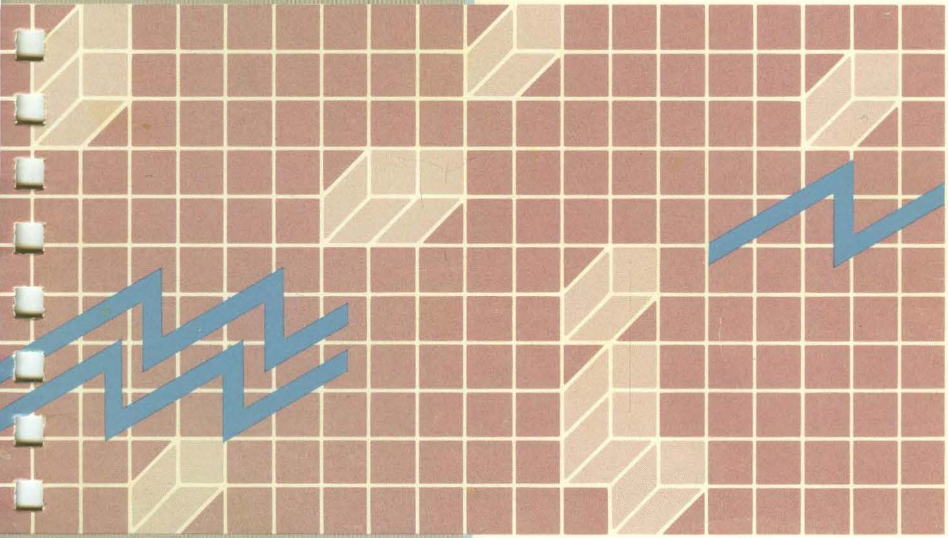
Introduction to LEVEL II COBOL™



Introduction to LEVEL II COBOL™

First Edition (May 1988) 73–00473–A

This document was produced using the Document Designer Series.

Contents

About this Manual

This manual is your introduction to using LEVEL II COBOL™ in the CTOS™ environment. It is a companion manual to the *High Performance LEVEL II COBOL Operating Guide*, the *High Performance LEVEL II COBOL Language Reference Manual*, and the *High Performance LEVEL II COBOL Error Messages Manual.*

Use this manual for an overview of LEVEL II COBOL and to learn how to compile and link your program on CTOS. This manual also provides information about making calls that use the CTOS operating system libraries, such as the Forms library and CTOS.lib.

The three Micro Focus companion manuals describe how to use LEVEL II COBOL in the MS-DOS and CPM-86 environments. The *High Performance LEVEL II COBOL Language Reference Manual* describes the COBOL ANSI standard language. The *High Performance LEVEL II COBOL Operating Guide*, describes the LEVEL II COBOL compiler features.

In general, LEVEL II COBOL works very similarly under CTOS as it works under MS-DOS or CPM-86. This manual, *Introduction to LEVEL II COBOL,* explains those concepts and tasks that are different under CTOS, so you will want to read this manual carefully before using the companion manuals.

Use the companion manuals for detailed reference about the LEVEL II COBOL language and compiler features. Note that in most cases this manual describes the information you need to know first, then references the other manuals for additional information.

LEVEL II COBOL runs in real mode under either the CTOS or the CTOS/VM operating system. This manual assumes that you are familar with the basic concepts of the Executive and your operating system. If you are not, take some time to look at the Executive manuals for material on how to use the command line interpreter and a basic description of the file system, the "Getting Started" chapter of the *CTOS Programmer's Guide* for an introduction on programming in the CTOS environment, the *Linker Manual* for material on Linker concepts, and the *CTOS/VM Concepts Manual* for an introduction to the operating system itself.

This manual also assumes that LEVEL II COBOL is installed on your workstation. If it is not, see the *LEVEL II COBOL Release Notice* for installation instructions. Note that you should ignore *Getting Stared with High Performance Level II COBOL on MS-DOS* if it is bound with your *Operating Guide*. Use the release notice instead for installation information.

Keyboard Differences between CTOS workstations and PC's

CTOS uses a different keyboard than is used by most MS-DOS machines. The Micro Focus manuals occasionally refer to explicit key names, so you should understand these differences. The table below shows the differences

Table 1. Keyboard Differences

DOS key	CTOS key
Enter*	**Go**
Escape	**Cancel**
Control	**Code**
Home*	not applicable

*For use with COBOL ACCEPT statement, this key is a configuration option. See Chapter 7 for more information.

LEVEL II COBOL offers a special configuration program that can be used to set up keyboard mapping. For example, it is used to specify a

Home key. The configuration program is described in Chapter 7 of this manual.

