BRITISH BROADCASTING CORPORATION

MICROCOMPUTER SYSTEM

CIS COBOL™ with ANIMATOR™ and FORMS-2™ USER GUIDE

DATA DIVISION
WORKING-STORAGE
PERFORM
COMPUTE CONSOLE
PICTURE FILLER
RENAMES

CIS COBOL with ANIMATOR and FORMS-2 User Guide

Part no 409000
Issue no 1
Date February 1984

The product described in this manual and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:

Technical Enquiries
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN

First published 1984
Published by Acorn Computers Limited

CONTENTS

GETTING STARTED
WITH
CIS COBOL
ON CP/M

WELCOME TO CIS COBOL

This booklet is provided to start you off using CIS COBOL on your own computer.

CONTENTS

Issue 1

INITIAL CONSIDERATIONS

RUNNING CP/M

This handbook assumes that you have already familiarised yourself with CP/M. Full information on installing and running CP/M is contained in the CP/M manuals.

If you have not already done so, read through and follow the instructions contained in your CP/M manuals.

You will also need to use some of the software utilities described, in particular the diskette formatting utility, the CP/M System Copy utility, the diskette copy (duplication) program, the file copy utility and a suitable text-file editor.

CHECKING THE CONTENTS OF THIS PACK

We suggest that you now quickly check that your pack contains all the items it should. Do it before anything gets lost or borrowed.

In this binding, you should find in addition to this guide:

CIS COBOL Operating Guide for CP/M.

ANIMATOR Operating Guide.

FORMS-2 Utility Manual.

Also you will find the software on diskettes, labelled:

CIS COBOL

with a serial number printed on the label, in the format:

URN: xx/nnnn/Bx

where nnnn is a number between 0001 to 9999 and x is a unique identifier.

This URN (User Reference Number) is a serial number which is unique to you and is embedded in the software on your disks. You will need to quote this reference if you call or write to Acorn Computers for technical support, so make sure you know where it is.

The serial number of each of your disk label(s) should be the same as that on your Warranty Acknowledgement form on the CIS COBOL Release Disk sheet. Check now that it is. Please read this important paperwork enclosed with your disk and fill in NOW the Warranty Acknowledgement form and Software Licence Agreement. It will enable us to mail new product information and helpful newsletters to you as they become available, and also to help you individually if you need it through our Technical Support facilities.

In the event that your pack does not contain everything described here, return it to your supplier who will take appropriate action.

CIS COBOL START UP

GETTING READY TO RUN CIS COBOL

Your CIS COBOL software is supplied on diskettes which do not contain the CP/M bootstrap. The first thing to do therefore is to create a working CIS COBOL system by copying your CIS COBOL masters onto "working" diskettes.

You will need blank diskettes prepared to receive CIS COBOL by following the instructions for initializing new CP/M diskettes given in the CP/M Manual.

Copy both the CIS COBOL software and the CP/M "bootstrap" tracks onto your working diskettes. One way of doing this is as follows:

With your CP/M master system disk in drive A, run the CP/M "System copy" utility to put the CP/M bootstrap on each of the working disks. Then use the file-copy program to copy the CIS COBOL Master Diskette(s) in turn onto your working diskette(s), as described in your CP/M manual. Alternatively, you can duplicate your CIS COBOL master diskette (using the diskette copy program) and then use the CP/M "System Copy" Utility program.

Label your working diskettes clearly and replace your CIS COBOL Master(s) in the sleeve in which they came and store them in a safe place for back-up purposes.

CHECKING THE CONTENTS OF YOUR DISKS

Now quickly check the contents of each of your newly created disks, using the CP/M command DIR. The contents should include:

COMPILER	RUN-TIME SYSTEM	CONFIGURATOR UTILITY
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

CONFIGURATION

CIS COBOL is supplied with a configurator utility program called CONFIG. Not only are there many different systems available for CIS COBOL to run in under CP/M but there are many different CRT's that can be used with these systems.

CONFIG enables you to tailor the CIS COBOL run time system to a specific computer and CRT by answering a series of questions that are self-explanatory via the CRT. This has already been done for your system and you need take no further configuration action.

CONFIG functions are:-

1. To tailor the Run Time System (RTS) (or FORMS2.COM or any "linked" programs you have created), so that the interactive features of CIS COBOL can be used with your CRT.

2. Optionally to reserve an area within the RTS into which you may enter assembler or other language subroutines for use by the CALL statement in a CIS COBOL program. (You are unlikely to want to do this at this early stage).

3. If you wish, to form a library of up to 10 CRT definitions which may be used to automatically configure an existing RTS.

Detailed instructions for running CONFIG and an explanation of the questions asked are contained in your CIS COBOL Operating Guide.

COMPILING AND RUNNING THE DEMONSTRATION PROGRAMS

We have provided you with three domonstration programs. The source code for these programs is on the COBOL disk, in the files:

PI.CBL
STOCK1.CBL
STOCK2.CBL

PI.CBL — This program simply displays on the screen the mathematical constant PI to 12 decimal places and is the basic screen test for CIS COBOL DISPLAY.

STOCK1.CBL — This program should not be run until you are confident that PI.CBL is working correctly. It is the test for CIS COBOL ACCEPT, which provides the basic interactive functions, and Indexed file Input-Output.

STOCK2.CBL — This program uses a data file created by running STOCK1 and hence is dependent on having run that program successfully. The source code contains a deliberate error, which does not affect the program's execution but is there as an example of a CIS COBOL error message.

These programs introduce you to the simple compile and run development cycle of CIS COBOL. They also give an indication of the way in which very simple CIS COBOL programs can have sophisticated screen and file handling features.

PI.CBL

To compile PI, load into Drive A a disk which contains the file PI.CBL and also has some spare capacity for two other files which will be created during the compilation.

Now either reset the system or perform a warm boot by keying ctrl-C.

When you get the prompt:

```
A>
```

key in the command line:

```
COBOL PI.CBL
```

and press RETURN.

The compiler will not start executing. The first lines displayed immediately tell you that the compiler has been loaded and is executing:

```
**CIS COBOL V4.5 COPYRIGHT 1978, 1982 MICRO FOCUS LTD
**COMPILING PI.CBL
```

When the compilation is finished, the compiler reports the results as follows:

```
**ERRORS=ØØØØ DATA=nnnnn CODE=nnnnn DICT=nnnnn:nnnnn/nnnnn GSA FLAGS=OFF
```

Note: the number of bytes of dictionary space remaining depends on the available RAM of your computer.

The compiler will have generated two files: PI.LST, which is the list file, and PI.INT, which contains the intermediate code. You can examine the list file with the CP/M editor. Alternatively, we recommend that you list it out to the screen with the command:

```
A>TYPE PI.LST
```

and press RETURN.

If you have a line printer, you can obtain a printout by using the CP/M file copy utility. The command would be:

```
A>PIP LST:=PI.LST
```

To run the compiled PI program, load the Run Time System disk in Drive B, key ctrl-C, and execute the intermediate code by typing:

```
A>B:RUNA A:PI.INT
```

and press RETURN.

After a few moments the RTS identification line will be displayed then you will see the screen clear, the cursor will appear at the top left, and the "PI" screen will be displayed as illustrated below for the final term.

```
CALCULATION OF PI

NEXT TERM IS 0.000000000000

PI IS 3.141592653589
```

STOCK1.CBL

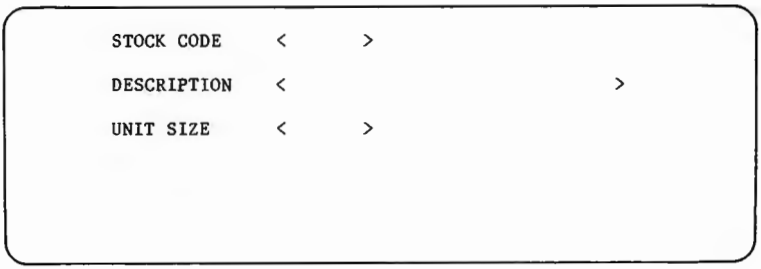
To compile STOCK1.CBL, follow the procedures outlined above for PI.CBL, substituting "STOCK1" for "PI".

Then run STOCK1.INT by keying the command line:

```
A>B:RUNA A:STOCK1.INT
```

then press RETURN.

After a few moments the RTS identification line will be displayed, you will see the screen clear, the cursor will appear at the top left, and then the "STOCK1" screen will be displayed as illustrated below.



At this point, the program is waiting for you to enter data via the keyboard using your normal cursor movement controls and the "accept-data" key (usually RETURN); as CONFIGured into your RTS.

Before entering any data, try moving the cursor around the screen using these keys.

Once the cursor is operating correctly, you may begin to enter data. The final two functions that you can check out are:

- left zero fill, which may be tested using the '.' on data entered into UNIT SIZE: keying <1. > should result in <0001>
- "accept-data", to enter your first screen-full of data (usually the RETURN key).

There are two cases when pressing the RETURN key will not result in the data being written away;

- If the UNIT SIZE is not numeric;
- If a record with this STOCK CODE number already exists on the file.

Reference to the listing of the source program will show you why.
The relevant statements are:

"IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR."

and

"WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR."

Thus, case 1) above is the result of an explicit test by the programmer for valid data input. Case 2) arises from the fact that the STOCK CODE is being used as the record key and duplicate deys are not permitted in the Indexed Sequential file to which these records are being written.

In order to subsequently run the STOCK2 program against the data files produced by STOCK1, you should make a note of the STOCK CODE numbers you enter to the program. We recommend you use six sequential numbers, say 11 through 16, entered in random order.

Important!

To terminate the run cleanly, you must key spaces into the STOCK CODE field and hit the "accept-data" key. The program tests for this end-of-run signal in the line:

"IF CRT-STOCK-CODE = SPACE GO TO END-IT."

STOCK2.CBL

Compile STOCK2 in the normal way. You will see that an error is displayed on the screen. This is a deliberate error inserted into the program to demonstrate CIS COBOL's method of handling compilation errors. This error does not affect the operation of the program.

Run the program, and retrieve the records you entered to the file using STOCK1, by entering into the STOCK CODE field the values you previously used. Again, spaces in the STOCK CODE field must be used to terminate the run.

NOTE:

A run-time error may occur if the files STOCK.IT and STOCK.IDX, generated and referenced by STOCK1 and STOCK2 have been corrupted; for example by a previous run of STOCK1 or STOCK2 which was incorrectly terminated. To recover from this situation use the CP/M ERA command to delete only the two files STOCK.IT and STOCK.IDX from your disk, and then start again with the STOCK1 program.

CREATING A CIS COBOL SOURCE PROGRAM

We suggest that the next step should be to write a small COBOL program of your own, then key it in using whatever editor is available on your system. Then compile and run it.

DO NOT USE THE TAB KEY when keying in a CIS COBOL program; it is not an ANSI character and is not accepted by the CIS COBOL compiler. Use only spaces to format the source program.

If you are not in the mood to write your own program, another way of creating a CIS COBOL source program is to use the CIS COBOL utility FORMS-2.

To help you use CIS COBOL remember you are provided with:

Operating Guide - how to use your CIS COBOL
ANIMATOR Operating Guide - describes the ANIMATOR debugging facility
FORMS-2 Utility Manual - describes the FORMS-2 screen handling facility

Don't forget to fill in your user registration form if you haven't already done so.

CIS COBOL

OPERATING GUIDE

For Use With the CP/M Operating System

Version 4.5

Micro Focus Ltd.

Issue 8
April 1982

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 (R) 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, DS127A5260-2760, 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.

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 of 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
TITLE
} Chapter Heading

ORDER ONE HEADING
ORDER TWO HEADING
Order Three Heading
Order Four Heading
} Text 3 lines down

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

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

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CHAPTER 2

COMPILER CONTROLS

CHAPTER 3
RUN TIME SYSTEM CONTROLS

CHAPTER 4

CIS COBOL APPLICATION DESIGN CONSIDERATIONS

CHAPTER 5

CONFIGURATION UTILITY

CHAPTER 6

INCORPORATING FORMS-2 UTILITY PROGRAM OUTPUT

CHAPTER 7

USING THE ANIMATOR UTILITY PROGRAM

APPENDIX A

SUMMARY OF COMPILER AND RUN-TIME DIRECTIVES

APPENDIX B

COMPILE TIME ERRORS

APPENDIX C

RUN TIME ERRORS

APPENDIX D

OPERATING SYSTEM ERRORS

APPENDIX E

INTERACTIVE DEBUG COMMAND SUMMARY

APPENDIX F

CP/M DISK FILES

APPENDIX H

EXAMPLE CONFIGURATION
SPECIFYING TAB STOP MODIFICATION

APPENDIX J

EXAMPLE CONFIGURATION
SPECIFYING USER SUBROUTINES

APPENDIX K

EXAMPLE CONFIGURATION IN WHICH
NO CRT TAILORING IS PERFORMED

APPENDIX M

EXAMPLE RUN TIME SUBROUTINES

APPENDIX N

EXAMPLE USE OF RUN TIME SUBROUTINES

APPENDIX P

CONSTRAINTS

TABLES

ILLUSTRATIONS

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 RUN TIME 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=ØØØØØ DATA=00661 CODE=00235 DICT=00409:nnnnn/nnnnn GSA FLAGS = 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.  URN XX/nnnn/XX
```

This clears the screen, followed by -

```
CALCULATION OF PI

NEXT TERM IS Ø.ØØØØØØØØØØØØ

   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.  URN XX/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 CONTROL FACILITIES 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  URN XX/nnnn/XX
```

This clears the screen followed by -

```
                GOODS INWARD

          STOCK CODE        <   >
          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.

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.

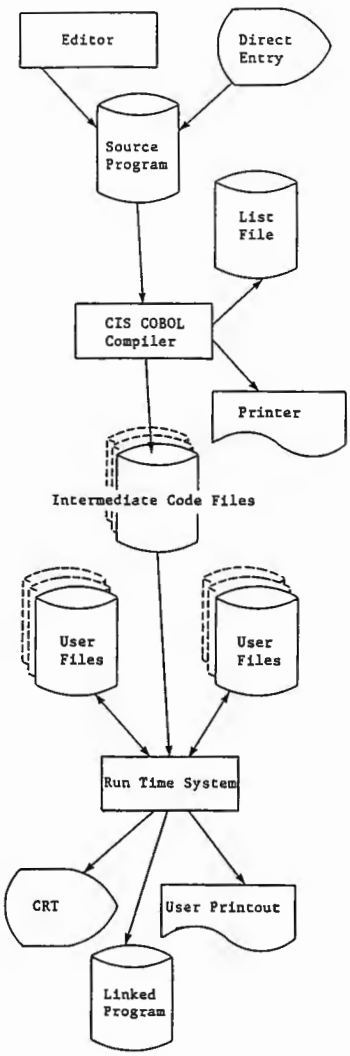


Figure 1 - 1. Program Development Cycle.

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)

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).

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=mmmmm: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 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=mmmmm: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.

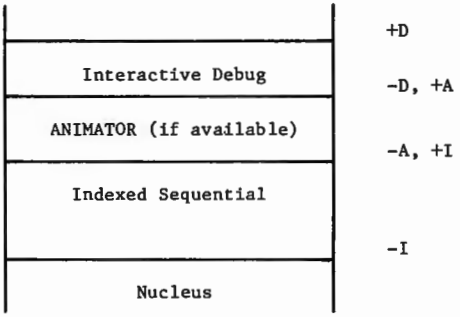


Figure 3-1. Run Time System Memory Layout.

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:

$$\left[\left(\{\pm\} \left\{ \begin{matrix} D \\ n1 \end{matrix} \right\} [[,]\{\pm\}n2]... \right) \right]$$

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 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, fro

which to load directly, to prevent it being overwritten on the next use of '=' parameter. The REName 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 REName 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 REName 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
```

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 addresses
               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(80) VALUE SPACE.     0036
     .                                  .
     .                                  .
```

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 X L 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 available 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.

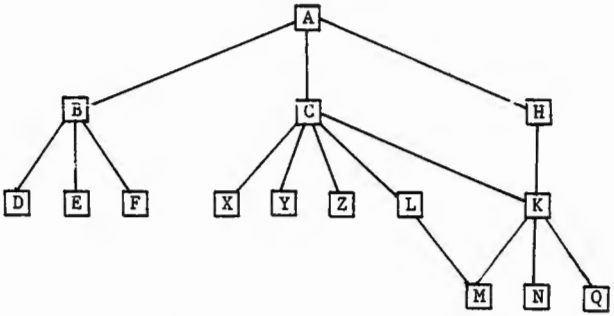


Figure 4-1. Sample CALL Tree Structure.

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 needs 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.

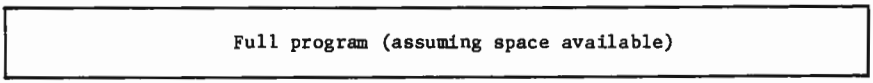
The CIS COBOL Run Unit is an application that is written in CIS COBOL and arranged into a number of separate CIS COBOL programs; these programs communicate with, invoke and cancel each other by use of COBOL "CALL" and "CANCEL" statements.

FORMAT OF CIS COBOL "CALL"

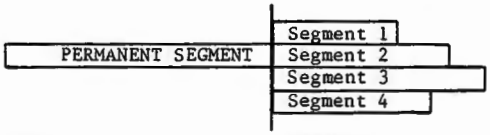
The general format of the CIS COBOL "CALL" and "CANCEL" statements are given in the CIS COBOL Language Reference Manual.

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.

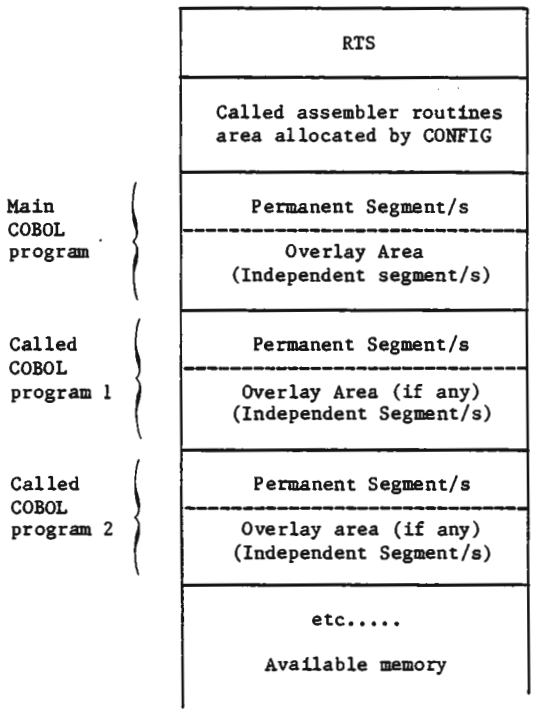


Figure 4-2. Memory Layout using Segmentation and Inter-Program Communication.

