding with spaces on record input. Conversely trailing spaces are replaced by CR LF on record output.

   1.3 Line sequential files were designed to hold ASCII data only. COMP data that contains bytes with a value of 1AH or byte pairs of value 0D0AH must not be used in Line Sequential files.

2. FILE USAGE

   2.1 No more than 13 files may be open at any one time, excluding console input and output and line printer files. Remember that one Indexed Sequential file counts as two files when opened; also one of these 13 files is required for overlay loading or the calling of a sub-program. The overlay or sub-program file is open only during execution of the GO TO, PERFORM or CALL statement that causes the load. Note that another of the 13 files is required when a program is to be debugged using the ANIMATOR debugging tool.

   2.2 CIS COBOL source files under CP/M must not contain lines greater than 80 characters, nor must they contain "Tab" or any other control characters (i.e., 00H through 1FH).

3. UNSUCCESSFUL COMPILATION

   The generated intermediate code from any unsuccessful compilation should not be used. The intermediate code file should be deleted and the source code corrected and recompiled.

4. LIMITS NOT SPECIFIED IN THE DOCUMENTATION

   4.1 The maximum length of the Data Division in a CIS COBOL program is 32K bytes. The total length of all Linkage Section items is included in this figure although memory for them is not required at run time.

   4.2 The maximum length of the Procedure Division is 32K bytes although the actual amount of code is permitted to exceed this value if it is overlaid (segmented),

   4.3 The maximum number of records that may be accommodated in a relative or indexed file (assuming that disk space is available) is 65,535.

5. REDECLARATION OF AN INCORRECT DATA DECLARATION

   If there is an error in a data declaration the appropriate compiler error message for the error is displayed. Subsequent data-declarations may then be ignored by the compiler resulting in spurious error messages being generated if such data-items are referred to in the program.

6. INSPECT STATEMENT

   The following restriction applies to items to be inspected: They must not be in a Linkage Section

7. COMPILER SIZE INFORMATION

   The Data Division and code sizings output from the compiler do not take into account an overhead that is required if the program is segmented. This overhead is variable and an approximate guide is to allow 60 bytes overhead for the root segment and 30 bytes additional overhead for each overlay.

8. I-O ERROR HANDLING

CIS COBOL offers 3 mechanisms for handling file I-O errors:

a. Use of AT END or INVALID KEY clauses as appropriate.

b. Declaratives to handle AT END or INVALID KEY conditions where the appropriate clause has not been specified. (Note that no other errors will be passed to the Declarative routines).

c. Use of FILE STATUS key checks. If no status field is defined, status byte one '9' errors cause a message to be displayed on the console and the Run Time System to terminate. If a status field is defined, all errors are returned to the user program and it is the programmer's responsibility to check for any problems, and proceed accordingly. A sample program enabling the return value to be displayed as a decimal CP/M error number is provided in the CIS COBOL CP/M Operating Guide.

Colophon

This book was reconstructed into DocBook format from a scanned PDF found on the Internet. The PDF file already had OCR performed and the text was embedded in the file.

The original was published by Acorn Computers Limited in cooperation with the British Broadcasting Corporation.

Source version: 1.0.2