Related Documentation

Besides the Micro Focus manuals discussed above, there are several manuals that you may need to use as you familarize yourself with writing COBOL programs in the CTOS environment. For a complete list of workstation documentation, see the *Convergent Publications Catalog*.

The *CTOS Programmer's Guide* is a reference guide for programming under the CTOS operating system. It describes CTOS programming practices and introduces the system to programmers who are using it for the first time.

The *CTOS/VM Concepts Manual*, together with the *CTOS/VM Reference Manual*, describes the CTOS/VM operating system. The *CTOS/VM Concepts Manual* introduces the CTOS/VM operating system to the programmer by presenting concepts in a basic-to-advanced order. Included among the concepts in this manual are management of processes, messages, memory, exchanges, video, keyboard, files, disks, printers, communications, tape, and timers. CTOS/VM operations pertaining to each concept are described briefly at the end of each chapter. The manual also explains how to use the CTOS/VM operations and provides information on the administrative aspects of the operating system.

The *CTOS/VM Reference Manual*, together with the *CTOS/VM Concepts Manual*, describes the CTOS/VM operating system. The *CTOS/VM Reference Manual* describes each operation that is contained in the System Image and in the standard object module library, CTOS.lib. The manual also contains the format of each system structure.

The *Executive Manual* describes the interactive command interpreter that interacts with the CTOS and CTOS/VM operating systems. The manual is both a user's guide and a reference to the available commands. It addresses command execution, file management and protection, and program invocation. The manual also provides descriptions and details about parameter fields for Executive commands.

The *Forms Manual* describes the Forms facility that includes the Forms Editor, which is used to interactively design and edit forms, and the Forms run time, which is called from an application program to display forms and accept user input.

The *ISAM Manual* describes both the single–user and the multiuser indexed sequential access method. It specifies the procedural interfaces (and how to call them from various languages) and the utilities.

The *Linker/Librarian Manual* describes both the Linker, which links together separately compiled object files, and the Librarian, which builds and manages libraries of object modules.

The *Status Codes Manual* contains a complete list of all the status codes that can be generated by a CTOS workstation or a Shared Resource Processor (SRP), including bootstrap ROM error codes and CTOS initialization codes. The manual also describes and interprets error status codes.

Conventions

Italics are used to show variable items, for example, file names when you would supply your own file name.

All numbers given are decimal unless indicated.