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 RUN TIME 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 SPACES TO 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 Ø     Highest subroutine number which is available
BYTE 1+2   Address of routine to satisfy CALL "Ø"
BYTE 3+4   Address of routine to satisfy CALL "1"
BYTE 5+6   Address of routine to satisfy CALL "2"
```

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

```
ORG       NNNNH          ;PROG ADDRESS FROM CONFIGURATOR
DB        2D             ;3 ROUTINES AVAILABLE
DW        ADDRØ          ;ADDRESS OF CALL "Ø" ROUTINE
DW        Ø              ;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 SUBROUTINES 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 from.

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 to 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(10).
          01   Devstr PIC X(6).
          01   Extstr PIC X(5).

          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 RUN TIME 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.

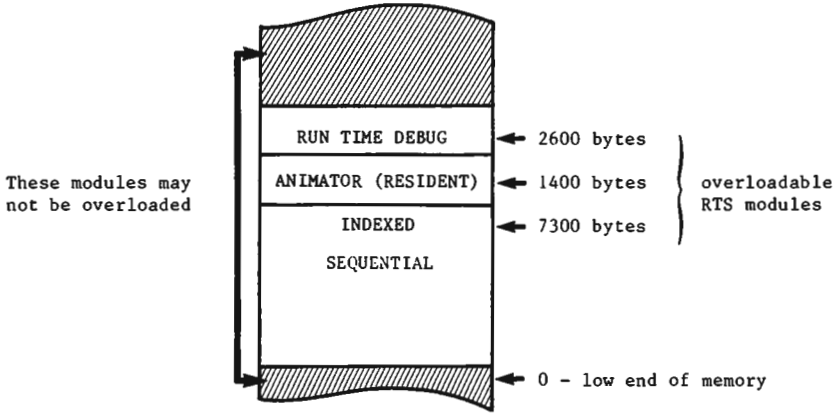
Having ascertained where the run time subroutines should be located the user is asked to specify the length of the subroutines in order that the load point for the intermediate code may be determined. It is important to ensure that the figures input for the length of the subroutines is the maximum that 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.

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 documentation. 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:

.DDC
.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.CBL 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<<
```

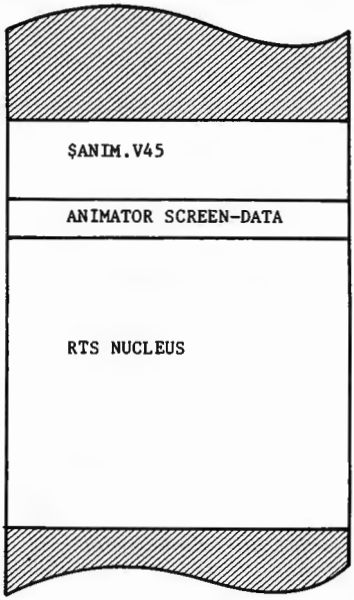
is invalid (only 1 load parameter allowed) and results in the message

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).

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, renumbering 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).

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 (6Ø).

ERRLIST

The listing is limited to those COBOL lines containing syntax errors 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, i.e. console or printer or a disk file.) The default is: source-file.LST.

For list to console use: LIST (CON:)
For list to line printer use: LIST (LST:)

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 in decimal for physical printer page throw. Default is PAGETHROW(12).

ANIM

The program is compiled for run-time 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
COMMIT
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

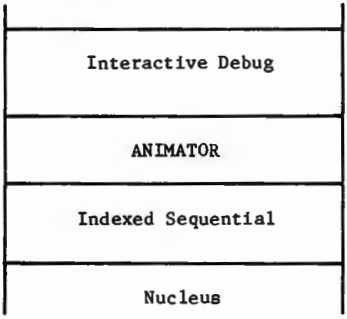
The command line syntax for running a CIS COBOL object program is as follows:

```
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:

$$\left[(\pm \begin{matrix} D \\ n1 \end{matrix} \left[[,][\ ] \pm n2 \right] ...) \right]$$

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 used 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
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
104		Procedure name has been declared twice
105		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 run-time 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. See Appendix F. 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 management failure
169	Invalid indexed 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 an active program
176	Illegal inter-segment reference
180	COBOL file malformed
194	File size too large (>0.5MB)
195	DELETE/REWRITE 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[1]

1 - 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-OUPUT 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: LST: :LP:	-	Line Printer	PUN: :TP:	-	Punch Device
			:HP:	-	High Speed Punch
:CI:	-	Keyboard Input	:RDR: :TR:	-	Reader Device
:CC:	-	Screen Output			
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)

[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.

Examples of 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 The 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 for 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 DESCRIPTION - 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.

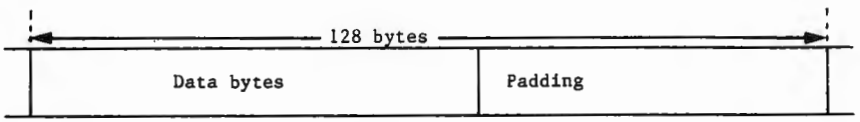
In this way different external files can be used as a common internal user file during any run of a program, but care is required to ensure that the correct file is allocated at any given time.

NOTE:

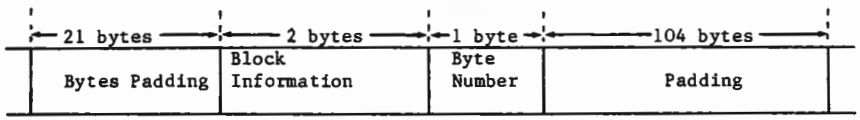
The device assignment B: in the file name above can be replaced by the format :F1: for compatability with other operating systems.

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:



LAST DATA RECORD



TRAILER RECORD

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 "[O]" parameter (alpha O).

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

$$\text{index levels} \qquad \log_k (\text{number of records})$$

$$\text{where } k = \frac{150}{\text{keylength} + 2}$$

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:

$$\text{data} = (\text{record length} + 2) * \text{max. no of records}$$

$$\text{index} = \frac{\text{no of records}}{k - 1} * 256 \quad \text{where k is as defined above}$$

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 be 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 RECORDS CAN 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 H

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 RUN TIME SYSTEM HAS BEEN CONFIGURED
```

APPENDIX J

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 RUN TIME SYSTEM HAS NOW BEEN CONFIGURED
```

APPENDIX K

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 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 RUN TIME SYSTEM HAS BEEN CONFIGURED
```

APPENDIX M

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 ADDRESS OFF STACK
        XTHL                    ;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              ;                   *8
         DAD     B              ;         *8 + *2 1 *10
                                ; (IE. 8X + 2X 1 10X)
                                ;-----------------------
```

```
        LDAX    D               ;GET THE DECIMAL CHAR
        INX     D
        ANI     OFH             ;CONVERT TO BINARY CHAR
        MVI     B,OH
        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,E
        POP     H               ;GET ADDRESS OF RESULT AREA
        MOV     M,D             ;STORE MS BYTE
        INX     H
        MOV     M,E             ;STORE IS BYTE
        RET
;*
```

```
;*ROUTINE:        BINDEC
;*
;*CALLING SEQUENCE:
;*                CALL "O2" 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 1 ADDRESS OF 16 BIT (2 BYTE) QUANTITY.
;*                        WILL BE IN REG B,C ON ENTRY
;*
;*                PARA2 1 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 COVERSION.
        LXI     H,-1000
        PUSH    H
        LXI     H,-10000
        PUSH    H
;*                              ;D,E 1 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      CN5O            ;NO - FINISH OFF
        PUSH    H               ;YES - RESTORE IT
        JMP     CN25
C5O:
        MOV     A,C             ;INSERT UNITS
        ADI     BOH             ;CONVERT TO ASCII
        STAX    D
        RET                     ;RETURN
;*
;*
;*
```

APPENDIX N

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 04D2.               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.       000A
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 P

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.

CIS COBOL

ANIMATOR

OPERATING GUIDE

Version 1.0

Micro Focus Ltd.

Issue 2
May 1982

WELCOME TO ANIMATOR!

This manual describes your ANIMATOR software which enables you to animate your Micro Focus COBOL application programs with the COBOL source code in front of you on the screen. You can activate parts of the program to suit yourself and at selected speeds and examine data items at break-points or even during running. You can examine and debug your programs using only the source code with no need to refer to details of the object code.

RELATED PUBLICATIONS

Descriptions of the Acorn COBOL Products are contained in the following manuals:

CIS COBOL Language Reference Manual
CIS COBOL Operating Guide specific to your Operating System.

AUDIENCE

This manual is intended for COBOL programmers using Micro Focus COBOL development systems.

NOTATION IN THIS MANUAL

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

CHAPTER n
TITLE
} Chapter Heading

ORDER ONE HEADING
ORDER TWO HEADING
Order Three Heading
} Text 3 lines down

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.

TABLE OF CONTENTS

CHAPTER 1

INTPODUCTION

CHAPTER 2

PREPARING FOR ANIMATION

CHAPTER 3

OPERATING COMMANDS

FIGURES

CHAPTER 1

INTRODUCTION

GENERAL DESCRIPTION

ANIMATOR is a COBOL oriented debugging tool for use with a Micro Focus COBOL product.

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.

Figure 1-1 shows how ANIMATOR fits into the COBOL system.

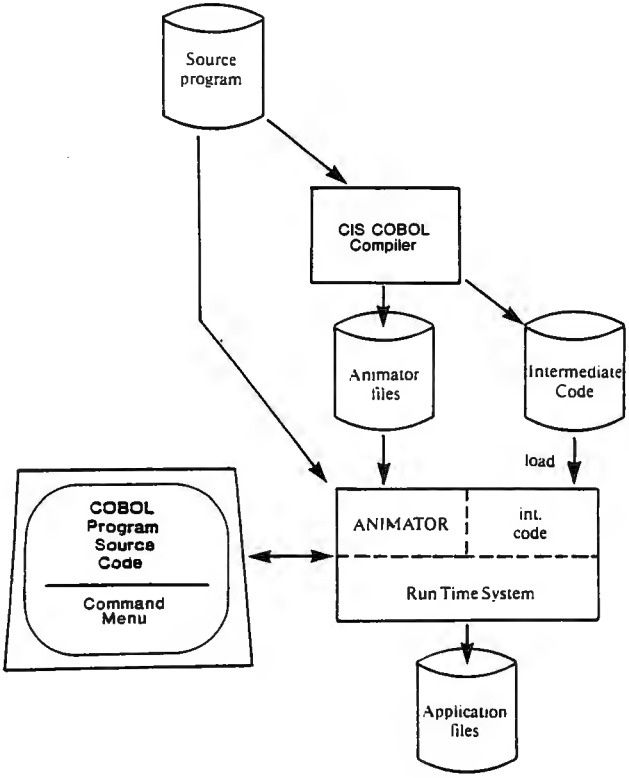


Figure 1-1. System Flowchart

GETTING STARTED WITH ANIMATOR

ISSUE DISK

Each user is provided with the program that makes up the ANIMATOR debugging tool on an ANIMATOR Issue Disk. This program is supplied in a file called $ANIM.V45.

FIRST STEPS

You will by now have working CIS COBOL system disks which you created from your CIS COBOL Issue Disk. In order to use ANIMATOR with CIS COBOL you need only copy the file $ANIM.V45 from your Issue Disk to the CIS COBOL system disk which you are using. Do this now with your CP/M copy software. (Remember you need to register with us as a user of ANIMATOR just as you did with CIS COBOL so fill in your User Registration form now if you have not already done so.)

To get off the ground with ANIMATOR this section uses the STOCK1 demonstration program that is described in Chapter 1 of the CIS COBOL Operating Guide. Getting started with ANIMATOR basically involves the same processes as were described there. You need to recompile STOCK1 for use with ANIMATOR and, of course, run it. As in the CIS COBOL example, it is assumed that all your files are on one disk in drive A.

Compilation

The compilation command is:

```
A>COBOL STOCK1.CBL ANIM RESEQ<<
```

The compiler will output a title and information just as it does normally. A resequenced listing will be produced in the file STOCK1.LST which you will find handy to print and have next to you, although it is less essential for debugging with ANIMATOR than with conventional debuggers as you will see. We include it here:

```
** CIS COBOL V4.5                    STOCK1.CBL                    PAGE: 0001
**
** OPTIONS SELECTED :
**      COPYLIST RESEQ ANIM
**
000010 IDENTIFICATION DIVISION.                                        0118
000020 PROGRAM-ID. STOCK-FILE-SET-UP.                                  0118
000030 AUTHOR. MICRO FOCUS LTD.                                        0118
000040 ENVIRONMENT DIVISION.                                           0118
000050 CONFIGURATION SECTION.                                          0118
000060 SOURCE-COMPUTER. MDS-800.                                       0118
000070 OBJECT-COMPUTER. MDS-800.                                       0118
000080 SPECIAL-NAMES. CONSOLE IS CRT.                                  0118
000090 INPUT-OUTPUT SECTION.                                           0118
000100 FILE-CONTROL.                                                   0118
000110     SELECT STOCK-FILE ASSIGN  STOCK.IT"                         0184
000120     ORGANIZATION INDEXED                                        0186
000130     ACCESS DYNAMIC                                              0186
000140     RECORD KEY STOCK-CODE.                                      0186
000150 DATA DIVISION.                                                  01BE
000160 FILE SECTION.                                                   01BE
000170 FD  STOCK-FILE; RECORD 32.                                      01BE
000180 01  STOCK-ITEM.                                                01BE
000190     02  STOCK-CODE PIC X(4).                                   01BE
000200     02  PRODUCT-DESC PIC X(24).                                01C2
000210     02  UNIT-SIZE PIC 9(4).                                    01DA
000220 WORKING-STORAGE SECTION.                                        01E0
000230 01  SCREEN-HEADINGS.                                           01E0 00
000240     02  ASK-CODE PIC X(21) VALUE "STOCK CODE     <    >".     01E0 00
000250     02  FILLER PIC X(59).                                      01F5 15
000260     02  ASK-DESC PIC X(16) VALUE "DESCRIPTION    <".           0230 50
000270     02  SI-DESC PIC X(25) VALUE "                    >".       0240 60
000280     02  FILLER PIC X(39).                                      0259 79
000290     02  ASK-SIZE PIC X(21) VALUE "UNIT SIZE      <    >".     0280 A0
000300  01  ENTER-IT REDEFINES SCREEN-HEADINGS.                       01E0 00
000310     02  FILLER PIC X(16).                                      01E0 00
000320     02  CRT-STOCK-CODE PIC X(4).                               01F0 10
000330     02  FILLER PIC X(76).                                      01F4 14
000340     02  CRT-PROD DESC PIC X(24).                               0240 60
000350     02  FILLER PIC X(56).                                      0258 78
000360     02  CRT-UNIT-SIZE PIC 9(4).                                0290 B0
000370     02  FILLER PIC X.                                          0294 B4
000380 PROCEDURE DIVISION.                                             0000
000390 SR1.                                                            001C 00
000400     DISPLAY SPACE.                                              001D
000410     OPEN I-O STOCK-FILE.                                        0020
000420     DISPLAY SCREEN-HEADINGS.                                    0024
000430 NORMAL-INPUT.                                                   0038 00
000440     MOVE SPACE TO ENTER-IT.                                     0039
000450     DISPLAY ENTER-IT.                                           003F
000460 CORRECT-ERROR.                                                  0056 00
000470     ACCEPT ENTER-IT.                                            0057
000480     IF CRT-STOCK-CODE = SPACE GO TO END-IT.                     006E
000490     IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.           0078
000500     MOVE CRT-PROD-DESC TO PRODUCT-DESC.                         0081
000510     MOVE CRT-UNIT-SIZE TO UNIT-SIZE.                            0087
000520     MOVE CRT-STOCK-CODE TO STOCK-CODE.                          008F
000530     WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR.              0095
000540     GO TO NORMAL INPUT.                                         00A1
000550 END-IT.                                                         00A4 00
000560     CLOSE STOCK-FILE.                                           00A5
000570     DISPLAY SPACE.                                              00A9
000580     DISPLAY "END OF PROGRAM".                                   00AC
000590     STOP RUN.                                                   00BD
** CIS COBOL V4.5 REVISION 0                          URN AA/0000/AA
** COMPILER COPYRIGHT (C) 1978,1982 MICRO FOCUS LTD
** ERRORS=00000 DATA=00768 CODE=00256 DICT=00362:21117/21479 GSA FLAGS=  0FF
```

Running With ANIMATOR

Enter the normal CIS COBOL run command:

```
A>RUNA +A STOCK1.INT<<
```

The run-time system software displays its header line:

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

This is replaced by the ANIMATOR header lines

```
CIS COBOL ANIMATOR V1.0

COPYRIGHT (C) 1981, 1982 MICRO FOCUS LTD

URN XX/nnnn/XX
```

This is in turn replaced by the COBOL source listing of the program on the screen, starting at the first line of the Procedure Division with the cursor positioned under the first verb. Available commands are displayed on a line below this listing display.

You are now ready to "animate" the STOCK1 program. To give you a "feel" for using ANIMATOR, some arbitrary instructions follow. You can, of course, carry out any of the commands given in Chapter 3. Chapter 3 describes all the commands and error lines in details.

Enter F for the Find command. The command line goes blank. Enter "CRT-STOCK-CODE" (the delimiters are necessary) and press RETURN. ANIMATOR will find the first reference to this data-name (line 000480).

Enter M to Monitor this data-item. The Monitor options are displayed. Press S for Set. A line of # symbols flashes on the screen. This denotes an error. The cursor is positioned at the end of the data-item. It must be at the first character to set monitoring. Move it there by use of the "character-back" key on your keyboard, and enter M and then S again. Nothing happens yet because the data-item is blank. As soon as you fill it, the contents will be displayed under the command line.

Enter B to set a Breakpoint (i.e. where the program will stop whenever it passes that point). The Breakpoint options are displayed. This time we realize we are in error. You can only set breakpoints at statement verbs. We therefore press the space bar to reset to the command line. Move the cursor down to the MOVE verb on line 00500 and position it under the M.

Enter B and then S to set the breakpoint here.

We now enter E for Execute and the Execute options are displayed. Press G for Go and execute starts at the default speed (4). Enter 1 to select the slowest execution speed so we can see what is happening. The cursor now runs through the COBOL code as it executes, bringing the COBOL program to life or "animating" it. You can halt animation at any time by pressing the space bar and restart by pressing G again. Note that you can enter G without the E(xecute) command.

The data entry screen is displayed and you enter values as you did when getting started with CIS COBOL.

After accepting your data the program stops at your breakpoint as set (line 00500). Notice that the value you entered as STOCK CODE is now displayed under the command line because you asked to monitor its content.

Enter G to go again and then select speed 9. Notice the difference in animation speed. Enter a new record into your data entry screen and the program stops at the breakpoint again with the new value showing in the monitored data-item under the command line.

Enter Z (for Zoom). Note that this is another E(xecute) command that you can enter directly. Animation stops and the program reverts to normal execution. If you enter a blank stock code it terminates.

You may now revert to the RUN +A command and repeat it, but this time using the commands from Chapter 3 and experimenting.

Appendix E contains a command summary and the CIS COBOL Pocket Guide contains a more concise summary.

CIS COBOL FACILITIES NOT AVAILABLE WITH ANIMATOR

This section describes constraints on the use of a CIS COBOL program with ANIMATOR.

ANSI COBOL DEBUG MODULE

The COBOL Debug module provides facilities for monitoring procedures by means of code activated by an object time switch. Object time activation of this code is inhibited by the use of ANIMATOR, therefore this object time switch should not be set.

ALTER STATEMENT

ANIMATOR makes use of a paging mechanism so that the animation of large programs is not prevented by the presence of the ANIMATOR module.

This has the effect that the state of ALTERed GOTO statements cannot be predicted with any certainty. Therefore care must be exercised on the use of ANIMATOR with programs containing ALTER statements.

However, note that ANIMATOR provides facilities enabling the effect of such control transfers to be seen, and to reset the point of program execution if required.

COMMAND LINE PARAMETERS

CIS COBOL provides the facility to read parameters entered on the command line by means of the first ANSI-format ACCEPT statement executed.

This facility is inhibited when animating; such parameters must be entered at the time the ACCEPT statement is executed.

CHAPTER 2

PREPARING FOR ANIMATION

COMPILATION

The steps involved in compiling a program are described in detail in the CIS COBOL Operating Guide.

Note that if it may be required to animate the program the compiler directive ANIM must be specified.

Two other compiler directives can be particularly useful; these are COPYLIST and RESEQ. Use of these directives will ensure that the compilation listing matches the ANIMATOR screen display.

The use of the ANIM directive causes certain files required for animation to be produced in addition to the normal intermediate code file(s). These will have the extensions .DDC, .SDB and .SCP. The main source file must have the extension .CBL.

Note that the program can be run without animation and will behave exactly as if the ANIM directive had not been used for compilation.

RUNNING

The invocation of ANIMATOR to run a program is described in the CIS COBOL Operating Guide.

The "+A" directive specifies that animation is required. The ANIMATOR module itself must be present on the default drive. The special ANIMATOR files (.DDC, .SDB, .SCP), along with the main source file (which must have the extension .CBL), should be present on the same device as the intermediate code if animation of the particular program is required. If any of these files is missing the program is executed normally - this allows particular programs within a hierarchy to be selected for animation.

Any library (COPY) files should also be present, on the device specified by the COPY statement or, if none is specified, on tne same device as the intermediate code.

CHAPTER 3

OPERATING COMMANDS

When a program is loaded for animation, as described in Chapter 2, the first screenful of procedural code is displayed with the cursor positioned on the first executable statement. Execution will not commence until initiated by a command as described in this Chapter.

At the bottom of the screen is a delimiting line of hyphens followed by the main command menu:-

B(rk-pnts) D(isp) E(xec) F(ind) L(evel) M(on) P(-c) Q(uery) S(creen) U(ser)

Commands are entered by simply pressing the appropriate key. If an invalid entry is made at any time ANIMATOR indicates this by briefly replacing the prompt line with a line of hash (#) signs and "bleeping".

Several of the main commands merely serve to invoke subordinate menus of more detailed commands. For convenience some of the more commonly used of these subordinate commands can also be entered directly whilst the main prompt line is displayed (see detailed descriptions). After execution of a subordinate command the main prompt line is redisplayed.

Some of the commands require that the cursor is first moved to point to the appropriate position in the displayed source code. The main prompt line shown above is displayed whenever execution is suspended. Whilst this prompt line is displayed the cursor can be moved over the screen using the normal cursor control keys. For details of these (and how to change them) refer to the CIS COBOL Operating Guide.

Note that with ANIMATOR the back field and forward field function keys are used to move the cursor up and down within a screen column. In addition RETURN will move the cursor to the start of the next line.

DETAILED COMMAND DESCRIPTIONS

The main commands fall into 4 broad categories:

- Source code "window" manipulation (S,F)
- Execution and animation control (B,E,L,P)
- Examination and amendment of data (D,Q,M)
- User screen display (U)

These commands may only be entered when the main prompt line is displayed.

SOURCE CODE WINDOW MANIPULATION

ANIMATOR uses the screen as a window into the source code text. These commands allow you to reposition the source code window to any point within the source COBOL program. Note that this in itself does not affect the point at which execution will be resumed.

ANIMATOR automatically displays resequenced line numbers against the source text. These will be the same as those appearing on the compilation listing if the directives RESEQ and COPYLIST (if appropriate) were specified to the compiler.

The S(creen) Command

Press S and the following subordinate menu is displayed:

SCREEN - N(ext) P(revious) T(op) E(nd) V(iew) H(alf) F(ull) =/+/-

These commands reposition the window to display a different part of the source text.

Press the appropriate key where:

N displays Next screen from source text. Note that this overlaps the current screen by two lines.

P displays Previous screen from source text.

T displays screen at Top of source text.

E displays screen at End of source text.

V repositions window so that the source line indicated by the cursor is on the third line. Note: the cursor must be positioned before pressing S.

H Splits screen in Half (ie into two windows) with a dividing line of hyphens. The lower window is positioned to show the top of source text. Note: Subsequent screen commands operate on the window in which the cursor is positioned.

F restores Full screen display (single window).

=n repositions the window such that the nth source line is aligned at the third screen line. Note: n is equivalent to the displayed line number omitting the trailing zero.

+n moves the window forward n lines.

-n moves the window back n lines.

Note: =, +, - all position the cursor for entry of a numeric quantity followed by RETURN.

There is one special case of the screen command. If there is a split screen display and the cursor is positioned on the dividing line of hyphens, then when S is pressed the following subordinate menu is displayed:

SCREEN DIVIDER - U(p) D(own)

These commands allow the relative size of the two windows to be altered.

U moves the screen divider Up one line.

D moves the screen divider Down one line.

The F(ind) Command

This command instructs ANIMATOR to search forward from the current cursor position through the source text for a specified string of characters. If found the screen window is positioned with the line containing this string as the third screen line and the cursor is positioned following the string. If not found the main screen display remains unchanged, but ANIMATOR indicates the failure by bleeping and restoring the main prompt line.

Press F and the cursor is positioned for entry of:

Either

"string" followed by RETURN

or

"string"M followed by RETURN

where:

- any character not forming part of the string can be used in place of ".
- string is any sequence of characters (including spaces); it need not be a complete word.
- the optional M instructs ANIMATOR to only search the main source file and not any library (COPY) files.

Note:

ANIMATOR only examines columns 7 - 72 of the source text, the displayed line numbers are ignored.

EXECUTION AND ANIMATION CONTROL

These commands allow initiation and control of program execution, also control of the degree of animation.

The B(reakpoint) Command

Press B and the following subordinate menu is displayed:

BREAK-POINTS - S(et) U(nset) C(ancel) eX(amine)

These commands allow break-points to be set at which execution will halt automatically. Some of these commands require the cursor to be positioned to point to the relevant COBOL statement. The cursor must be positioned on the first character of the COBOL verb, prior to pressing B.

Up to 4 break-points may be set concurrently.

Press the appropriate key, where:

S Sets a break-point at the statement pointed to by the cursor.

U Unsets the break-point pointed to by the cursor.

C Cancels all break-points currently set.

X eXamines break-points by repositioning the window within the source code and positioning the cursor at the relevant statement. Successive calls on this function will move the cursor to point to each break-point in turn.

The E(xecute) Command

Press E and the following subordinate menu is displayed:

EXECUTE - X(single step) sK(ip) I(till next If) G(o) Z(oom) S(top run)

These commands initiate execution, with or without animation, in a variety of ways.

NOTE:

Any of these commands except for S(top run) may also be entered directly against the main prompt line without the preceding E.

Press the appropriate key, where:

X executes a single COBOL statement and moves the cursor to the next statement.

K skips a single COBOL statement, without executing it, and moves the cursor to the next statement.

Note: If the final statement of a PERFORMed paragraph is skipped, control does not exit from the PERFORM but passes to the next statement in the source code.

I Executes without animation up to the next IF statement, at which point execution halts and the cursor is repositioned at this IF statement.

G Initiates animated execution. As each statement is executed the cursor is moved to the next statement in the source code. The speed of animated execution can be varied by typing a digit from 1 to 9 (1 = slowest, 9 = fastest). The speed may also be entered before initiating execution. Execution proceeds until halted as described below.

Z Initiates execution without animation (Zooms). Upon reaching the first DISPLAY UPON CRT or ACCEPT FROM CRT the user screen is displayed, replacing the source code, and remains on the screen until execution is halted as described below.

S Stops execution after displaying the current user screen.

After initiation by one of the above commands, execution proceeds as described above, unless it is halted in one of the following ways:

1. If the space-bar is pressed execution is immediately halted.

2. If a previously set break-point is reached execution is automatically halted.

3. If execution is about to "drop through" the end of the program the following prompt is displayed:

   WARNING - Next instruction is implied STOP RUN - S(top run) C(ontinue)

   Press any key to restore the prompt line and regain control. If execution is restarted without first resetting the program counter the program will terminate; see the P(-C) command

4. If a run-time error occurs the following prompt is displayed:

   RTS ERROR: nnn - S(top) C(ontinue)

   Press S to stop, or C to regain control.

   Error numbers are detailed in the CIS COBOL Operating Guide.

The L(evel) Command

Animation normally traces execution into any level of nested PERFORM. This command allows a "threshold" level to be set at any level of nesting, such that any PERFORM's subordinate to this level are treated as a single statement for animation purposes, i.e. the cursor will not be moved into PERFORMed procedures below the threshold level.

Press L and the following subordinate menu is displayed:-

PERFORM LEVEL - S(et) U(nset) E(xit)

Press the appropriate key, where:

S sets the threshold level at the current level

U unsets the threshold level, restoring animation at all levels.

E completes execution of the current PERFORM without animation, repositioning the cursor to the statement following the PERFORM, and then sets the threshold level at this point.

The P(-C) Program Counter Command

This command provides facilities to ascertain the point at which execution will start (or resume), or to change this point. To change the restart point it is first necessary to place the cursor at the required position.

Press P and the following subordinate menu is displayed:

PROGRAM COUNTER - W(here) R(eset)

Press the appropriate key, where:

W repositions the screen window as necessary and positions the cursor at the next statement to be executed. This is useful as a check after use of the source screen manipulation commands, but note that it is not necessary since ANIMATOR will resume execution at the correct position unless the following command is used.

This command may also be entered against the main prompt line (without the preceding P).

R resets the execution start point (program counter) to the current cursor position. Before pressing P the cursor should be positioned on the first character of an executable statement (ie a COBOL verb).

EXAMINATION AND AMENDMENT OF DATA

These commands provide facilities for the examination and amendment of the contents of specified data items. Selected data items are indicated by either specifying the dataname or pointing with the cursor.

The D(isplay) Command

This command allows display/amendment of a data item referenced by typing the name.

Press D and the cursor is positioned for entry of the required data-name followed by RETURN.

The value of the specified data item is displayed with appropriate conversion for usages COMP and COMP-3 (see the CIS COBOL Language Reference Manual). For alphanumeric or group items, non-ASCII characters are displayed as ^.

For signed numeric data items a leading sign is displayed.

Following display of the data item the cursor is positioned to the next line for entry of a replacement value if required.

If no replacement is required simply press RETURN, otherwise enter the new value in COBOL literal format (ie alphanumeric literals must be in quotes). Replacement is performed in accordance with the rules for the COBOL MOVE statement.

NOTE:

> Only up to 80 characters will be displayed. The attempted replacement of a longer data item results in padding with spaces in accordance with the rules for a MOVE statement (See the CIS COBOL Language Reference Manual).

The Q(uery) Command

This command allows display/amendment of a data item referenced by pointing with the cursor.

Position the cursor to any occurrence of the data-name within the source code, then:

> Press Q

The data item is displayed and may be amended as described for the D(isplay) command.

The M(onitor) Command

This command enables automatically repeated display of a single specified data item (without amendment).

Press M and the following subordinate menu is displayed.

MONITOR - S(et) U(nset) N(ame)

Press the appropriate key, where:

S sets the monitor on the data item referenced by the cursor. The cursor must be moved to point to any occurrence of the data-name before pressing M.

U unsets the monitor.

N positions the cursor for entry of the required data-name followed by RETURN.

The monitored item is redisplayed after each animation "step" - eg after each statement if in "GO" mode.

USER SCREEN DISPLAY

During animation any ANSI format ACCEPT/DISPLAY statements are diverted to the bottom of the screen so that they do not interfere with ANIMATOR's use of the screen.

This approach is not possible for the ACCEPT FROM CRT and DISPLAY UPON CRT extensions for full screen interaction. Therefore all such ACCEPT/DISPLAY screen data is buffered internally.

This "user screen" automatically replaces the source code window display when an ACCEPT is executed, so that the user can interact with the screen in the normal way.

Additionally the user screen can be examined by means of the U(ser) command.

The U(ser) Command

Press U and the user screen is displayed, replacing the source code window display. If the Z key is pressed at this point, execution without animation (Zoom) is initiated, and the user screen remains on display. Otherwise, if any other key is pressed, the display reverts to the source code.

APPENDIX A

ANIMATION OPTION COMMAND SUMMARY

This Appendix is a summary in alphabetical order (rather than logical order) of the commands that the user can select from the CRT menu that is displayed when the Animation Option is invoked in the Run command. A brief summary of each command is given. A summary is also contained in the CIS COBOL Pocket Guide.

B - The Breakpoint command allows breakpoints to be set by the user at which execution will halt automatically.

The Breakpoint command offers a menu of four options:

S - Set breakpoint at statement currently pointed to by the cursor.

U - Unset the breakpoint currently pointed to by the cursor.

C - Cancel all breakpoints.

X - Examine next breakpoint. Can be used successively to examine all set breakpoints.

D - The Display command enables display and/or amendment of the named data-item.

E - The Execute command is used to specify the way in which the user requires execution of the program. On entry, the command displays an option menu as follows:

X - EXecutes a single COBOL statement and moves the cursor to the next statement.

K - SKips a single COBOL statement without execution and moves the cursor to the next statement.

I - Executes (without Animation) to the next If statement, halts and positions the cursor at this statement.

G - Initiates execution (GO) with Animation. As each statement is executed (at a specified speed) the cursor is moved to the next statement in the source code. Speed of execution can be set before entering G or altered after entering G by entry of a numeric character 1 thru 9, where 1 is slowest and 9 is fastest.

Z - Initiates execution without Animation activated (Zooms)

S - Stops execution after displaying the current user screen.

F - The Find command searches from the current cursor position through the source text for a specified string of characters.

L - The Level command allows a "threshold" level to be set at any level of nested PERFORM such that any PERFORM statements at this level are treated as a single statement for Animation purposes, i.e., the cursor will not be moved into PERFORMed procedures below threshold level.

S - Sets the threshold at the current level.

U - Unsets the threshold level and restores Animation at all levels.

E - Completes the current PERFORM without further Animation, then sets the threshold at the level of the statement immediately suceeding the current PERFORM statement.

M - The Monitor command enables automatic repeated display of a single specified data-item (without amendment) during program execution.

S - Sets the Monitor on the data-item at the current cursor position.

U - Unsets the Monitor.

N - Positions the cursor for operator entry of the Name of the data-item to be Monitored.

P - The program-counter command provides facilities to ascertain the point at which execution will start (or resume), or to alter this point. On entry, the P command displays an option menu as follows:

W - Where - repositions the cursor at the next statement to be executed.

R - Resets the execution start point to the current cursor position.

Q - The Query command allows display and/or amendment of the data-item pointed at by the cursor when Q is entered.

S - The Screen command repositions the screen window to display a different part of the source text as follows:

N - displays Next screen from source text.

P - Displays Previous screen from source text.

T - Displays screen at Top of source text.

E - Displays screen at End of source text.

V - Repositions window so that the source line indicated by the cursor is on the third line. Note: the cursor must be positioned before pressing S.

H - Splits the screen in half (i.e. into two windows) with a dividing line of hyphens. The lower window is positioned to show the top of source text. Note: Subsequent screen commands operate in the window in which the cursor is positioned.

F - Restores Full screen display (single window).

=n - repositions the window such that the nth source line is aligned at the third screen line.

+n - Moves the window forward n lines.

-n - Moves the window back n lines.

Note: =, +, - all position the cursor for entry of a numeric quantity followed by RETURN.

There is one special case of the screen command. If there is a split screen display and the cursor is positioned on the dividing line of hyphens, then when S is pressed the following subordinate menu is displayed:

SCREEN DIVIDER - U(p) D(own)

These commands allow the relative size of the two windows to be altered.

U moves the screen divider Up one line.

D moves the screen divider Down one line.

U - The User command displays the current user screen replacing the source code window display until any key is pressed.

CIS COBOL

FORMS-2 UTILITY

MANUAL

Version 1.3

Issue 9
November 1983

WELCOME TO FORMS-2!

This manual describes your FORMS-2 package to design, create and edit interactive screen layouts for use in CIS or L/II COBOL application programs.

As the product is compatible with both the Micro Focus base COBOL products CIS COBOL and LEVEL II COBOL, the common abbreviation "CIS or L/II COBOL" is used throughout this manual.

Most application programs will be written separately according to individual requirements but FORMS-2 can automatically generate a powerful user-oriented indexed data entry and filing system maintained by use of the screen layouts (forms).

If you want to generate an automatic indexed filing system Index program and then use it, carry out Chapter 1 procedures, read Chapters 2 and 3 briefly and then refer straight to Chapter 8 before performing the comprehensive sample run in Chapter 9.

If you wish to generate some screen layouts, carry out the procedures in Chapter 1, read briefly Chapters 2 and 3, and then perform the comprehensive sample run in Chapter 7.

To gain a further understanding of the FORMS-2 features and FORMS-2 operation read Chapters 1, 2 and 3 in detail. Chapters 4, 5, 6 and 8 are descriptions of FORMS-2 output file contents and relate directly to optional features that can be used. Read these Chapters only if you require the feature described.