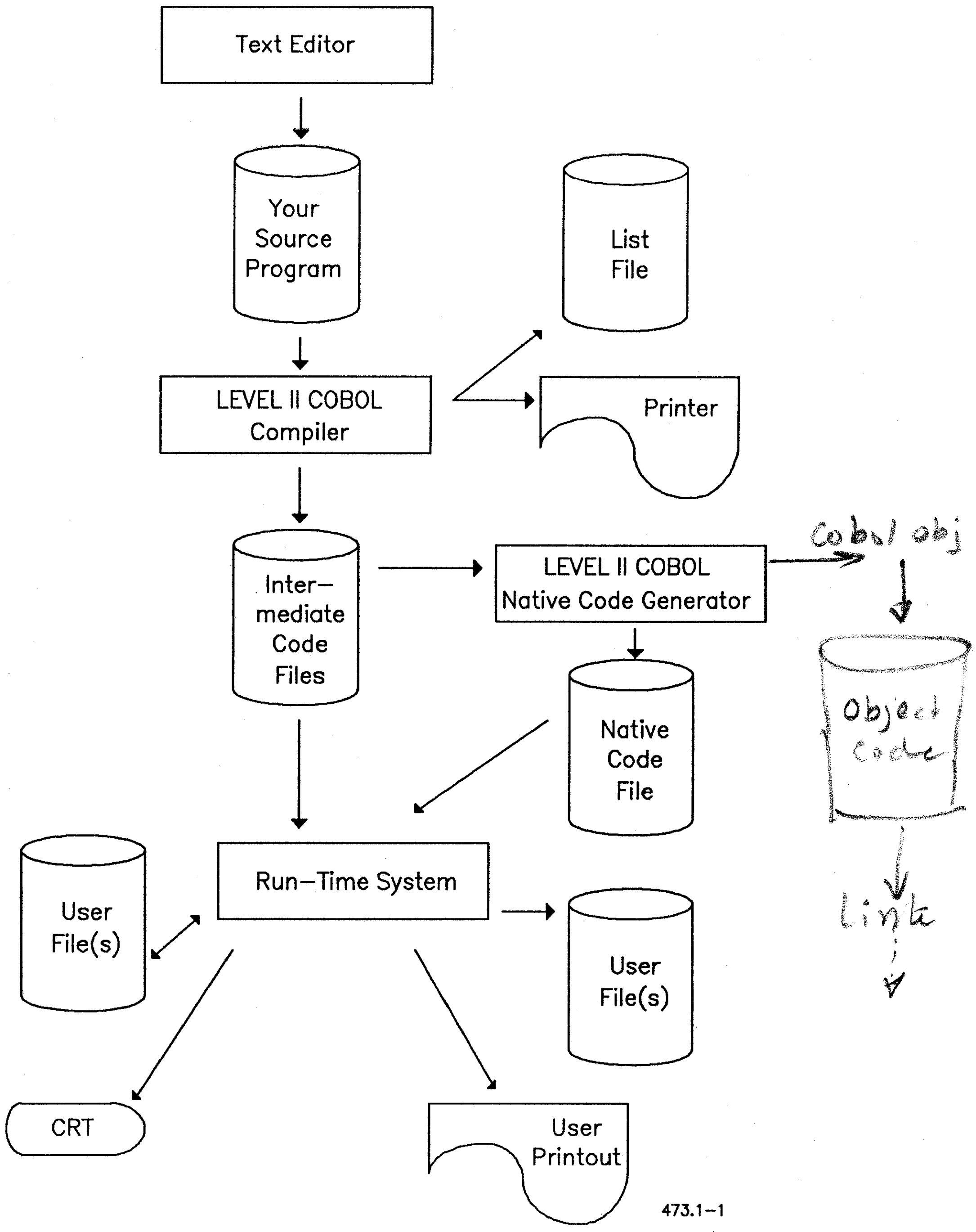
1 Introduction

LEVEL II COBOL is a standard COBOL language system for use in the CTOS environment. It is ANSI '74 COBOL as specified in *American National Standard Programming Language COBOL* (ANSI X3.23 1974). LEVEL II COBOL runs in real mode under either the CTOS or the CTOS/VM operating system.

To create a LEVEL II COBOL program, you must first create source files using one of the workstation text editors. The program must then be compiled using the **COBOL Compile** command. The resultant file is an intermediate file. The LEVEL II COBOL Run–Time System can interpret and execute the intermediate file. You use the **COBOL Run** command to invoke the COBOL Run–Time System (RTS).

After you have finished the debugging stages of programming, you may want to use the **COBOL Generate** command (the Native Code Generator) to convert the intermediate code into a native code file (mostly machine language), which will execute more quickly. The native code file is also executed through the Run–Time System with the **COBOL Run** command. See the figure on the next page for a graphic description of this process.

The commands **COBOL Compile**, **COBOL Generate**, and **COBOL Run** are all described in subsequent chapters of this manual.


Text Editor
Your Source Program
List File
LEVEL II COBOL Compiler
Printer
Inter–mediate Code Files
LEVEL II COBOL Native Code Generator
Cobol obj
Object code
Native Code File
Run–Time System
User File(s)
User File(s)
Link
CRT
User Printout

473.1–1

While you are debugging, you can use the ANIMATOR to make the task easier. The ANIMATOR works with the Run-Time System, so that you can debug your program interactively. The ANIMATOR, which you execute using the **COBOL Animate** command, is also described later in this manual.

Note that the Micro Focus manuals that describe LEVEL II COBOL refer to an interactive Command Line Interpreter, which allows programs to be entered interactively as they are executed. This feature is not supported in the CTOS environment.

The Micro Focus manuals also refer to a product called FORMS-2. This forms product is also not supported in the CTOS environment. To use forms with your COBOL program, use the CTOS product named Forms.

Ignore any references to the Command Line Interpreter or the FORMS-2 products that you find in the Micro Focus manuals.

There are some differences between the MS-DOS version of LEVEL II COBOL described in the *High Performance LEVEL II COBOL Operating Guide* and the CTOS version. For example, the Micro Focus manuals describe a .BIN file name extension used with CALL and a SAVE86 file used to create a custom command. When such differences occur, they are called out in this manual. Be sure to read this manual as your first source, then use the *High Performance LEVEL II COBOL Operating Guide*.

2 Writing a COBOL Program on CTOS

I guess → All of this chapter is still valid for Cobol 3.1

This chapter describes special information you need to know when you are writing your LEVEL II COBOL programs. It frequently refers you to the *High Performance LEVEL II COBOL Operating Guide*.

You can use any standard workstation text editor to create your source code. Follow the standards for COBOL described in the *High Performance LEVEL II COBOL Language Reference Manual*. The compiler rejects most non–alphanumeric characters embedded in source code, unless they are part of literal strings enclosed in quotes.

When you have completed your program, use the **COBOL Compile** command, described in Chapter 3 of this manual, to compile it into intermediate code. The Run–Time System (**COBOL Run** command) can then be used to execute the intermediate code.

Chapter 4, "Writing COBOL Programs," and Chapter 5, "CRT Screen Handling," in the *High Performance LEVEL II COBOL Operating Guide* give useful information on writing efficient programs and writing interactive screen programs. Note that with CTOS you do not have to use the file **Config.Pfk** referred to in Chapter 5, "CRT Screen Handling."

You will also find several useful programming examples are distributed with LEVEL II COBOL. See the *LEVEL II COBOL Release Notice* for more information.

If you have a program with a particularly large PROCEDURE DIVISION, you may want to take advantage of LEVEL II COBOL's segmentation feature to split the PROCEDURE DIVISION into a small permanent segment and multiple overlays, called independent segments. This is discussed in detail in Chapter 2, "Handling Large Programs," in the *High Performance LEVEL II COBOL Operating Guide*. Note that the overlays discussed in that chapter are a COBOL feature and are not the same as CTOS Virtual Code Segment Management, which you cannot use from COBOL.

You might also want to make a particularly large program into several smaller programs. LEVEL II COBOL supports the ability to dynamically call programs or subprograms, whether written in COBOL or in assembly language. You can also make calls to compiled subprograms originally written in other languages, as long as they have been compiled using the Intel medium model of computation.

Using the CALL verb, you can:

- CALL another COBOL module using ANSI standard Inter–Program Communication.
- CALL LEVEL II COBOL extensions that are embedded in the Run–Time System. The extensions provided with LEVEL II COBOL are described in Chapter 3 of the *High Performance LEVEL II COBOL Operating Guide*.
- CALL non–COBOL procedures that are linked into the COBOL Run–Time System. For example, you can make calls to library functions such as the Forms library functions or CTOS.lib functions using the CALL verb. You can also call compiled procedures written in other languages in this manner. You must explicitly link with the necessary routines to be able to use this feature.

Using CALL for Inter–Program Communication

A COBOL application system can consist of more than one separately compiled program. The set of COBOL programs that constitute the application are known as the application suite. The COBOL application suite is run by using the file name of the main program. All programs other than the main program should have a LINKAGE SECTION in the DATA DIVISION. The LINKAGE SECTION permits COBOL programs to communicate, that is, pass parameters.

Programs communicate with each other using the CALL verb. The general format of the CALL verb is discussed in detail in the *High Performance LEVEL II COBOL Language Reference Manual.*

Use CALL to invoke each program or sub–program as needed. Called programs are not linked into the main program, so they must be compiled separately into intermediate or native code and must be physically present to be used. Up to 64 calls can be made from any one program.

When the CALL verb is executed, the intermediate code of the called program is loaded into memory, assuming there is sufficient space. The ON OVERFLOW verb detects whether a CALL has failed due to lack of memory space. The CANCEL verb reclaims memory that was allocated to programs which are no longer in use. The DATA DIVISION of each program in the application suite must fit in memory. If it does not, the CALL is ignored. You can use the ON OVERFLOW verb to detect this.

One segment of the PROCEDURE DIVISION must also fit into memory. If necessary, other PROCEDURE DIVISION modules are overwritten and reloaded as required.

Using CALL to Take Advantage of Run–Time Extensions

Several useful procedures, such as procedures to read or write to a specific memory location, are embedded in the Run–Time System. These too are used through the CALL verb.

Most of the Run–Time extensions are described on pages 3–11 to 3–34 of the *High Performance LEVEL II COBOL Operating Guide.* Note that the references to *Filename*.**Bin** at the beginning of Chapter 3 can be ignored by the CTOS user.

The additional extensions provided with the CTOS version of LEVEL II COBOL are described on the following pages.

All the built in procedures can be called without configuring COBOL.

ConvertWord

The COBOL internal representation for storing data is based on the format used for Motorola memory, which is the inverse of the representation used in Intel memory. Since workstations use Intel 808*x*6 processors, you may need to convert from one format to another.

The ConvertWord procedure reorders the bytes that comprise *Word–In* and stores the results in *Word–Out*. *Word–In* and *Word–Out* may be the same data item.

Syntax:

```
Call "&ConvertWord" USING  WORD-IN, WORD-OUT.
```

where

```
WORD-IN  IS   PIC 9(4) USAGE IS COMP.
WORD-OUT IS  PIC 9(4) USAGE IS COMP.
```

ConvertQuad

The COBOL internal representation for storing data is based on the format used for Motorola memory, which is the inverse of the representation used in Intel memory. Since workstations use Intel 808*x*6 processors, you may need to convert from one format to another.