It is recommended that, in any case, you run the sample programs before using FORMS-2 to generate your own forms or indexed files.

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. The carriage return (CR) or equivalent data input terminator key is referred to throughout this manual as the RETURN key.

3. The symbol << in this manual indicates that the RETURN key must be pressed once.

4. The space bar or key is referred to throughout this manual as the SPACE key.

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

CHAPTER n
TITLE — Chapter heading

ORDER ONE HEADING
ORDER TWO HEADING
Order Three Heading
Order Four Heading — Text two lines down

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.

RELATED PUBLICATIONS

For details of CIS or L/II COBOL operation and language respectively, refer to the documents:

CIS or LEVEL II COBOL Operating Guide
CIS or LEVEL II COBOL Language Reference Manual

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

CHAPTER 2
INITIALISATION PHASE

CHAPTER 3
WORK PHASE

CHAPTER 4
DATA DESCRIPTIONS

CHAPTER 5
THE CHECK-OUT PROGRAM

CHAPTER 6
SCREEN IMAGE FILE

CHAPTER 7
FORMS-2 USER SCREEN
GENERATION EXAMPLE 4-48

CHAPTER 8
INDEX PROGRAM

CHAPTER 9

TABLES

CHAPTER 1

INTRODUCTION

GENERAL DESCRIPTION

FACILITIES

The FORMS-2 package is an extension to the CIS or L/II COBOL software development systems which enable interactive creation and editing of data entry screens for applications programs at a CRT. The package provides four powerful facilities to aid the design and development of interactive applications written in CIS or L/II COBOL:

* Translation of user screen layouts into COBOL record descriptions for inclusion in COBOL applications programs.

* Verification of user screen layouts in a Check-Out program before their incorporation in an application program.

* Retention of exact screen images of the user screens in disk files for subsequent editing and printing.

* Generation of an entire COBOL program to allow data capture, update and interrogation by means of application screens and an indexed sequential file.

OUTPUTS

The user can choose any valid combination of the above facilities and dependent on the options selected FORMS-2 will automatically produce the following four types of disk output file:-

* A source file of CIS or L/II COBOL Data Description Statements defining the user designed screens (forms). These statements can subsequently be included in a CIS or L/II COBOL application program using the COPY verb. The file is generated as filename.DDS.

* A source file of a Check-Out program incorporating the Data Description Statements defining the user's screens. After compilation the user is able to verify the data entry form prior to building the actual application. The file is generated as filename.CHK.

* Screen Image files of exact copies of the user defined forms. The files are generated as filename.Snn (nn is 00 to 99).

* A file of the source of an Index program based on the user screen. After compilation the generated program can be used for the storage, retrieval, updating and deletion of data entered via the users form. The file is generated as filename.GEN.

PHASES

FORMS-2 processing is divided between a number of logically distinct units but two main phases can be identified - Initialisation Phase and Work Phase:

Initialisation Phase

The Initialisation phase is passed through once only and establishes the characteristics of this particular run of the program. It is a series of screens containing self-explanatory prompts to which the user replies as necessary.

Work Phase

At least two Work Phases are passed through for each data entry screen required by the application.

The FORMS-2 screen is analogous to a paper form in which the printed fixed text is used as a guide to entering the variable data in the spaces provided. To the human eye it is obvious where the variable data entry areas occur on a form but the computer needs to have these areas defined explicitly. There are, therefore, two types of Work Phase: one in which fixed text is specified and one in which variable data fields are specified.

OPERATOR INTERFACE

FORMS-2 is written in COBOL and uses the ACCEPT and DISPLAY extended CIS and L/II COBOL features. These two verbs are described in the CIS or L/II COBOL Language Reference Manuals, as are the cursor control features.

Advantages of this CRT interface are:

* Corrections can be directly overtyped
* Numeric fields accept only numeric characters
* The full stop or period (.) when keyed in a numeric field automatically zero-fills the field from the left.

CURSOR MOVEMENT FACILTIES

The user has the ability to move the cursor quickly and easily about the screen using the standard CIS and L/II COBOL ACCEPT statement interactive facilities.

The general functions of the cursor control keys are summarised in the following table. The actual key may vary from that shown dependent on the particular CRT being used.

Table 1-1. Cursor Control Keys.

KEY:	FUNCTION:
←	Position cursor right one data character
→	Position cursor left one data character
↓	Position cursor at start of succeeding data field
↑	Position cursor at start of preceding data field
ctrl ↑	Move cursor to start of first data field, referred to as HOM or ↖ throughout this manual.
TAB	Position to next tab stop

In Edit Mode the screen is split into fields 80 characters long. There are 22-24 fields per screen depending on the number of lines specified at Screen I01.

Corrections to text may be made by overtyping or by switching into Command Mode and using the editing commands.

GETTING STARTED

The FORMS-2 Issue Disk is a formatted data disk formatted to the requirements of CP/M.

Load the Operating System system disk and the FORMS-2 Issue Disk, bootstrap load the Operating System as usual, and obtain a directory of the FORMS-2 Issue Disk. All the files listed in Table 1-2 should be present on the disk. If they are not, or any problems arise, contact your Distributor.

Table 1-2. FORMS-2 Issue Disk Contents

FILE	CONTENTS	DESCRIPTION
FORMS2.COM FORMS2.I51 FORMS2.I52 FORMS2.I53	FORMS2 program	The manual
FORMS2.I01 FORMS2.I02	Initialisation Phase Screens	Chapter 2
FORMS2.W01 FORMS2.W02	Work Phase Screens	Chapter 3
FORMS2.H01 FORMS2.H02 FORMS2.H03 FORMS2.H04	Help Screens	Chapter 3 (? Command)
FORMS2.CH1 FORMS2.CH2	Check-Out Program Skeleton	Chapter 5
FORMS2.GN1 FORMS2.GN2	Index Program Skeleton	Chapter 8

Having validated that all the files are present copy all of them to a working disk using the standard copy program provided, and store the Issue Disk as a back-up master.

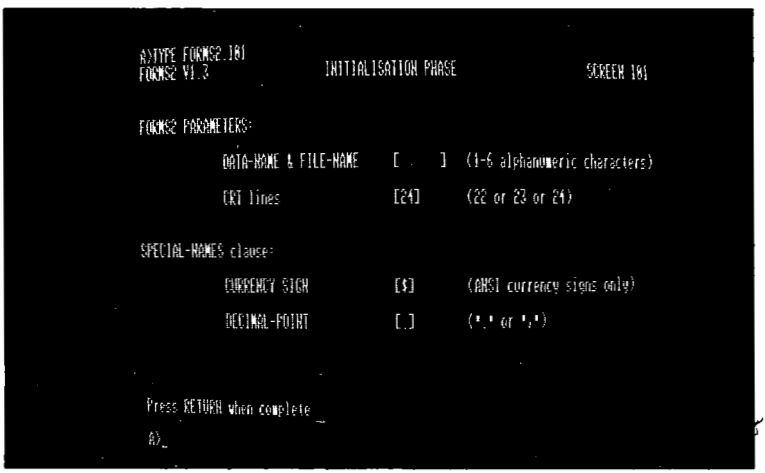
FORMS-2 VALIDATION

Before using your copy of FORMS-2, you should first validate the main files on your disk by performing the simple run below:

1. Boot the system up, and load FORMS-2 by entering the command:

   FORMS2<<

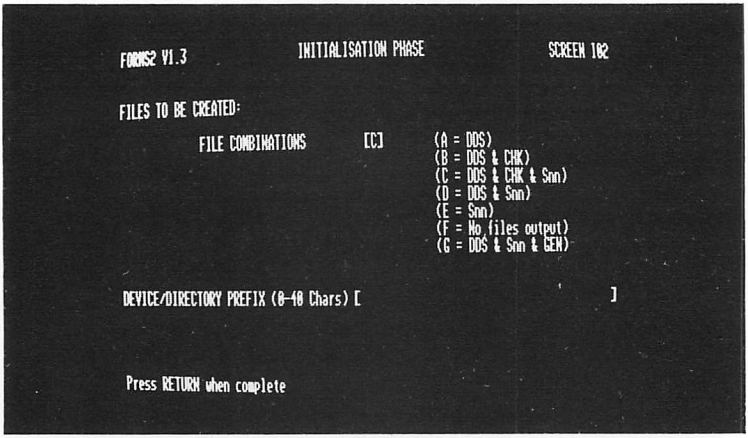
2. The program will run, and will come up with the first screen, thus:



Press RETURN when complete.

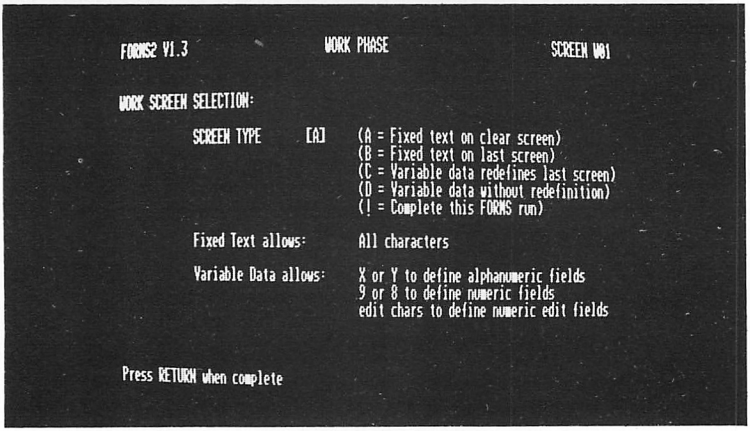
A six-character base for file-names and data-names is requested followed by three other questions. You need only key DEMO followed by the RETURN key to accept the defaults displayed.

3. FORMS-2 displays Screen I02 to request the output file option type and device/directory prefix.



Key F then RETURN.

4. FORMS-2 displays Screen W01 to request the Screen Type option, thus:



Note the default "A" and press the RETURN key.

5. FORMS-2 displays a blank screen. You are currently in Edit Mode, and should be able to position the cursor at any point on the screen. Use the cursor control keys and the normal character keys to set up the following text on the screen:

```
NAME      [                    ]
```

Finally press the RETURN key.

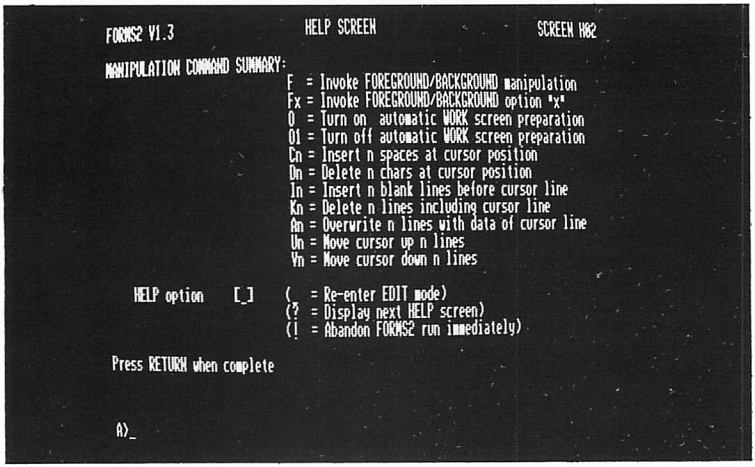
6. FORMS-2 puts "__" in the top left of the screen indicating that you are now in Command Mode. Enter ? and then press the RETURN key.

7. FORMS-2 displays screen H01, thus:

```
FORMS2 V1.3                    HELP SCREEN                    SCREEN H01

GENERAL COMMAND SUMMARY:
                        SPACE  = Process the work screen
                               = Re-enter EDIT mode
                            ?  = Display the next HELP screen
                            ?n = Display the nth HELP screen
                            @  = Re-enter WORK PHASE screen selection
                            !  = Terminate FORMS run immediately
                            &  = Position commands at EDIT mode cursor
                            ‡  = Indicate Index Form's data area start

   NOTE:   SPACE is the command to process the EDIT mode screen

   HELP option    [_]     (   = Re-enter EDIT mode)
                          (?  = Display next HELP screen)
                          (!  = Abandon FORMS2 run immediately)

Press RETURN when complete


A)_
```

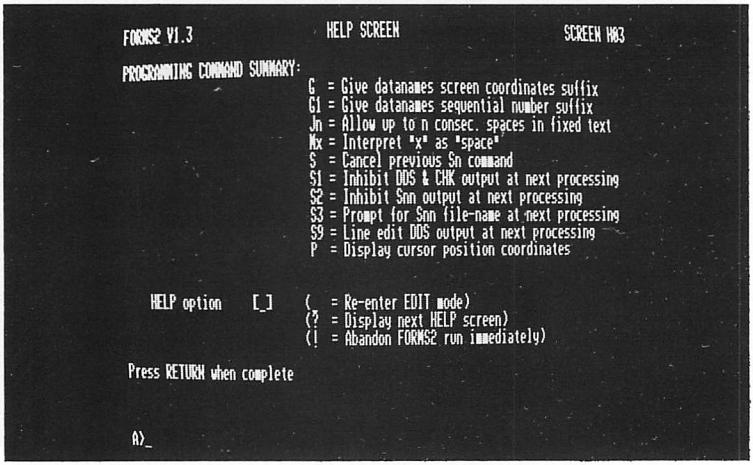
Enter ? then RETURN.

8. FORMS-2 displays screen H02, thus:



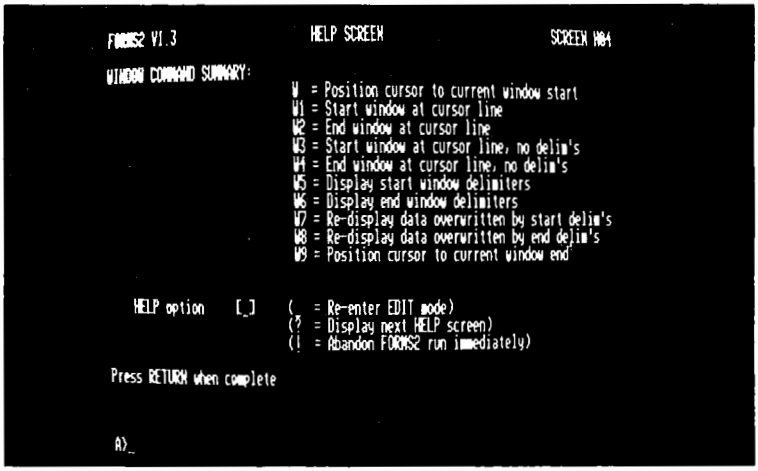
Enter ? then RETURN.

9. FORMS-2 displays screen H03, thus:



Enter ? then RETURN.

10. FORMS-2 displays screen H04, thus:

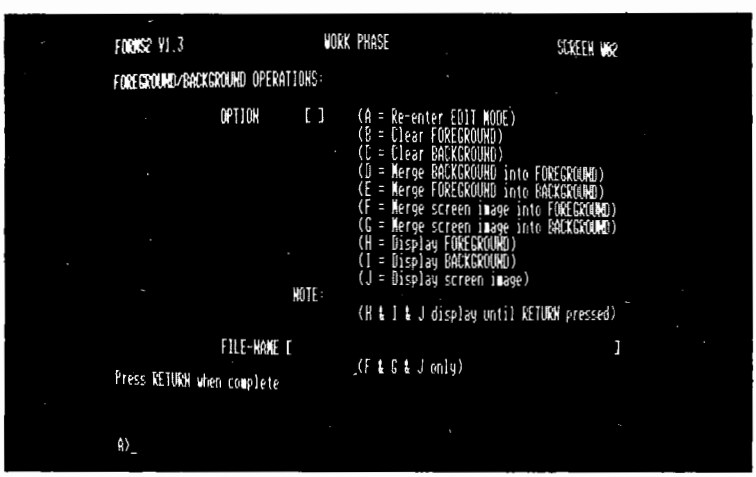


Simply press RETURN.

11. FORMS-2 redisplays the fixed text that you keyed in at step 5. Press RETURN.

12. FORMS-2 puts "__" in the top left of the screen. Enter F then press RETURN.

13. FORMS-2 displays screen W02, thus:



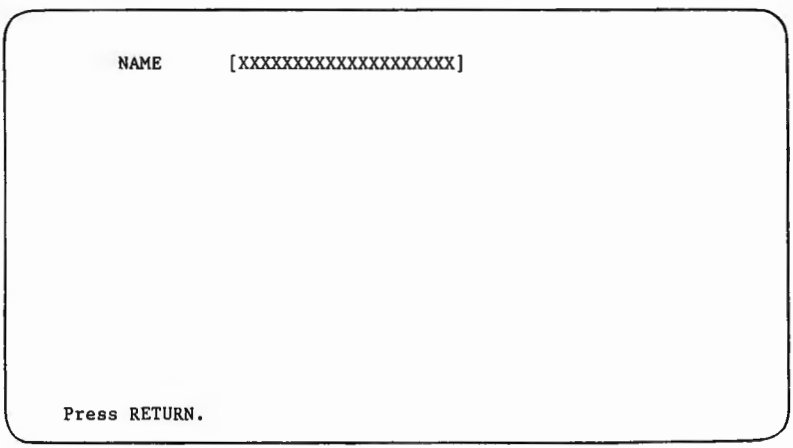
Enter A then RETURN.

14. Again FORMS-2 redisplays the fixed text entered at step 5. Press RETURN.

15. FORMS-2 puts "__" in the top left hand of the screen. Press the SPACE and then RETURN keys.

16. FORMS-2 displays screen W01 again to request the Screen Type option. Note the default is "C" and press RETURN.

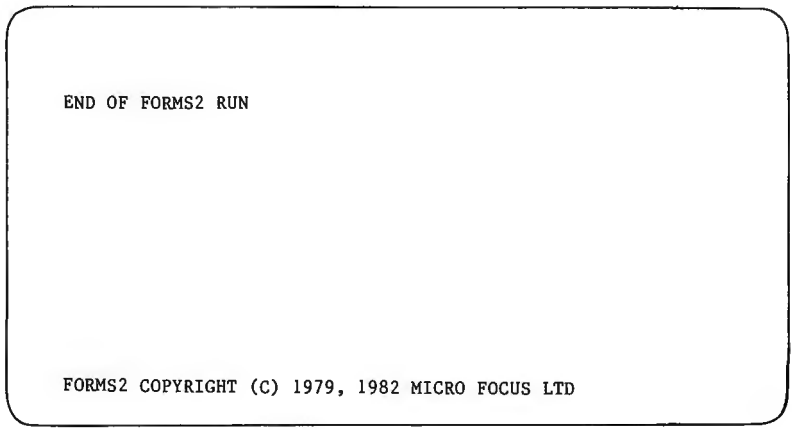
17. FORMS-2 displays the fixed text screen. Use the cursor control keys and key in X's alone to set up the screen as follows:



18. FORMS-2 puts "__" in the top left of the screen. Press the SPACE key and then RETURN.

19. FORMS-2 displays screen W01 again. This time enter ! and press RETURN to complete the run.

20. FORMS-2 terminates with the message:



You have now used all the FORMS-2 Screens and you can be sure that you have a usable product.