Reorders the bytes that comprise *Quad–In* and stores the results in *Quad–Out*. *Quad–In* and *Quad–Out* may be the same data item.

Syntax:

```
CALL "& ConvertQuad" USING QUAD-IN, QUAD-OUT.
```

where

```
QUAD-IN IS     PIC 9(9) USAGE IS COMP.
QUAD-OUT IS    PIC 9(9) USAGE IS COMP.
```

GetPointers

GetPointers is used when you need to use pointer arithmetic. The memory address of *Data-Val* is stored in *Pointer-Val.*

Syntax:

```
CALL "&GetPointer" USING POINTER-VAL, DATA-VAL.
```

where

```
POINTER-VAL IS  PIC 9(9) USAGE  IS COMP.
DATA-VAL IS     any picture clause.
```

MakePointer

The pointer whose segment address is *Segment–Addr* and relative address is *Relative–Addr* is stored in *Pointer–Val*.

Syntax:

```
CALL "MakePointer" USING POINTER,-VAL SEGMENT-ADDR, RELATIVE-
ADDR.
```

where

```
POINTER-VAL IS      PIC 9(9) USAGE IS COMP.
SEGMENT-ADDR IS PIC 9(4) USAGE IS COMP.
RELATIVE-ADDR IS  PIC 9(4) USAGE IS COMP.
```

UnMakePointer

The segment address portion of *Pointer–Val* is stored in *Segment–Addr*. The relative address portion is stored in *Relative–Addr*.

Syntax:

```
CALL "&UnMakePointer" USING POINTER-VAL, SEGMENT-ADDR,
RELATIVE-ADDR.
```

where

```
POINTER-VAL IS      PIC 9(9) USAGE IS COMP.
SEGMENT-ADDR IS     PIC 9(4) USAGE IS COMP.
RELATIVE-ADDR IS    PIC 9(4) USAGE IS COMP.
```

WordAligned

Some CTOS procedures, such as OpenByteStream, requre the use of word–aligned buffers. The Word–Aligned procedure checks for a word–aligned buffer.

If *Data–Val* is word aligned, a nonzero value is stored in *Flag*; otherwise, 0 is stored in *Data–Val*.

Syntax:

```
    CALL "&WordAligned" USING FLAG, DATA-VAL.
```

where

FLAG IS	PIC 9(2) USAGE IS COMP.
DATA–VAL IS	any picture clause.

Using CALL to Invoke Procedures Linked into the Run–Time System

LEVEL II COBOL can directly call non–COBOL procedures when they have been linked into the Run–Time System. This is how you call the Forms library routines or use CTOS.lib routines.

Note that COBOL has its own procedures that access the operating system and work effectively on all workstations. You only need to link CTOS.lib into the Run–Time System if you want to use CTOS.lib routines explicitly.

See the section entitled, "Linking Procedures into the COBOL Run–Time System," in Chapter 7, "Configuring COBOL," for instructions on how to set up the Run–Time System for this.

When using the CALL verb to invoke non–COBOL procedures, the object of the CALL is the non–numeric literal that is the name of the procedure, preceded by the ampersand (&) character.

For example, to call the CTOS Exit procedure write

```
CALL "&EXIT".
```

You can write the name of the procedure in either upper or lower case.

If the non–COBOL procedure does not return a value, pass the number of parameters required by the procedure. For example, the CTOS ErrorExit procedure requires one parameter, a status code, and does not return a value. To call this procedure, write

```
CALL "&ERROREXIT" USING ercEXIT.
```

If the procedure returns a value, pass an extra parameter at the beginning of the parameter list to receive the returned data. For example, the CTOS CloseFile procedure requires one parameter, a file handle, and returns a status code. To call this procedure write

```
CALL "&CLOSEFILE" USING erc, fh.
```

When passing a parameter, COBOL passes either its address or its value, depending upon the interface of the called procedure. This is explained in more detail below.

The COBOL Run–Time System provides several checks to detect incorrect procedure calls. These include checks for calling an unknown procedure and calling a procedure with an incorrect number of parameters.

Parameter Passing and Parameter Data Types

COBOL passes either parameter addresses or values, depending on the interface of the called procedure.

The Run–Time System gets information about procedure interfaces from the assembly language module COBOLGen.Asm. COBOLGen.Asm is discussed in more detail in Chapter 7 of this manual, "Configuring COBOL."

COBOL can pass bytes, byte strings, words, and double words (quads). However, COBOL cannot correctly pass structures containing words and quads unless certain type conversion statements are added to the COBOL program. This is explained in the section, "Passing Structures as Parameters," below.

The data types that can be passed between a COBOL program and a non–COBOL procedure are also described later in this chapter.

Byte

A byte is an 8–bit quantity, usually representing a character, an integer, or a boolean value.

The COBOL PICTURE clauses that define a byte are

```
    PICTURE 9(2) USAGE IS COMP.
```

which defines an integer or boolean value (true or false), and

```
    PICTURE X(1).
```

which defines a character.

When using bytes as boolean values, 0 means false and 1 means true.

The COBOL statements below show the definition and use of a byte parameter as a character (b) and as a boolean value (f0n).

```
    01 b          PICTURE X(1) VALUE "A".
    01 f0n        PICTURE 9(2) USAGE IS COMP VALUE 0.

    CALL "&WriteByte" USING erc, bswa, b.
    CALL "&SetKbdUnencodedMode" USING erc, f0n.
```

Byte String

A byte string is a contiguous sequence of bytes or characters.

The COBOL PICTURE clause that defines a byte string is

```
PICTURE X(n)
```

where n is the length of the byte string.

The COBOL statements below show the definition and use of byte string parameters (rgbFilename and rgbPassword).

```
01 rgbFilename     PIC X(8) VALUE "Testfile".
01 rgbPassword     PIC X(5) VALUE "xyzzy".

CALL "&OpenFile" USING erc, fh, rgbFilename, cbFilename, rgbPassword,
cbPassword, mode.
```

Word

A word is an 8-bit quantity, normally representing an integer.

The COBOL PICTURE clauses that define a word are

```
PICTURE 9(4) USAGE IS COMP
```

which defines an integer and

```
PICTURE X(2)
```

which defines two contiguous bytes.

The COBOL statements below show the definition and use of word parameters (erc, fh, cbFilename, cbPassword, mode) and byte strings (rgbFilename, RgbPassword).

```
01 erc            PIC 9(4) USAGE IS COMP.
01 fh             PIC 9(4) USAGE IS COMP.
01 rgbFilename    PIC X(8) VALUE "TestFile".
01 cbfilename     PIC 9(4) USAGE IS COMP VALUE 8.
01 rgbPassword    PIC X(5) VALUE "xyzzy".
01 cbPassword     PIC 9(4) USAGE IS COMP VALUE 5.
01 mode           PIC X(2) VALUE "mm".
.
.
.

CALL "&OpenFile" USING erc, fh, rgbFilename, cbFilename, rgbPassword,
cbPassword, mode.
```

Quad

A quad is a 32–bit quantity, normally representing an address (pointer) or a logical file address (lfa).

The COBOL PICTURE clause that defines a quad is

```
PICTURE 9(9) USAGE IS COMP.
```

The COBOL statements below show the definition and use of a quad parameter (pSegment).

```
01 Erc         PIC 9(4) USAGE IS COMP.
01 cBytes      PIC 9(4) USAGE IS COMP, VALUE1024.
01 pSegment    PIC 9(9) USAGE IS COMP.
.
.
.

CALL "&AllocMemorySL" USING erc, cBytes, pSegment.
```

Passing Structures as Parameters

Some procedures require structures as parameters. In these cases, the address of the structure is actually passed.

A structure is a contiguous group of data items. The individual data items are bytes, byte strings, words, and quads.

For example, the procedure RGParam takes a structure as a parameter. The interface for RGParam is

```
RgParam (iParam, jParam, pSdRet) : ErcType.
```

The final parameter, pSdRet, is a structure composed of a quad (pointer) followed by a word.

COBOL cannot correctly pass structures as parameters. COBOL stores the bytes that make up words and quads in a different order than is expected by non-COBOL procedures. The COBOL Run-Time System automatically reorders bytes for simple word and quad parameters, however, reordering does not occur for structures.

Two built-in non-COBOL procedures, ConvertWord and ConvertQuad, are provided so that you can explicitly reorder the word and quad components of a structure parameter. They are described in the section below.

If the word or quad contained in the structure is read by the non-COBOL procedure, call ConvertWord or ConvertQuad *before* the CALL to the non-COBOL procedure. If, however, the word or quad is read by the non-COBOL procedure, call them *after* the CALL to the non-COBOL procedure.

In the case of RGParam, the sd structure is written by the procedures. The following example demonstrates a CALL to RgParam.

```
01 erc                       PIC 9(4) COMP.
01 iParam                    PIC 9(4) COMP.
01 jParam                    PIC 9(4) COMP.
01 sd.
  03 pb                      PIC 9(9) COMP.
  03 cb                      PIC 9(9) COMP.

CALL "&RgParam"              USING erc, iParam, jParam, sd.
CALL ConvertQuad             USING pb, pb.
CALL ConvertWord             USING cb, cb.
```

Passing Parameters to the Forms Run–Time

The *Forms Manual* describes use of the Forms Run–Time in detail. You will probably want to be somewhat familiar with Forms to understand these examples.