CHAPTER 2

INITIALISATION PHASE

The Initialisation Phase of the FORMS-2 Utility program immediately follows program load, and is only carried out once in any one run of FORMS-2. It consists of replying to questions asked on the two Initialisation screens I01 and I02.

INITIALISATION SCREEN I01

This screen is displayed immediately FORMS-2 is loaded.

Four items of information are requested. Note that the effect of pressing the RETURN key during this screen display immediately enters all responses so far made and any remaining defaults. Do not therefore press RETURN until all the required entries have been made, for the following:

* Data-name and the File-name Base
* Lines per CRT Screen
* Currency Sign
* Decimal-Point Representation

DATA-NAME AND FILE-NAME BASE

The record/filename base keyed in at this point is used in the following ways by FORMS-2:-

1. It is taken as the first part of all the data-names and record-names generated in this run. Uniqueness is achieved by adding a two-digit sequence number for new records and adding the sequential number of the field within the form for datanames within records. Optionally by means of a Work Phase command, uniqueness may be achieved by adding the screen coordinates.

2. It is taken as the main filename for files generated. These can consist of:-

filename .DDS	for COBOL Data Description Statements (See Chapter 4)
filename .CHK	for Checkout program (See Chapter 5)
filename .Snn	for Screen images (nn=ØØ, Ø1, Ø2,...,99) (See Chapter 6)
filename .GEN	for Index program (See Chapter 8)

Note that only one DDS file is output per FORMS-2 run, whereas a separate screen image file is output for each screen built.

LINES PER CRT SCREEN

FORMS-2 can be used with screens of 22, 23 or 24 lines. FORMS-2 defaults to 24 for this entry.

CURRENCY SIGN

This entry allows the default currency sign ($) to be overridden. It will cause generation of an appropriate Special-Names entry in either the Checkout and Index program. The specified currency sign should be used when specifying numeric edited fields in the Work Phase, and will be used in the generated data description statements.

NOTE:

> The specified character is not validated. Users should refer to the CIS COBOL or L/II COBOL Language Reference Manuals for a list of valid characters.

DECIMAL-POINT

This option allows the roles of the period or full-stop sign (.) and the comma sign (,) to be exchanged. If "," is specified then a DECIMAL-POINT IS COMMA clause will be generated in the Checkout or Index Programs. The default is ".". The specified decimal point sign should be used when specifying numeric edited fields in the Work Phase, and will be used in the generated data description statements.

INITIALISATION SCREEN I02

Screen I02 is displayed immediately after screen I01 entries are terminated by pressing the RETURN key.

At this point the user specifies the following:

* Types of files to be created
* Device/Directory to which files are to be written.

Once screen I02 is released by pressing RETURN the Work Phase is entered and it is no longer possible to amend information specified during the Initialisation Phase.

FILE COMBINATIONS

FORMS-2 offers options for all valid combinations of these files, each identified by a unique file name extension as follows:

1. DDS - The user may generate COBOL source Data Description Statements (DDS) corresponding to the screens he has created. These are output to a standard ASCII text file and may be subsequently compiled into any program using the standard COBOL COPY facility. In particular they are used by the Check-Out and Index programs, (see below).

   The reader who is unfamiliar with screen handling in a CIS or L/II COBOL program should consult the appropriate Language Reference Manual (especially the sections on ACCEPT/DISPLAY, FILLER, REDEFINES).

2. CHK - In addition to generating DDS, FORMS-2 can also generate a Checkout program. This consists of simply the Procedure Division statements (ACCEPT and DISPLAY) which correspond to the screens that have been created. These statements are contained in filename.CHK (see record/filename prefix) and they are combined with the following COPY files:-

   filename.DDS, FORMS2.CH1, FORMS2.CH2

   The Checkout program allows the user to demonstrate on the screen exactly how the system will operate, by displaying successively the screens he has just created, and by allowing data to be entered just as it would be under actual operating conditions.

3. Snn - The user may also output the text of the screen just designed to a file on disk in the form of a screen image. This file can be retrieved later in this run or in subsequent FORMS-2 runs, for further amendment if required.

   Alternatively they may be printed, and the hard copy simply used as a means of communicating between different individuals at different times (e.g. the end user and the programmer).

4. GEN - FORMS-2 can generate an Index program. This includes all code necessary to set up and maintain an indexed sequential file with records corresponding to the structure of the user's form. The code is output to filename.GEN and is combined with the following copy files:-

    filename.DDS, FORMS2.GN1, FORMS2.GN2

    Index program generation places constraints upon the user during the FORMS-2 run. The creation and operation of the Index program is discussed in detail in Chapter 8.

NOTE:

If the Q command (see Chapter 3) is entered at this point, FORMS-2 will "quit" back to screen I01, allowing amendment of information given there. This can be useful if RETURN is inadvertently pressed before all options have been entered.

DEVICE/DIRECTORY PREFIX

All the files output by FORMS-2 are on the same device/directory. Specify the mnemonic device/directory prefix as required by your operating system. If this reply field is left blank, the files will be created on the default device/directory.

CHAPTER 3

WORK PHASE

The user defines the screen layouts (forms) to be used in a CIS or L/II COBOL application by entering text at the keyboard to produce model forms on the screen. The user may define as many forms as he wishes in a single FORMS-2 run. To define one form requires at least two distinct Work Phases: one to define the fixed text of the form, and another to define the variable data entry fields.

Most commonly the first Work Phase is used to specify the fixed text form and the subsequent Work Phase to specify the variable data fields within the form. However this need not always be the case and FORMS-2 needs to know which type of text is to be input in a particular phase. Therefore the Work Phase is introduced by a screen presenting the various options (W01).

SCREEN W01

SCREEN TYPE SELECTION

Fixed Text selections offered at this screen are as follows:

A - The CRT screen is cleared to spaces in preparation for the user to enter the fixed text for a new form.

B - The previous screen is redisplayed to assist the user in defining additional fixed text. Text from the previous screen is used only as a background in this case, and is not included in the record definition for the fixed text currently being keyed in. The user must therefore ensure that if any part of the previous screen is inadvertently overkeyed, the original characters must not be replaced but cleared to spaces.

Variable Data selections offered at this screen are as follows:

C - The previous screen is redisplayed to assist the user in the redefinition of the form to incorporate variable data field specifications. In the application the data is keyed into the fixed text form itself.

D - The previous screen is redisplayed to assist the user in the definition of variable data fields which will be kept separate from the fixed text within the applications program. This may sometimes be of assistance to the programmer even though it results in larger application programs.

TERMINATING THE RUN

Screen W01 is redisplayed after completion of each Work Phase, and is the screen used to terminate the program. This is done by entering the character ! and pressing the RETURN key.

WARNING!

> Use of the ! command at any other time causes immediate abandonment of the run.

On termination, the DDS file is closed and an identification message is displayed. If the Check-out facility was specified during initialisation then output of the Check-out program to disk is completed and the CHK file is closed with an identification message displayed.

Termination occurs automatically after the second Work Phase if an Index program is being generated (see Chapter 8).

WORK SCREEN

After the screen type has been selected, the user is presented with the appropriate Work Screen for that text to be entered, i.e., if option A (fixed text on clear screen) is selected, a blank screen is displayed. For the other options, the previous screen is redisplayed to allow correct alignment of the current input.

BACKGROUND/FOREGROUND

In order to process only the data entered in this phase, FORMS-2 must keep this data separate from previously entered data which is displayed purely for alignment purposes. FORMS-2 does this by constructing the displayed Work Screen from two separate data areas, termed Background and Foreground. The Foreground holds the data entered during the current Work Phase. The Background holds previously entered data which has been retained for alignment of the data entered in the current Work Phase. At the end of each Work Phase FORMS-2 processes the Foreground data only.

When screen W01 is next reached, if options B, C or D are chosen, the Foreground is overlayed on the current Background contents and then Foreground is cleared to spaces. If option A is selected both Background and Foreground are cleared to spaces.

In this way the new Work Screen is prepared automatically.

NOTE:

It is possible to override this automatic Work Screen preparation for the next phase by means of a Work Screen command, described later, and leave both areas unchanged.

Generally data is entered into the Foreground via the keyboard, and is moved into the Background only from the Foreground. The F Work Screen command described later provide facilities for further manipulation of these areas. In particular it is possible to input a Screen Image file from a previous run into the Foreground, thus enabling amendment of existing forms.

During entry of data into the Work Screen (i.e. Foreground) two modes can be invoked as follows:

* Edit Mode - The mode in which the user keys data to create the model form. The initial mode is always Edit Mode.

* Command Mode - Commands are available to assist in the creation of the edited Work Screen and in its processing.

EDIT MODE

Edit Mode is identified to the user as the mode in which the cursor can be freely moved to any part of the screen by use of the cursor control keys. Entries may also be made into any part of the screen, in accordance with the screen type selected at the start of this Work Phase.

Fixed Text

In the design of the Fixed Text of a form (i.e. the fixed fields analogous to the pre-printed text on a paper form) any legible characters can be entered anywhere on the screen. This text will be displayed as "prompt" text during a data entry run of the application.

Variable Data

In the design of the Variable Data fields of a form (i.e. the fields analogous to the entry spaces on a pre-printed form) the characters X, Y and 8, 9 can be entered.

When variable data is being keyed in, X denotes an alphanumeric character and 9 denotes a numeric character. If it is required to have two alphanumeric fields contiguous with each other, Y's are placed in the character positions of the second field. Similarly, for contiguous numeric fields 8's are used.

EXAMPLE:

```
INVOICE NO ......
```

Suppose in an application the operator must key in an Invoice Number. Then the fixed text in this example could be "INVOICE NO ....." One example value of an invoice number could be "CA3021". It is necessary to define the area and type of this variable data explicitly for the computer. Hence if the invoice number always had two alphanumerics followed by four numerics, the user of FORMS-2 would key in XX9999 at the point on the screen (the dots in this example) where he wishes the operator to key the actual invoice number when the application itself is running. (Note that CIS and L/II COBOL provide an automatic validation of numeric fields).

Additionally, special editing characters can be input to specify numeric edited fields. Note that these fields should be separated by spaces. Numeric Edited Fields are described in the CIS or L/II COBOL Language Reference Manual. The valid characters are:-

Z, *, +, -, CR, DB, .(period), ,(comma), B, /, 0(zero), $

The $ sign is the currency sign which may be replaced by another sign as specified in the SPECIAL-NAMES clause of the COBOL program either directly or as specified during the Initialisation Phase of the FORMS-2 run (See Chapter 2).

NOTE: The picture characters S, V, P are not allowed.

COMMAND MODE

To switch to Command Mode from Edit Mode, the user simply presses the RETURN key.

Command Mode is identified to the user as the mode in which two underline characters initially bound the cursor, and the cursor is constrained to stay within these two characters.

Commands are invoked by keying the command always followed by pressing the RETURN key.

When execution of a command is complete, all commands (except SPACE, ! and Q) return to Edit Mode.

The default command is the underline character (_) and this causes immediate re-entry to Edit Mode.

Commands

The commands available to the user during the Work Phase fall into three main groups. All commands are entered by typing the command character/s followed by pressing the RETURN key.

1. General Commands

   General commands perform such functions as releasing the Work Screen for processing.

2. Work Screen Manipulation Commands

   Work Screen Manipulation commands assist in the preparation and editing of the Work Screen. It is recommended that all users become very familiar with these commands.

3. Programming Commands

   Programming commands have been introduced mainly for the convenience of the COBOL programmer, and some of them will not be meaningful without an understanding of COBOL. They include commands to assist in producing efficient code, and to give more control over the files output.

NOTE:

Groups 1 and 2 are summarised within HELP screens 1 and 2 (H01, H02). Group 3 is summarised on HELP screens 3 and 4 (H03, H04). See Appendix C.

Work Screen Manipulation Commands

The commands F and O are preparation commands.

F - Invoke Foreground/Background Menu Screen (W02).

Screen W02 contains options to assist in setting up the Foreground component of the Work Screen. The concept of Foreground and Background is described earlier in this Chapter.

The options made available by the FORMS-2 W02 Screen are:-

A - Return to Edit Mode.

B - Clear Foreground to spaces.

C - Clear Background to spaces.

D - Overlay Background data on to Foreground.

E - Overlay Foreground data on to Background

F - Overlay a Screen Image file onto Foreground. This option allows forms defined in previous runs (and also earlier in this run) to be amended. If selected the user is prompted for the identity of the required file.

G - Overlay a Screen Image file onto Background.

H - Show Foreground. This displays just the Foreground component of the Work Screen for examination. The full Work Screen will be restored on return to Edit Mode.

I - Show Background. This displays just the Background component of the Work Screen for examination.

J - Show a Screen Image file. The user is prompted for the identity and the specified file is displayed, but without corrupting the current contents of either Foreground or Background. This enables a check to be made prior to using option D.

Note:

The options H, I and J cause a display which remains until RETURN is pressed.

Fx - Specifies Required Foreground/Background Option.

x is the option code as contained in screen WP02 above. The specified option is executed and control returned to Edit Mode without display of the WP02 screen.

01 - "Switch Off" Automatic Background/Foreground Preparation.

The Background/Foreground preparation sequence is described under BACKGROUND/FOREGROUND earlier in this Chapter. This command is used to prevent the current Foreground being merged into the Background or either area being cleared for the next phase.

The 01 command remains effective until the command 0 (described below) is entered.

0 or 00 - Reset Background/Foreground Preparation.

The Background/Foreground preparation sequence is reset to automatic (starting at the beginning of the next Work Phase).

Note:

The Q command has a similar effect (beginning at the next phase).

The commands C, D, I, K and A are editing commands and are controlled by the position of the cursor at the time Command Mode is entered (i.e. the current cursor position) and operate only on the Foreground data. Background data remains in the same position.

Cn - Insert Spaces

Inserts n (1-9) spaces prior to the character at the current cursor position. Only the current line is affected.

Dn - Delete Characters

Deletes n (1-9) characters including the character at the current cursor position. Only the current line is affected.

In - Insert Blank Lines

Inserts n (1-9) lines prior to the line containing the current cursor position, irrespective of the column. Only whole lines can be inserted.

Kn - Delete (Kill) Lines

Deletes n (1-9) lines including the line containing the current cursor position. Only whole lines can be deleted using this command.

An - Repeat Current Line

Repeats the line containing the current cursor position n (1-9) times.

Note:
This does not act as an insert. Any Foreground data in the next n lines will be overwritten.

Commands U and V are cursor positioning commands.

Horizontal cursor positioning can be achieved by means of the → and ← keys.

Vertical cursor positioning is partially achieved by means of the ↓ and ↑ keys, but the cursor is usually placed at column 1.

Vertical tabulation within the same column can be required when setting up a form. Two "tabulation" commands are provided.

Un - Move Cursor Up

Moves the cursor up n (1-9) lines from the current cursor position. Cursor position within the line is maintained.

Vn - Move Cursor Down

Moves the cursor down n (1-9) lines from the current cursor position. Cursor position within the line is maintained.

Programming Commands

G - The Dataname Structuring Command

This command is not available if option G (for Index Program) is specified at screen I02.

The default recordname format generated by FORMS-2 for inclusion in the user's COBOL source program for screen formatting is as follows:

bbbbbb-rr (01 level)

where: bbbbbb is the 1-6 character base specified at screen I01

rr is the record number, starting at 00 in the first Work Phase and increasing by one for each subsequent Work Phase.

NOTE:

If the window commands are used to define a window starting in other than line 1 then the default recordname generated will be:

bbbbbb-rr-ll

where ll is the line number.

The default elementary dataname structure generated by FORMS-2 for inclusion in the COBOL source program for screen formatting is as follows:

bbbbbb-rr-nnnn

where: nnnn is the sequence of this field within the screen, starting at 0001.

Alternatively:

The G (or G0) command causes nnnn within the dataname to be the screen coordinates of the start of the field. This can sometimes be of use to the programmer as a reference guide when using COBOL facilities to set cursor position.

The G1 command restores the default dataname generation to using sequential field numbers.

The commands J and M are Multiple Spaces and FILLER Commands

The COBOL interactive ACCEPT and DISPLAY verbs operate only on named fields; FILLER areas are left alone. The time taken to display a screen depends both on the size and also the number of constituent fields.

When processing fixed text screens, FORMS-2 by default generates FILLER wherever multiple spaces appear. On some forms this can result in many small fields separated by small FILLER fields. The problem may be alleviated by:

Jn - Reset Multiple Spaces

The Jn command resets the number of contiguous spaces FORMS-2 will allow within the VALUE clause of a named field. This is initially set to 1.

n may be 0 - 9

NOTE:

J or J0 will force FILLER's even for single spaces.

An alternative method of forcing spaces within named fields is by use of the underline which is designated for this purpose. Use of the underline character in a field results in an actual space in the corresponding position in the generated VALUE clause.

If it is required to change the designated character from underline to something else (presumably because there is a requirement to generate VALUE "_"), the command M is used as follows:

Mx - Change Default "space"

The Mx command changes the default character (underline) to that specified by x.

NOTE:
If the space character ( ) itself is specified this will force generation of named fields for the entire screen without any FILLER's.

Sn - File Output Control Command

This command is not available if Option G (for Index Program) is specified at screen I02.

S (or S0) cancels any other Sn commands in effect at the time.

S1 suppresses DDS (& CHK) text generation for this Work Screen. Generation of this text resumes for the next Work Screen unless the same command is repeated in the next phase.

S2 suppresses Screen Image (Snn) text generation for this Work Screen. Commonly used to suppress Screen Images of just variable data fields. Again the effect only lasts for the current phase.

S3 results in the user being given the opportunity to override the default Screen Image file identifier for the current Work Screen. Normally if a file already exists with the default identifier the user is given the option of overriding it. If he rejects this option he is prompted for an alternative file identifier. This command forces the alternative file identifier to be requested even when no file exists with the default identifier.

S9 causes FORMS-2 to halt after display of each line of code during DDS generation. FORMS-2 re-ACCEPTs the line before outputting it to the .DDS file. This provides the programmer with a limited editing capability, which may prove useful under special circumstances.
This facility is not available if option C or G (for generated programs) is specified at screen I02.

P - Cursor Coordinate Command

Causes the coordinates of the current cursor position to be displayed at the command area position. This display lasts a few seconds, after which the Work Screen contents at the command area are restored and control is returned automatically to EDIT MODE. Where sequential field numbers are being used within datanames, this command provides an easy alternative method of ascertaining the coordinates of any field.

Wn - Window Commands

These commands are not available if option G (for Index Program) is specified at screen I02.

The "window" defines the area (full lines) to be processed by FORMS-2 when generating DDS text. By default the window is the full screen. Where window start or end is other than start or end of screen, a delimiting line of hyphens may optionally be displayed on the line just outside the window. e.g. if window starts in line 4, delimiters appear along the length of line 3.

The principal use of the window is to allow a form to be created in situ, but save memory by avoiding the description of blank lines at the top of the screen.

NOTE:

Where window is used in this way the generated record name incorporates the start line number of the window which can then act as a guide to the programmer, using the CIS or L/II COBOL ACCEPT/DISPLAY AT coordinates facility.

The detailed commands give the user very comprehensive window formatting capability as follows:

W (or W0) positions the cursor at current window start. This is the equivalent of the HOM (↖) key when the window facility is in use.

W1 sets start of window to current line with delimiters on previous line

W2 sets end of window at end of current line with delimiters on next line.

W3 sets start of window to current line without delimiters.

W4 sets end of window at end of current line without delimiters.

W5 displays delimiters preceding current window start.

W6 displays delimiters following current window end.

W7 erases start delimiters and restores any Work Screen data to display.

W8 erases end delimiters and restores any Work Screen data to display.

W9 positions the cursor at current window end.

NOTES:

1. Delimiters do not corrupt Background/Foreground contents.

2. The output Screen Image will include the full Foreground part of the Work Screen without delimiters, irrespective of whether a window has been defined. This could be used to include annotation on the Screen Image which does not affect DDS generation.

3. A useful purpose for the window facility would be if a form is required to be displayed in two stages: the first 10 lines then the second 10 lines. This could be created as a single Screen Image including both sections of the form, and the programmer could 'window in' on the relevant portions as required when the DDS text is generated.

WORK PHASE COMPLETION

To complete the Work Phase of FORMS-2, the user selects Command Mode, keys the SPACE character and then the RETURN character. The SPACE character is the command to release the Work Screen for processing.

FORMS-2 completes the Work Phase (depending on the file selection at screen I02) as follows:

1. If this is a variable data field definition Work Phase (Option C or D at screen W01), validation then occurs with the message:

   WORK SCREEN VALIDATION in progress - DO NOT press RETURN

2. If DDS file generation is selected at screen I02, the source code produced is echoed to the screen as it is written to disk. If the S9 command was specified processing stops after each line of code to enable changes to be made as required. This is recommended only if special requirements dictate its use.

3. If a screen image file was requested at screen I02, the screen image is echoed to the screen as it is written to the disk file. The identity of the created file is displayed and the user must press RETURN to continue.

4. Screen W01 is redisplayed so that the run can be terminated or continued.

NOTES:

1. During validation of variable data only those characters listed in the description of text-types are permitted (plus space). If any other character is encountered, an error is notified by the validation routine by alternately displaying "?" and the offending character to give a flashing effect. This error indication then ceases and FORMS-2 returns to Edit Mode with the cursor positioned under the erroneous character. The user must reissue the SPACE command after making any corrections.

2. FORMS-2 will allow editing characters but will not verify that the combinations of these are valid; COBOL editing rules must therefore be obeyed to ensure error free code. Note that these fields should be separated by spaces.

3. Only Foreground data is output to the Screen Image file.

CHAPTER 4

DATA DESCRIPTIONS

This Chapter describes the CIS or L/II COBOL Data Descriptions that FORMS-2 can generate in the file filename.DDS, and a knowledge of COBOL is a prerequisite for reading it.

The CIS and L/II COBOL extensions to the ACCEPT and DISPLAY verbs allow comprehensive screen handling to be included in a user application. (See CIS or L/II COBOL Language Reference Manual). Programming the necessary data description statements can be tedious and expensive in terms of programmer time, particularly since it is very prone to simple errors.

FORMS-2 simplifies the production of error-free data descriptions by allowing screen layouts (forms) to be specified in the most convenient way, namely by setting them up in situ on the screen as described in Chapters 2 and 3. If the facility is invoked by selection of an appropriate option at screen I02 during the Initialisation Phase, FORMS-2 automatically converts this input to the necessary COBOL statements and outputs these to a data description (DDS) file. The user merely incorporates this code in his source application code by means of the CIS or L/II COBOL COPY verb and uses record-names and data-names consistent with those generated by FORMS-2.

RECORD-NAME AND DATA-NAME GENERATION

At Initialisation screen I01 a base name is requested from the user. This is a 6-character field into which the user enters a chosen name consistent with COBOL data naming (See CIS or L/II COBOL Language Reference Manual). This base is then used to generate the COBOL data-names.

RECORD NAMING

The default record-name format generated by FORMS-2 for inclusion in the user's COBOL source program for screen formatting is as follows:

bbbbbb-rr (01 level)

where: bbbbbb is the 1-6 character base specified at screen IN01

rr is the record number, starting at 00 in the first Work Phase and increasing by one for each subsequent Work Phase.

NOTE:

If the window commands are used to define a window starting in other than line 1 then the record-name generated will be:

bbbbbb-rr-ll

where ll is the line number. This serves as a useful reminder to the programmer when coding the appropriate ACCEPT/DISPLAY statements.

DATA NAMING

The elementary data-name structure generated by FORMS-2 for inclusion in the CIS or L/II COBOL source program for screen formatting is as follows:

bbbbbb-rr-nnnn

where: nnnn is the sequence of this field within the screen, starting at 0001.

Sometimes it may be more convenient to the programmer to have the screen coordinates incorporated in the data-name rather than a field sequence number. This can be achieved by use of the G command during the Work Phase.

PICTURE GENERATION

Generation of PICTURE clauses by FORMS-2 depends on the type of text selected at screen W01 at the start of each Work Phase. Note that FORMS-2 will force field boundaries at the end of each line in order to be compatible with certain types of CRT.

FIXED TEXT

At the end of a fixed text Work Phase FORMS-2 generates only FILLER areas or named alphanumeric fields with associated VALUE clauses.

The CIS or L/II COBOL interactive ACCEPT and DISPLAY verbs operate only on named fields; FILLER areas are left alone. The time taken to display a screen depends both on the size and also the number of constituent fields.

When processing fixed text screens, FORMS-2 by default generates FILLER wherever multiple spaces appear. This default can be altered by means of the J command described in Chapter 3. Alternatively _ (underline) can be used to force inclusion of spaces within a VALUE clause. The default character used for this purpose can be changed by means of the M command described in Chapter 3.

VARIABLE DATA FIELDS

At the end of a variable data Work Phase FORMS-2 generates alphanumeric, numeric or numeric edited fields depending on the actual characters keyed by the user (see Chapter 3). These are usually normal CIS or L/II COBOL picture characters 9 and X but note the additional use of 8 and Y as alternatives to 9 and X; and also the exclusion of S, V and P as described under Variable Data in Chapter 3.

EDITING THE DDS FILE

Normally the DDS output from FORMS-2 should be all that is required. Where special circumstances dictate the use of particular datanames or the disallowed picture characters, the S9 command (See Chapter 3) will allow DDS lines to be edited prior to output. Alternatively a conventional text editor can be used to edit the file. However note that this editing process must be repeated if ever the form is amended by means of FORMS-2.

It is also possible to suppress completely the DDS output for a particular Work Phase by means of the S1 command. Note that if this is used the record number incorporated in data-names will still be stepped up by 1 for the next Work Phase.

INCORPORATION OF DDS FILE CONTENTS

All that the user has to do to incorporate the generated data descriptions into the application program is to copy in the DDS file using the COPY statement available in CIS or L/II COBOL. This is described in the CIS or L/II COBOL Language Reference Manual.

The COPY statement to incorporate the DEMO1 sample forms designed in Chapter 7 would be:

```
000000  COPY  "DEMO1.DDS".
```

and would be coded within the Data Division.

This statement is included in all Check-Out or Index programs generated, and any of these can be referred to for an example.

CHAPTER 5

THE CHECK-OUT PROGRAM

This Chapter describes the Check-Out program that FORMS-2 can generate automatically while generating the created forms. The Check-Out program enables the user to:

* Validate the DDS file
* Demonstrate the operation of the proposed application
* Check the use of the new forms for data entry
* Check the use of the new forms for data amendment

The Check-Out source code which is in COBOL includes a COPY statement for the DDS file exactly as it would be coded in the user's application and is therefore a true validation of the DDS file when compiled.

The Check-Out program logic is a sequence of DISPLAY or ACCEPT statements for the screens defined in the FORMS-2 run, in the order in which they were created. Therefore by entering all required forms in a single FORMS-2 run, a demonstration program using all the forms can be simply and rapidly created, with no programming necessary. For a complex application the best method might be to create each form in isolation, using screen image output only. FORMS-2 can then be run again to produce the required Check-Out program, using the facility to re-input screen images (the FF command or the F command and the F option in the subsequent screen display). Use of this facility would also enable a complex sequence of screens to be set up for demonstration purposes incorporating the same screen more than once.

After passing through the sequence of screens, Check-Out gives the option of repeating the whole sequence. On the second pass previously entered data is redisplayed, allowing the user to check the use of his forms for both initial data entry and data amendment.

CHECK-OUT PROGRAM GENERATION

The facility is invoked by selection of an appropriate option at screen I02 during the Initialisation Phase. Note that the default option results in generation of the Check-Out Program.

The source code of the program is written to a file named:

basename.CHK

where: basename is the name entered by the user at screen I01 during the Initialisation Phase.

CHECK-OUT PROGRAM COMPILATION

The program is then compiled from the CHK file. The following files must be present during compilation:

basename.DDS - the DDS file produced in the FORMS2 run. This must be on the device/directory selected at Screen I02, or the default drive if none was selected.

FORMS2.CH1 }
FORMS2.CH2 } - the skeleton for the Check-Out program on the default device/directory

The Check-Out program is then compiled in the usual way by entering the standard CIS or L/II COBOL compile command for your Operating System with the source file:

basename.CHK

See Appendix D

Details of compilation using the CIS or L/II COBOL Compiler are given in the appropriate Operating Guide.

CHECK-OUT PROGRAM RUNNING

LOADING

The program can be loaded immediately after compilation by use of the standard O/S run command and the name of the intermediate code file:

```
basename.INT
```

However, to be able to load directly in subsequent use the = parameter of the run command should be used and the SAVE file renamed. (See Appendix D for Operating System specific commands). Thereafter the direct load command can be used.

CHECK-OUT PROCESSING

The basic function of the Check-Out program is to display the fixed text fields of the form and enable data to be entered into the variable data fields of the form in the sequence in which the screens were created.

However the detailed logic is slightly more sophisticated. The following notes make references to the options taken for screen type at Screen W01, and these are discussed in Chapter 3.

Fixed Text Screens

The fixed text of a form is displayed. If there are two consecutive fixed text forms, Check-Out pauses after the first display until the user presses RETURN.

Fixed text on clear screen

If option A was taken at creation of the form, Check-Out clears the CRT before displaying the screen.

Fixed text on last screen

If option B was taken for the creation of the screen, any text displayed remains on the CRT except where it is overwritten by the text of the new screen.

Variable Data Screens

An ACCEPT statement is issued for a variable data screen, allowing the user to enter data in the unprotected areas, (i.e. the fields specified by means of X's and 9's etc.).

Users can check the extents of the fields. For numeric fields they can also check that only numeric characters may be entered, and the effect of entering the left zero fill character ".". (Use of the "." character is described in the CIS or L/II COBOL Language Reference manuals under The ACCEPT Statement).

On other than the first pass through the sequence of screens the previously entered data is redisplayed before the ACCEPT is issued.

If the variable data screen includes numeric edited fields, the ACCEPT for the screen is followed by a corresponding DISPLAY to show the effect of the editing or normalisation performed by the CIS or L/II COBOL run time systems. Note that the normalised fields are not automatically echoed to the CRT.

CHECK-OUT COMPLETION

After the entire sequence of screens has been passed, the Check-Out program displays:

```
CHECK-OUT completed
Repeat? [N] (Y=Yes)
```

If it is required to repeat the sequence of screens, Key Y and press RETURN.

Otherwise simply press RETURN to take the default to terminate the program.

CHAPTER 6

SCREEN IMAGE FILE

This Chapter describes the Screen Image file that FORMS-2 can generate in addition to (or instead of) the COBOL data description statements described in Chapter 4. These files contain exact text images of the user described forms. These form images can be:

* Used to provide the basis for amendments to the form.
* Printed to yield printed copies of the form.
* Used as a means of communication between the system designer and the applications programmer.

SCREEN IMAGE FILE GENERATION

The facility is invoked by selection of an appropriate option at Screen I02 during the Initialisation Phase. Note that the default option will cause screen image output.

Screen images are output to files named:

basename.Snn

where: basename is the name entered by the user at screen I01 within the Initialisation Phase. nn is a number 00 - 99

The default filename can be overridden by issuing the S3 command during the Work Phase. This causes FORMS-2 to request input of the required filename during processing of this Work Screen.

A separate file is created at the end of each Work Phase, the numeric part of the name (nn) incremented by 1 each time. A screen image file is structured as a standard line sequential file with a record for each line of the screen. Each screen image contains only text entered during the Work Phase in which it is generated (i.e. Foreground data - see Chapter 3). Thus for a variable data Work Phase the output screen image contains only X's, 9's, Y's and 8's.

It is possible to suppress the screen image output from any Work Phase by issuing the S2 command during that phase. If this command is used the numeric part of the filename extension will still be updated for the next phase to keep in line with the record numbering within the generated data descriptions (DDS).

FORMS-2 MAINTENANCE

The COBOL Data Description statements have been generated from a user-designated form by FORMS-2 in a DDS file. There is likely to be a continuing requirement to make corrections and adjustments to maintain the form. The DDS file can be maintained using a conventional text editor, but this involves the high risk of simple but expensive errors which FORMS-2 eliminates. Therefore the user's form is output to a screen image file as an exact image, and FORMS-2 provides the facility to read screen images back from disc to allow for further amendment. This is achieved by running FORMS-2 and issuing the FF command once the first Work Screen is reached (see the Fx Work Screen Manipulation command in Chapter 3). The user is then prompted for the identity of the screen image required. FORMS-2 reads the screen image file into the Foreground area of the Work Screen and then returns to Edit Mode. The form is then displayed as if it had just been keyed and any required amendments can be made before releasing the screen for processing by means of the SPACE command.

NOTE:

> When FORMS-2 is used for maintenance in this way it will overwrite the existing files, but only after issuing warnings that the files already exist, and receiving confirmation to proceed. For screen image files FORMS-2 offers the facility of specifying an alternative file identity if the user wishes to retain the old version.

PRINTED FORMS

The screen image files are created as line sequential files in accordance with the conventions of the operating system. Standard software can therefore be used to print them, and the resultant hard copy will be an exact image of the user's form with no risk of transcription error.

FORM IMAGES IN THE DESIGN PROCESS

Form images can be used as a step within the applications design process, providing a valuable part of the designer/programmer interface.

For interactive applications, design of the user interface (i.e. the screen layouts or forms) may take place well in advance of the actual program being written, and the forms designer need not have any detailed knowledge of COBOL.

FORMS-2 enables a non-technical user to generate valid COBOL statements. An experienced COBOL programmer can make use of commands available to generate the most efficient code (e.g. by influencing the number of fields to be displayed).

Thus it may sometimes be advantageous to use screen image output alone as an intermediate stage in the design process, with the programmer using the image files as input to FORMS-2 to produce the final DDS file. If FORMS-2 is used in this way, both fixed text and variable areas could conveniently be indicated on a single fixed text screen. The programmer can easily then use this screen to generate the DDS file, and the form designer does not need to know any details of COBOL data field specifications.

CHAPTER 7

FORMS-2 USER SCREEN
GENERATION EXAMPLE

It is required to build the data entry form:

```
NAME      [                    ]
ADDRESS   [                    ]
          [                    ]
          [                    ]
TEL       [         ]
```

where NAME and ADDRESS are alphanumeric fields and TEL is a numeric field. At data entry time after insertion of the name, address and telephone number: P. Smith, 8 George Street, Plymouth, Devon, 88326, the form is required to appear as:

```
NAME      [SMITH. P            ]
ADDRESS   [8 George Street,    ]
          [Plymouth            ]
          [Devon               ]
TEL       [88326    ]
```

It is assumed the system is booted, the issued files have been copied to the CIS or L/II COBOL O/S system disk so that the CHK file contents can be compiled (Step 13), and that FORMS-2 has been configured to your CRT.

The following steps must then be carried out:

1. The operator loads FORMS-2 by entering the load command for your O/S.

   Note that the program name for load purposes is FORMS2.

   See Appendix D for specific Operating System format for this command.

2. FORMS-2 displays Screen I01 requesting a six-character base for file-names and data-names followed by three other questions. If the CRT is standard (24 lines) no further questions need be answered for this screen. Key DEM01 followed by the RETURN key if the default screen size (24) is correct.

3. FORMS-2 displays Screen I02 to request the output file option type and drive number. Key RETURN to accept the default values.

4. FORMS-2 displays Screen W01 to request the Screen Type option. Note the default "A" and press the RETURN key.

5. FORMS-2 displays a blank screen. Use the cursor control keys and the normal character keys to set up the following text on the screen:

```
NAME      [                    ]
ADDRESS   [                    ]
          [                    ]
          [                    ]
TEL       [            ]
```

   Press the RETURN key.

6. FORMS-2 puts "__" in the top left of the screen. Press the SPACE and RETURN keys

7. FORMS-2 processes the screen to create a fixed text form. This takes a short period and involves the following displays on the CRT:

   DDS source code as generated, followed by a redisplay of the fixed text as it is written to the Screen Image file.

   A message is then displayed giving the name of the fixed text Screen Image file created. Press RETURN as requested.

8. FORMS-2 displays Screen W01 to request the Screen Type option. Note the default is "C" and press the RETURN key.

9. FORMS-2 displays the fixed text screen as background data. The operator then uses the cursor control keys and keys in X's and 9's alone to set up the screen as follows:

```
NAME      [XXXXXXXXXXXXXXXXXXXX]
ADDRESS   [XXXXXXXXXXXXXXXXXXXX]
          [XXXXXXXXXXXXXXXXXXXX]
          [XXXXXXXXXXXXXXXXXXXX]
TEL       [9999999]
```

   Press the RETURN key

10. FORMS-2 displays "__" in the top left hand of the screen; press the SPACE and RETURN keys. There is a short pause while FORMS-2 validates the screen content, during which the following message is displayed:

    WORK SCREEN VALIDATION in progress - DO NOT press RETURN

11. FORMS-2 processes the X's and 9's to create a variable data form, with the following displays to the CRT as it goes:

DDS source code as generated, followed by a redisplay of the variable text as it is written to the Screen Image file.