COBOL correctly passes parameters, including structures, to all procedures in the Forms Run–Time if the parameter data definitions contained in the library file COBOLForms.edf are used. COBOLForms.edf is distributed with LEVEL II COBOL. Check your *LEVEL II COBOL Release Notice* for more information.

To use COBOLForms.edf, first copy it from the distribution media to your working directory. Next, insert the following statement in the WORKING–STORAGE section of the COBOL program that uses Forms.

```
    Copy "COBOLForms.edf".
```

The COPY statement causes the parameter data definitions in COBOLForms.edf to be included in your COBOL program. These definitions are listed in the example on the next page.

The parameter data definitions listed in the example are needed in the Forms procedures GetFieldInfo and UserFillField.

Use fieldInfo and cbFieldInfo as the last two parameters to GetFieldInfo. When GetFieldInfo returns, field information is accessed by referencing the elementary data items subordinate to fieldInfo.

Use InitState and ExitState as the last two parameters to UserFillField. Initialize the *init–ich* field of InitState before the call to UserFillField. When UserFillField returns, field state is accessed by referencing the elementary items subordinate to ExitState.

```
01  InitState
   02 init-ich                PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 filler                  PIC X(6) COMP.

01  ExitState
   02 exit-ich                PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 exit-ich                PIC X(1).
   02 filler                  PIC X(1).
   02 fAutoExit               PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 fModified               PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 fEmpty                  PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 filler                  PIC X(6).

01  cbFieldInfo               PIC 9(2) COMP VALUE 32.

01  fieldInfo.
   02 info-iCol               PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-iLine              PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-cCol               PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-fShowDefault        PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-fAutoExit          PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-fRepeating         PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-attrSel            PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-attrUnSel          PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-indexFirst         PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-indexLast          PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 filler                  PIC X(10).
   02 info-cchDefault         PIC 9(2) COMP.
   02 filler                  PIC 9(2) COMP.
   02 info-rgchDefault        PIC X(n).
   where n is the value of info-cchDefault
```

Device and File Management

LEVEL II COBOL offers four methods of file organization: sequential, line sequential, relative, and indexed sequential.

Note that only files of ORGANIZATION INDEXED can be shared for read/write access from COBOL. Access is controlled by the CTOS ISAM system service, which runs on the master workstation when operating in a cluster configuration. Other file types will return the "file locked" status if write-sharing is attempted.

File assignment is discussed on pages 6–5 to 6–9 in Chapter 6 of the *High Performance LEVEL II COBOL Operating Guide*. That manual is designed for the MS–DOS or CPM–86 user, so bear in mind that for use on CTOS you need to use CTOS device and file specifications in place of any device or file specifications used as examples. For example, use [VID] not CON: for the video display. See the *Executive Manual* or the *CTOS/VM Concepts Manual* if you need more information on how to use CTOS device and file specifications. Note too that drive residency as discussed in the *High Performance LEVEL II COBOL Operating Guide* is not an issue for CTOS users.

Sequential Files

Sequential files (ORGANIZATION IS SEQUENTIAL) are read and written using fixed length records. Use pages 6–9 to 6–14 of the *High Performance LEVEL II COBOL Operating Guide* to learn about sequential files.

Sequential files cannot be shared for read/write access. Use Indexed Sequential files instead if you need this capability.

Note that these are stored as CTOS Sequential Access Method files. The Sequential Access Method (SAM) is discussed in detail in the *CTOS/VM Concepts Manual*.

Line Sequential Files

Line Sequential files support variable length records and are used primarily for text manipulation. (For example by a line editor program). Use pages 6–14 to 6–16 of the *High Performance LEVEL II COBOL Operating Guide* to learn about line sequential files.

Line Sequential files are stored as CTOS SAM files.

Relative Files

Relative files are used when you want to access data randomly by specifying its position in the file. Relative files on CTOS are implemented using the Direct Access Method (DAM) relative files. DAM is discussed in more detail in the *CTOS/VM Concepts Manual.*

Do not use the *High Performance LEVEL II COBOL Operating Guide* to learn about relative files.

All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1: subsequent logical records have relative record numbers of 2, 3, 4, and so on.

The data item specified by *date–name–1* communicates a relative record number between you and the operating system.

Relative files cannot be shared for read/write access. Use Indexed Sequential files instead if you need this capability.