A message is then displayed giving the name of the variable data screen image file created. Press RETURN as requested.

12. FORMS-2 displays screen W01 again. Key ! followed by RETURN to terminate the run. FORMS-2 displays the names of the DDS and CHK files created and displays an END OF RUN message.

13. Compile the check-out program by typing the standard CIS or L/II COBOL compilation command for your Operating System with the directive COPYLIST and file name DEM01.CHK (See Appendix D).

14. When the compilation finishes, the two screens can be checked out by using the standard run command for your Operating System to load the intermediate code from file:

    DEM01.INT

    See Appendix D

15. The Demonstration program will then run. The fixed data form is displayed on the screen. The variable data form is used to accept data.

    Satisfy yourself that the cursor can only be placed in the variable fields, and that the data keyable into the fields depends on whether X or 9 was specified. The effect of left fill character "." may also be tested.

    When satisfied, press RETURN to complete. A message is displayed as follows:

    ```
    CHECK-OUT completed
    Repeat? [N] (Y=Yes)
    ```

    Press RETURN to accept the No default and complete.

16. The Check-Out program displays:

    END OF FORMS2 CHECK-OUT

NOTE:

The variable form is used in the demonstration for ACCEPTing data. In practice the form can be used for DISPLAYing data as well as ACCEPTing it. The demonstration shows the extent and type of each field which will be the same in DISPLAY as well as ACCEPT. A useful technique for clearing just the variable data fields on the screen is to move spaces to the ACCEPT record and then display it.

17. You can now examine the disk files:

```
DEMO1.DDS
DEMO1.CHK
DEMO1.SØØ
DEMO1.SØ1
DEMO1.INT
DEMO1.LST
```

to check the output from FORMS-2 during this use.

18. You have now learnt how to use FORMS-2 to create screens of fixed and variable data automatically for inclusion in your CIS or L/II COBOL program.

If you continue with steps 19 onwards you will learn to update both the fixed and variable data screens already created by moving them from background into foreground.

19. Reload FORMS-2 by typing the load command for your O/S. See Appendix D.

20. FORMS-2 displays Screen I01 requesting the six-character file- and data-name base as at step 2. Answer the questions as necessary at step 2 and press RETURN.

21. FORMS-2 displays Screen I02 requesting the output file option type and drive number; key RETURN to accept the default values.

22. FORMS-2 displays the message:

```
DEMO1.DDS already exists
overwrite? [N] (Y=Yes)
```

Key Y and press RETURN

NOTE:

If the No default is entered here, the run is abandoned.

23. FORMS-2 displays the message:

```
DEMO1.CHK already exists
overwrite? [N] (Y=Yes)
```

Key Y and press RETURN

NOTE:

If the No default is entered here, the run is abandoned.

24. FORMS-2 displays Screen W01 again. Press RETURN to accept the default option A.

25. FORMS-2 displays a blank screen in Edit Mode. Press RETURN to enter Command Mode, then F followed by RETURN to invoke the Foreground/Background selection screen. (We want to update our form so it must be in Foreground).

26. FORMS-2 displays the Foreground option screen. Enter option F then the filename DEM01.S00, then press RETURN.

27. FORMS-2 displays screen W02 again. Select option A to return to Edit Mode.

28. FORMS-2 displays the fixed text screen (previously created at step 5). Move the cursor to the word ADDRESS and overtype it with ABODE. Remember to overtype the extra characters SS with spaces, and then press RETURN.

29. Enter the SPACE command, then RETURN.

30. FORMS-2 displays the following message reminding you that your altered fixed text Screen Image is about to overwrite your previous Screen Image in the file:

    ```
    DEM01.S00 already exists
    overwrite?  [N] (Y=Yes)
    ```

    NOTE:

    If the No default was entered here, a file identity for a new Screen Image would be requested.

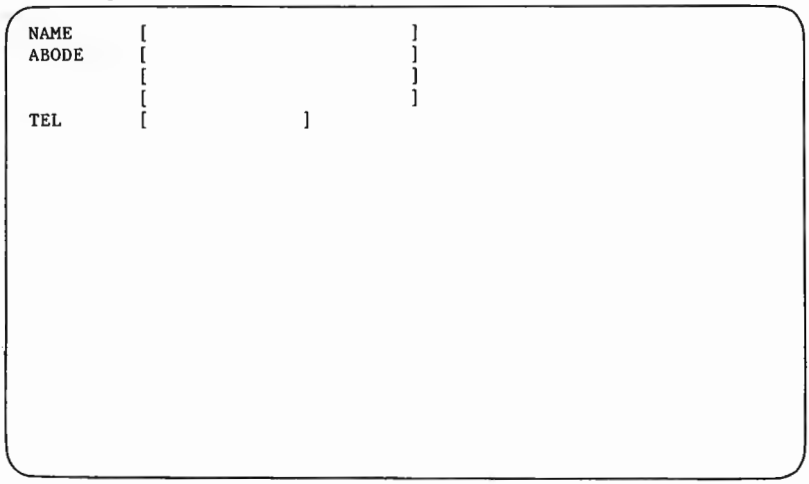
31. FORMS-2 displays the screen image and then displays the file name as follows:

    ```
    File created = DEM01.S00
    ```

    Press RETURN to continue.

32. FORMS-2 displays screen W01 with option C as default to enable specification of variable data fields. Enter RETURN to accept the default.

33. FORMS-2 displays the altered fixed text as follows to assist in defining the variable fields.



Press RETURN to enter Command Mode then F then RETURN.

34. FORMS-2 displays the Foreground/Background Operations screen again. Enter the option F then the file name DEM01.S01, then press RETURN to retrieve your variable text created at step 9.

35. FORMS-2 displays Screen W02 with option H as default. If you press RETURN to accept this default, FORMS-2 displays the current Foreground contents. Note that this is only the X's and 9's that define the variable data fields (the fixed text is in the Background area). Press RETURN to re-invoke Screen W02.

36. FORMS-2 displays Screen W02 with A as default. Press RETURN to accept this default.

37. FORMS-2 displays the whole form again. (We could now alter the variable text fields if required).

You have now seen facilities to retrieve fixed text and variable text from previously created files. Note that with a small number of variable data fields such as in this example it would, in practice, be easier to re-key them.

38. Press RETURN then SPACE then RETURN to process the altered form. Again there is a pause while FORMS-2 validates the variable fields.

39. FORMS-2 produces the DDS file then displays the message:

```
DEMO1.S01 already exists
overwrite?  [N] (Y=Yes)
```

40. A message is displayed as follows:

```
File created = DEMO1.S00
```

41. FORMS-2 displays screen W01 again. This time enter ! and press RETURN to complete the run.

42. Repeat steps 13 to 16 if you wish to run the Check-Out program again to verify the altered form.

CHAPTER 8

INDEX PROGRAM

FORMS-2 provides facilities for automatically generating a COBOL Index program to create and maintain an indexed sequential file. The input required to generate the Index program and use it to maintain files is supplied interactively by the Operator through the CRT.

The user designs a data entry screen using FORMS-2 by specifying the fields that will comprise the indexed sequential file records in the usual fixed text and variable text work phases described in Chapter 3.

The user interface to the generated Index program is simply the form designed by the user that reflects the desired record structure. Users need give no thought to setting up specific 'command' areas, but only to consider their data requirements.

It should be noted that the user must have access to the CIS or L/II COBOL software to compile the source Index program that FORMS-2 produces.

The generated Index program is written to the file filename.GEN and provides the following facilities required for the creation and maintenance of an indexed sequential file.

* Select records by key field for display (Enquiry by key field)
* Select records sequentially for display (Sequential Enquiry)
* Amend existing records
* Delete existing records
* Insert new records

The program has been developed so that it is not necessary for the user to explicitly state the facility to be invoked at any one time; the program is able to follow the logic from the way the actual data and cursor position are manipulated.

It can be seen that only the variable text data is written to the file and the fixed text data is merely a template to enable each field to be entered separately at data entry time.

A record in the indexed sequential file is constructed by concatenating the variable fields of the form, in the order in which they appear. The record must include a key area by which it can be uniquely accessed. The Index program logic requires that this key area must be at the beginning of the record i.e, must be the first integral field/s in the form, and must not exceed 32 characters in length.

This key area constitutes part of the record data. For convenience, the remaining fields are known as the data fields.

Chapter 9 shows the sample application used in Chapter 4 adapted to create and maintain a file of names, addresses and telephone numbers.

INDEX PROGRAM GENERATION

An Index program is generated using FORMS-2 as described in Chapters 2 and 3.

All existing FORMS-2 facilities are present, but logic is incorporated to prevent the use of inappropriate features if the Index Program option is taken. The steps involved are:-

1. Initialisation

   a. Screen I01
      Specify name-base etc. as normal

   b. Screen I02
      Specify option G for Index program generation.

2. Work Phase One

   a. Screen W01

      Work Screen Selection - The program forces the default option 'A' for fixed text entry by refusing to accept anything else (except ! to abandon the run or ? to display Help screens).

   b. Fixed Text Work Screen

      A blank work screen is then displayed for input of the fixed text form.

      FORMS-2 commands as described in Chapter 3 are available except:-

      G - The generated program relies on the default dataname structure. This command is rejected.

      S - It would be inappropriate to switch off DDS generation, and this command is rejected.

      W - This feature is not available to the user, and the command is rejected. However the program reserves the bottom line for potential use in the generated program for system messages ("RECORD NOT FOUND" etc.), and a delimiting line of hyphens marks this fact.

      The screen is released for processing by the sequence 'RETURN SPACE RETURN', when the fixed text screen has been completely entered.

      The Work Screen Selection screen is again displayed.

3. Work Phase Two

   a. Screen W01

   b. This time the program forces the default option 'C'.

      Variable fields are specified as described in Chapter 3 i.e., X/Y/8/9 and editing characters. At some time before releasing this screen it is necessary to define the end-of-key/start-of-data bound within the record. This is done by positioning the cursor on the first data field, entering Command Mode and keying the '*' command (i.e. the sequence 'RETURN * RETURN').

      NOTE:

      A key field cannot exceed 32 characters.

      The screen is released by the usual 'RETURN-SPACE-RETURN' sequence. If the program is not satisfied with the specification of the key/data boundary it will return to Edit Mode.

      Upon completion of the variable text screen FORMS-2 completes its processing and terminates automatically without any need for the termination (!) command. In fact the ! command is only used to abandon the run when generating the Index Program.

FILES GENERATED

The following files are written to disk by FORMS-2.

basename.S00 }	-	Screen image files
basename.S01 }		
basename.DDS	-	COBOL data description statement file
basename.GEN	-	Source file for the generated Index program.

INDEX PROGRAM COMPILATION

The Index program can now be compiled from the .GEN file in the usual way. The following files must be present during compilation:

basename.DDS - the DDS file produced in the run. This must be on the device/directory selected at screen I02, or the default if none was specified.

FORMS2.GN1
FORMS2.GN2 } - the skeleton for the generated program on the Default device/directory.

The Index program is then compiled in the usual way by entering the standard CIS and L/II COBOL compilation command for your Operating System (O/S) to load the Index program from file:

basename.GEN

See Appendix D

Details of compilation using the CIS and L/II COBOL compilers are given in the appropriate Operating Guides.

After compilation, the user can run the generated program.

INDEX PROGRAM RUNNING

LOADING

The program can be loaded immediately after compilation by using the standard run command for your Operating System to load the program from file:

basename.INT

However, to be able to load directly in subsequent use the = directive of the command must be used, and the SAVE file renamed to basename.COM. See Appendix D for the commands for your O/S and the CIS or L/II COBOL Operating Guide for fuller details of load directives. Thereafter the direct load command can be used.

Immediately the program is loaded, the user designed form is displayed.

The form remains on the screen throughout a run, processing being controlled by manipulation of the data in the variable fields.

A screen display reflects the structure of a single record. The required processing function is instigated by entering data and positioning the cursor as described below, and then pressing the RETURN key. Index program messages are displayed in an unused area of the screen as necessary.

The basic operator functions and Index program messages are described below, and will suffice in general use. Details of the Index program interpretation of data manipulation and cursor position follow this description.

Enquiry by Key Field

Amend key fields only to required key, and press RETURN. The required record is displayed. If the record is not found (i.e. key cannot be found) the message RECORD NOT FOUND is displayed.

Sequential Enquiry

Simply press RETURN to show next record. If the end of the file is reached, the message END OF FILE - RETURN WILL TERMINATE is displayed.

Amend Displayed Record

Amend data fields only and press RETURN. The message RECORD AMENDED is displayed.

Delete Displayed Record

Press the HOM or ↖ key then press RETURN. The message RECORD DELETED is displayed and the data fields are blanked out.

Insert New Record

Amend key and data fields as required and press RETURN. If the data fields currently displayed do not need changing (i.e. it is required to enter the existing data fields under a new key) prior to pressing RETURN, either press HOM (↖) OR press ↓ repeatedly until end of the last data field is reached.

The message NEW RECORD WRITTEN is displayed if insertion takes place.

If a record already exists with the specified key the current display is retained and the warning message RECORD ALREADY EXISTS WITH THIS KEY is displayed. The facilities available on the subsequent input are restricted to three as follows:

1. Force replacement of existing record:

   Either press HOM or press ↓ repeatedly until the cursor reaches the end of the last data field, then press RETURN.

   The record is replaced and, the message RECORD REPLACED is displayed.

2. Amend key field and re-attempt the insertion:

   Amend key fields and press RETURN (cursor position is immaterial).

3. Abandon insertion attempt and display existing record:

   Press RETURN only.

Terminate Run

Enquire up to the end-of-file by means of continual sequential enquiry or a combination of enquiry by key to a specific record, then sequential enquiry.

When end-of-file is reached the message END OF FILE - RETURN WILL TERMINATE is displayed. Press RETURN to terminate the run.

USE ON MULTI-USER SYSTEMS

The Index program is compatible with the FILESHARE optional product for use with LEVEL II and CIS COBOL.

The Index program will lock any record that you access, thus preventing another user from updating that record whilst you are processing it. The lock is released when you move on to another record. Similarly it will not make available to you a record which another user has locked.

The effects on Index operation are as follows:-

Enquiry

If you attempt to read a record, either by sequential enquiry or by key, and that record is locked, the message RECORD LOCKED is displayed. The key of the locked record is displayed but the data fields are blanked out. On the subsequent input you may:

1. Retry the enquiry,
   Press RETURN only.

2. Abandon the enquiry and read next record;
   Move cursor away from default (start-of-data)position and press RETURN.

3. All other functions operate as normal:
   i.e., as though the enquiry had failed with RECORD NOT FOUND.

Amend Displayed Record

This will be unaffected because the preceding read will have locked the record, denying any other user access.

Insert New Record

If a record exists with the specified key then RECORD ALREADY EXISTS WITH THIS KEY will be returned irrespective of whether the record is locked or not. The subsequent imput is as described earlier for that condition, except that:

1. If you attempt to force replacement and the record is locked, the message EXISTING RECORD LOCKED is displayed. The subsequent imput is again as described for the RECORD ALREADY EXISTS condition.

2. If you simply try to display the existing record and it is locked, this will be handled exactly as described under ENQUIRY.

NOTE: Immediately after update operations (Amend/Insert), Index attempts to read the record again to re-establish the lock:

Under exceptional circumstances it is possible for another user to read the record, thus locking it for his own purposes, before the lock can be re-established. In this case the normal message confirming the update will be displayed, immediately followed by the RECORD LOCKED message and blanking of the data fields. You will then be in the same position as if an enquiry had failed because of a locked record, and the subsequent input is as described under Enquiry

USER REQUIREMENT INTERPRETATION SUMMARY

The Index program interprets the user's requirements depending on the change status of key and data fields and the cursor position as follows:

Key and Data Fields Unchanged

The function performed depends upon cursor position as follows:

* If an end-of-file condition has just been reported, a request to terminate the run is assumed irrespective of cursor position.

* Otherwise if the cursor has been moved to the HOM position and a record is currently displayed, a delete request is assumed.

* If cursor is at start-of-data position (default) and a record-locked condition exists, a request to retry reading the locked record is assumed.

* If none of these conditions exists, a request to display the next record relative to the 'current' position in the file is assumed, and either the record is displayed, or a lock condition is reported, or an end-of-file condition is reported.

Key Changed and Data Unchanged

The function performed depends on cursor position as follows:

* If cursor is moved to either the HOM position or the last data character position, an attempt to insert a record is assumed, and processing is as described under Key and Data Changed

* Otherwise an enquiry with respect to this key is assumed, and either the record is displayed or its absence is reported, or a lock condition is reported.

Key Unchanged and Data Changed

This is a request to update the file. If a lock has previously been established against this record it will be amended. Otherwise an insert attempt is assumed, and processing is as described under Key and Data Changed.

Key and Data Changed

This is a request to insert a new record. However, it is assumed that the user should not overwrite a record without at least being informed of its presence. Therefore if a record exists with the specified key, a warning message is displayed, and the subsequent three functions can be performed depending on the change status of key and data fields and the cursor position:

1. Key and Data Unchanged

   The function required depends on cursor position as follows:

   * If the cursor has been moved to either the HOM position or the last data character position, a request to overwrite the existing record is assumed. Either the existing record is overwritten, or an existing-record-locked condition is reported in which case the subsequent input is processed in the same way as the current input.

   * If the cursor is at any other position a request to abandon the insertion attempt and display the existing record is assumed, and processing is as described under Key Changed and Data Unchanged.

2. Data Unchanged and Key Changed

   An attempt is made to insert the data under the new key irrespective of cursor position. If necessary the warning message will be repeated.

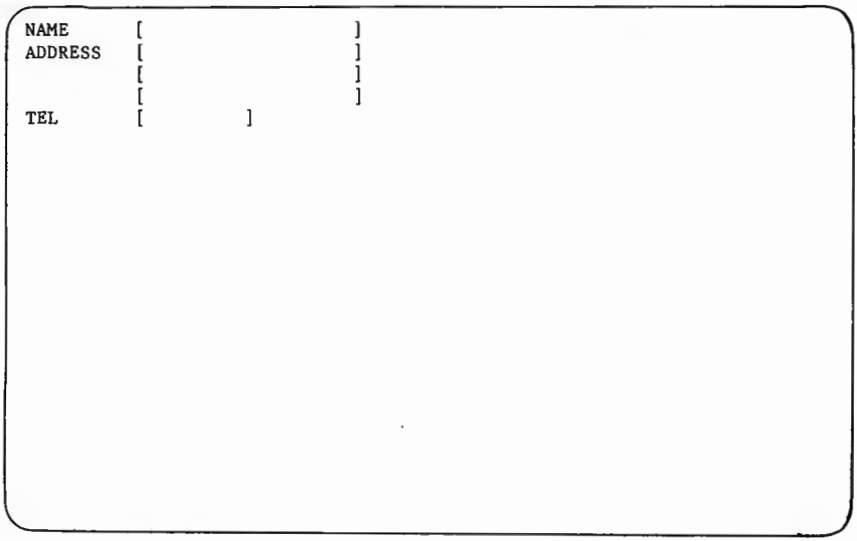
3. Key and Data Changed

   A normal insert request as described above is assumed.

CHAPTER 9

USER INDEX
PROGRAM EXAMPLE

It is required to generate an indexed sequential file that contains records of names, addresses and telephone numbers with name as key field, and process these records using the form as used in Chapter 7:



NAME and ADDRESS are alphanumeric fields and TEL is a numeric field.

At data entry time after insertion of the name, address and telephone number: P. Smith, 8 George Street, Plymouth, Devon, 88326, the form is required to appear as:

```
NAME      [SMITH. P          ]
ADDRESS   [8 George Street,  ]
          [Plymouth          ]
          [Devon             ]
TEL       [88326    ]
```

It is assumed the system is booted, the issued files have been copied to the CIS or L/II COBOL CP/M system disk so that the Index program can be compiled (Step 13), and that FORMS-2 has been configured from your CRT.

The following steps must then be carried out:

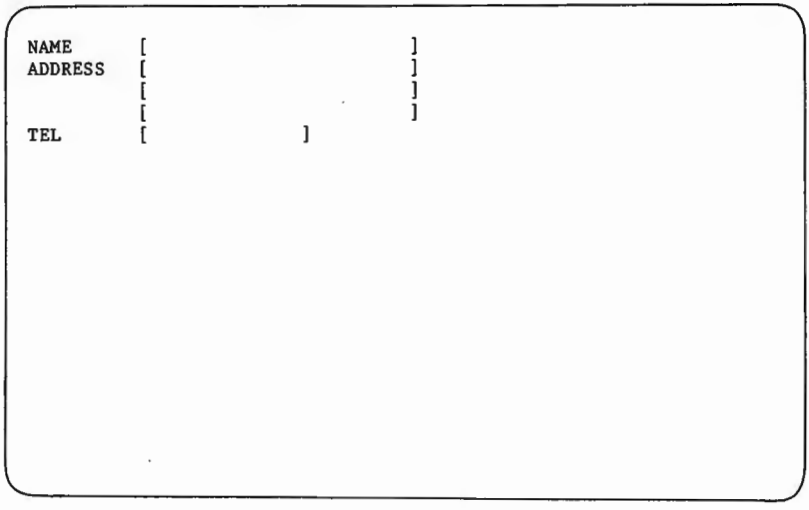
1. The operator loads FORMS-2 by entering the load command for your O/S. See Appendix D for specific Operating System format for this command.

2. FORMS-2 displays Screen I01 requesting a six-character base for file-names and data-names followed by three other questions. If the CRT is standard (24 lines), no further questions need be answered for this screen. Key DEM02 followed by the RETURN key if the default screen size (24) is correct.

3. FORMS-2 displays Screen I02 to request the output file option type and drive number. Key G then RETURN to select the option for the Index program.

4. FORMS-2 displays Screen W01 to request the Screen Type option. Note the default "A" and press the RETURN key.

5. FORMS-2 displays a blank screen with the end of window one line up from the bottom of the screen and delimiters in the bottom line. Use the cursor control keys and the normal character keys to set up the following text on the screen:



Press RETURN key.

6. FORMS-2 puts "__" in the top left hand of the screen. Press the SPACE and RETURN keys.

7. FORMS-2 processes the screen to create a fixed text form. This takes a short period and involves the following displays on the CRT:

   DDS source code as generated, followed by a redisplay of the fixed text as it is written to the Screen Image file.

   A message is then displayed giving the name of the fixed text Screen Image file created. Press RETURN as requested.

8. FORMS-2 displays screen W01 to request the Screen Type option. Note the default is "C" and press the RETURN key

9. FORMS-2 displays the fixed text screen as background data; now use the cursor control keys and key in X's to fill the NAME variable data field. Move cursor to the first character position in the address variable data field and then press RETURN to enter Command Mode. Enter * to set the first character position in the ADDRESS variable data field as the start of data position and then press RETURN. Continue to enter X's and 9's to fill the data fields as shown below.

```
NAME      [XXXXXXXXXXXXXXXXXXXX]
ADDRESS   [XXXXXXXXXXXXXXXXXXXX]
          [XXXXXXXXXXXXXXXXXXXX]
          [XXXXXXXXXXXXXXXXXXXX]
TEL       [9999999]
```

Note that you have now specified the NAME variable data field as the key field.

10. FORMS-2 displays "__" in the top left hand of the screen; press the SPACE and RETURN keys. A message is displayed showing validation in progress.

11. FORMS-2 processes the X's and 9's to create a variable data form, with the following displays to the CRT as it goes:

    DDS source code as generated, followed by a redisplay of the variable text as it is written to the Screen Image file.

12. FORMS-2 terminates automatically after displaying the end of run screen:

```
Files created DEMO2.DDS
              DEMO2.GEN
END OF FORMS2 RUN
```

13. Compile the Index program by typing the standard CIS or L/II COBOL compilation command for your Operating System (O/S) using the Index program source file name:

    DEM02.GEN

    and the COPYLIST directive of this command.

14. When the compilation finishes, the generated Index program DEM02 can be run by use of the standard run command for your O/S to load the program intermediate code from file:

    DEM02.INT

    The compilation and run commands for your O/S are described in Appendix D.

15. The generated Index DEM02 program will then run. Your screen as designed in step 9 is displayed. The fixed text form is displayed on the screen. The variable data fields are used to accept data.

16. You are now ready to practice all the file maintenance commands. The next steps show all of these in practice but you can vary the sequence or add any steps to these once you have gained confidence.

17. To insert the first record into the new indexed sequential file, simply key names and addresses into the screen format terminating each record by RETURN key. (Remember to enter surname first before initials to keep the application feasible).

18. Enter two more complete records overkeying all data from the previous record, because all displayed data is written to the file.

19. When three records have been inserted, you can amend the second record as follows:

    Enter the name field as for the second record added followed by RETURN. The whole record is displayed because the name is the key which finds that record. You have now seen the enquiry facility operated. All records can be recalled as easily as that.

20. Change the town field and press RETURN. The message RECORD AMENDED is displayed.

21. Press RETURN and the third record is diplayed. You could progress through a whole file in this way.

22. To delete the third record entered move the cursor to HOME position and press RETURN. The fields clear showing deletion of that record, and a message RECORD DELETED is displayed.

23. Press RETURN. The Index program attempts to show the next record but one does not exist so an end-of-file message is shown: END OF FILE REACHED - RETURN WILL TERMINATE.

24. Press RETURN with end-of-file showing and termination occurs.

    You have now seen the record handling method demonstrated and can, if you wish, generate a more ambitious Index program or reload as at step 14 and familiarise further with record manipulation.

25. Before you do this, however, you can examine the files on the disk.

DEM02.DDS	-	Data Description Statements for form (COBOL source)
DEM02.GEN	-	Source code of Index program DEM02
DEM02.INT	-	Intermediate code of Index program DEM02
DEM02.LST	-	List file code of Index program DEM02
DEM02.IDX	-	Index file
DEM02.DAT	-	Sequential data file

Indexed Sequential file

NOTES:

1. The two files DEM02.IDX and DEM02.DAT constitute the Indexed Sequential file created by the generated Index program, and in any further runs of this program these two files will be used.

2. This example shows the demonstration program run in a single-user environment. The program does however include facilities for use in a file-sharing environment with the Micro Focus FILESHARE package and appropriate error handling and notification is provided if used in this environment and locked files or records are encountered.

APPENDIX A

INITIALISATION SCREENS

The Initialisation screens I01 and I02 shown below are displayed automatically at Initialisation Phase (See Chapter 2).

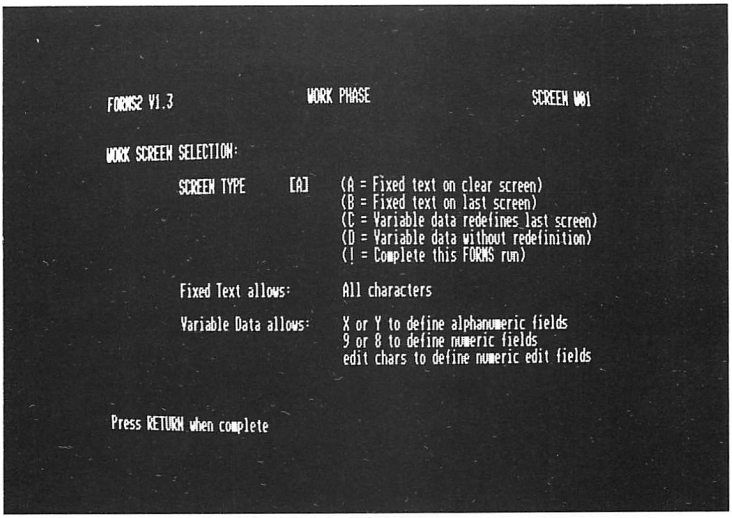
```
A)TYPE FORMS2.I01
FORMS2 V1.3              INITIALISATION PHASE                SCREEN I01

FORMS2 PARAMETERS:

          DATA-NAME & FILE-NAME    [     ]  (1-6 alphanumeric characters)

          CRT lines                [24]     (22 or 23 or 24)

SPECIAL-NAMES clause:

          CURRENCY SIGN            [$]      (ANSI currency signs only)

          DECIMAL-POINT            [.]      ('.' or ',')


 Press RETURN when complete

 A)_
```

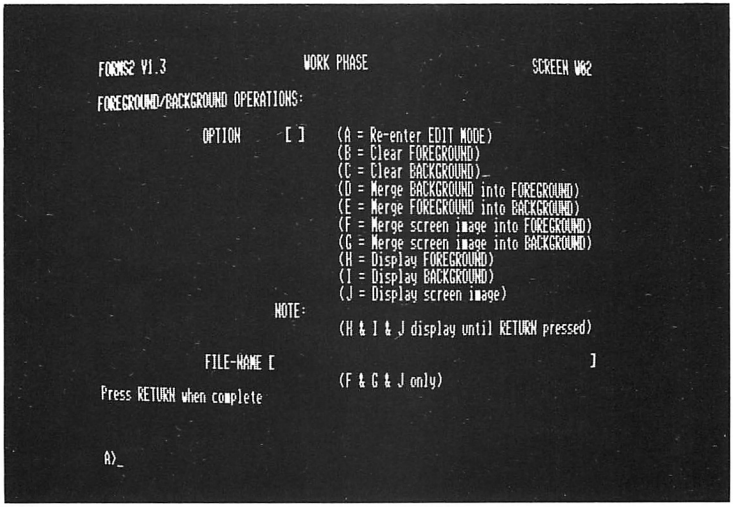
```
FORMS2 V1.3              INITIALISATION PHASE                SCREEN I02

FILES TO BE CREATED:

          FILE COMBINATIONS        [C]      (A = DDS)
                                            (B = DDS & CHK)
                                            (C = DDS & CHK & Snn)
                                            (D = DDS & Snn)
                                            (E = Snn)
                                            (F = No files output)
                                            (G = DDS & Snn & GEN)

DEVICE/DIRECTORY PREFIX (0-40 Chars) [                              ]


Press RETURN when complete
```

APPENDIX B

WORK SCREENS

The Work Screen W01 shown below is displayed at the start of the Work Phase. Work screen W02 can be summoned by the F command (See Chapter 3).

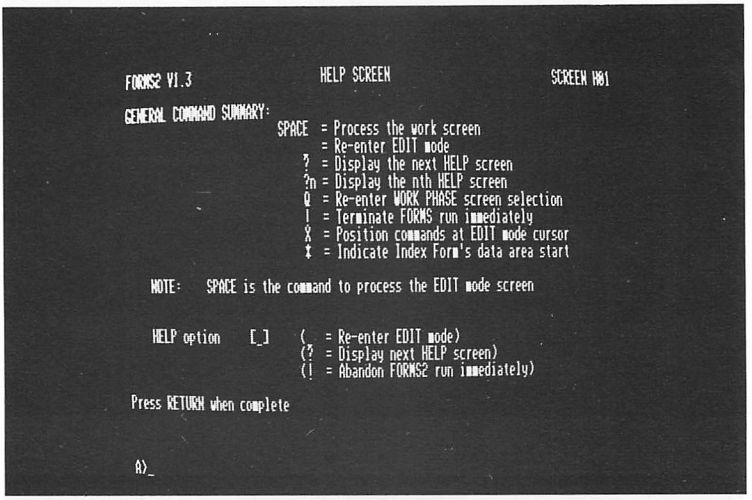


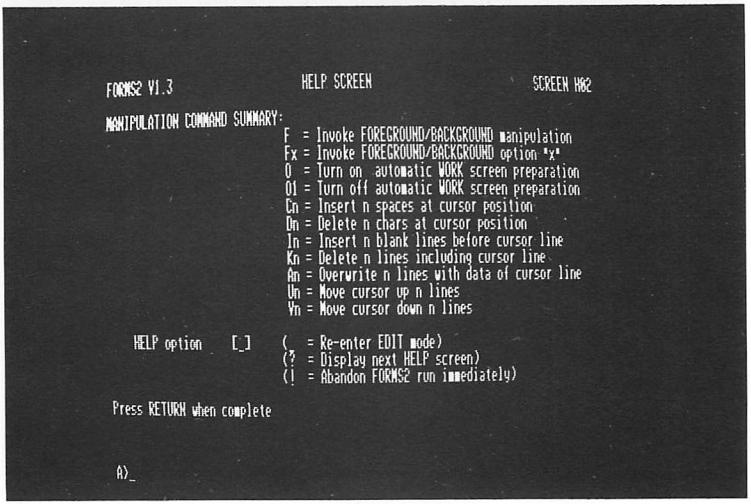


APPENDIX C

HELP SCREENS

The four screens contained in this Appendix (H01 to H04) can be summoned for display by the Operator keying ? from Command Mode at any time. They are intended for advisory purposes only. The command ?n (where n is the number 1 to 4 corresponding to the screen number H0n) summons a particular Help Screen.





```
FORMS2 V1.3                    HELP SCREEN                        SCREEN H03

PROGRAMMING COMMAND SUMMARY:
                                G  = Give datanames screen coordinates suffix
                                G1 = Give datanames sequential number suffix
                                Jn = Allow up to n consec. spaces in fixed text
                                Mx = Interpret "x" as "space"
                                S  = Cancel previous Sn command
                                S1 = Inhibit DDS & CHK output at next processing
                                S2 = Inhibit Snn output at next processing
                                S3 = Prompt for Snn file-name at next processing
                                S9 = Line edit DDS output at next processing
                                P  = Display cursor position coordinates


   HELP option     [_]     (  = Re-enter EDIT mode)
                           (? = Display next HELP screen)
                           (! = Abandon FORMS2 run immediately)

Press RETURN when complete


A)_
```

```
FORMS2 V1.3                    HELP SCREEN                        SCREEN H04

WINDOW COMMAND SUMMARY:
                                W  = Position cursor to current window start
                                W1 = Start window at cursor line
                                W2 = End window at cursor line
                                W3 = Start window at cursor line, no delim's
                                W4 = End window at cursor line, no delim's
                                W5 = Display start window delimiters
                                W6 = Display end window delimiters
                                W7 = Re-display data overwritten by start delim's
                                W8 = Re-display data overwritten by end delim's
                                W9 = Position cursor to current window end


   HELP option     [_]     (  = Re-enter EDIT mode)
                           (? = Display next HELP screen)
                           (! = Abandon FORMS2 run immediately)

Press RETURN when complete


A)_
```

APPENDIX D

OPERATING FORMS-2 WITH CP/M

To clarify the sequences, the CP/M prompts are included before the commands in this Appendix. The symbol << indicates that the RETURN key or equivalent should be pressed once.

It is assumed that CP/M is loaded and that the issued files have been copied to the CIS COBOL CP/M system disk.

NOTE:

FORMS-2 is preconfigured for your CRT. It can be reconfigured if necessary for any other special CRT.

FORMS-2 LOADING

To load FORMS-2 as issued, the command is:

```
B>FORMS2<<
```

FOMRS-2 CHECKOUT PROGRAM COMPILATION

To compile the Check-Out program that enables you to check your fixed text and variable data fields, the general command is:

```
B>COBOL basename.CHK COPYLIST<<
```

In the sample runs in Chapters 2, 7 and 9 basename is, of course, DEMO1 and DEMO2 respectively.

FORMS-2 CHECK-OUT PROGRAM RUNNING

The Check-Out program that enables you to check your fixed text and variable data fields can be loaded immediately after compilation by the general command:

```
B>RUN basename.INT<<
```

In the sample runs in Chapters 7 and 9, basename is, of course, DEMO1 and DEMO2 respectively.

To be able to load the Check-Out program directly in subsequent use, the following general command is entered:

```
B>RUN = basename.INT<<
```

Thereafter the general command following can be used to load the Check-Out program:

```
B>basename<<
```

FORMS-2 INDEX PROGRAM COMPILATION

To enable the Index program that processes an indexed sequential data file from your FORMS-2 screens to be compiled the following general command is entered:

```
B>COBOL basename.GEN COPYLIST<<
```

FORMS-2 INDEX PROGRAM RUNNING

The Index program that processes an indexed sequential data file from your FORMS-2 screens can be loaded immediately after compilation by the general command:

```
B>RUN basename.INT<<
```

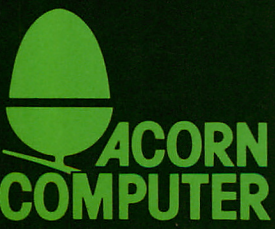
In the sample run in Chapter 9 basename is, of course, DEMO2.

To be able to load the Index program directly in subsequent use, the following general command is entered:

```
B>RUN = basename.INT<<
```

Thereafter the general command following can be used to load the Index program:

```
B>basename<<
```



Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, England

Printed by Saunders & Williams (Printers) Ltd, Croydon, Surrey