Indexed Sequential Files

Indexed Sequential files are implemented using the CTOS Indexed Sequential Access Method (ISAM). See the *ISAM Manual* for detailed information. Do not use the *High Performance LEVEL II COBOL Operating Guide* for information about indexed sequential files.

Note that the name you supply for an indexed sequential file in your program is the name of the data file. The default name for the associated key (i.e. index) file is produced using the extension .Ind with the root of the data file name. For example the data file **MyFile** would generate a key file name *MyFile*.**Ind**.

File and Record Level Locking

ISAM's powerful record–level and file–level locking capabilities provide secure and independently controlled file access for each user in a multi–user configuration. File and record locks permit exclusive access by one user to a file or a record within a file.

LEVEL II COBOL provides the following methods for sharing ISAM data files in a multi–user environment:

- Direct calls to the Convergent ISAM procedures. These may be linked into the Run–Time system using the facility described in Chapter 7, "Configuring COBOL." The Convergent ISAM procedures are described in the *ISAM Manual*.

- The compile–time directives "FILESHARE", "AUTOLOCK", and "WRITELOCK". These directives are described in this section, and in Section 3, "Compiling a COBOL Program".

- Extensions to the 1974 ANSI standard syntax which can be included in your COBOL programs. These extensions to 1974 ANSI COBOL and their interaction with the "FILESHARE", "AUTOLOCK", and "WRITELOCK" compiler directives are also described in this section.

Caution: *It is not recommended that you mix direct calls to ISAM procedures with COBOL FILE CONTROL and PROCEDURE DIVISION constructs for indexed sequential file operations in one program.*

Semantics of File and Record–Level Locking

A lock regulates concurrent access to a file or record, thereby maintaining data integrity when more than one user accesses the same file. A lock is used to prevent other users from accessing a record of a file that is locked for any purpose.

A file–level lock restricts access for all the records in a file, while a record–level lock only restricts access to a single record.

Locking Modes

Locking modes control the way a user can access an ISAM record or file when another user is already using it. LEVEL II COBOL supports three locking modes. The ways a file can be opened and the locking modes allowed for each case are shown in the table below:

The following are characteristics of EXCLUSIVE file–level locking:

> EXCLUSIVE locking mode prevents any user except the one which has the exclusive lock from accessing a file that is opened mode EXCLUSIVE.
>
> The whole file is locked as soon as the application executes an OPEN on a file defined to be MODE EXCLUSIVE.
>
> An application *cannot* open a file defined to be EXCLUSIVE if some other application is already accessing that file.
>
> If an application opens a file OUTPUT the locking mode is EXCLUSIVE.

The following are characteristics of AUTOMATIC record–level locking:

> Under AUTOMATIC locking, the locking action is not specified in the PROCEDURE DIVISION code.
>
> If an application opens a file I–O, it can acquire a lock on one or more records. There is a limit to the number of records that can be locked simultaneously; that maximum is a function of constraints in ISAM.
>
> If an application opens a file INPUT, it can never acquire a lock on a record.

If an application opens a file OUTPUT, the implied locking mode is EXCLUSIVE.

When a record is locked, other applications can neither read from nor write to the record.

The following are characteristics of MANUAL record–level locking:

Under MANUAL locking, the locking action, when applicable, must be specified in the procedure division code.

If an application opens a file I–O, it can acquire a lock on one or more records. There is a limit to the number of records that can be locked simultaneously; that maximum is a function of constraints in ISAM.

If an application opens a file INPUT, it can never acquire a lock on a record.

If an application opens a file OUTPUT, the implied locking mode is EXCLUSIVE.

When a record is locked other applications can neither read from nor write to the record.

Specifying the Locking Mode

Locking modes can be specified by using the extensions to the COBOL syntax which is provided in LEVEL II COBOL, or by using the compiler directives and recompiling your program. Tables 3–1 through 3–3, later in this chapter, summarize the effect of various combinations of compiler directives and extensions available in LEVEL II COBOL syntax.

Using Compiler Directives to Specify Locking Mode. The compiler directives available to specify locking mode are "FILESHARE", "AUTOLOCK", and "WRITELOCK".

The default locking in the LEVEL II COBOL that is released is:

OPEN INPUT	No locking.
OPEN I-O	Single record lock when READ is executed.
OPEN OUTPUT	Entire file locked when OPEN is executed.

These defaults were obtained by specifying "NOFILESHARE", "AUTOLOCK" and "NOWRITELOCK" when the LEVEL II COBOL compiler was built. You may change the defaults for the compiler by rebuilding it as described in the section, "Changing the Compiler Defaults," in Chapter 3, "Compiling a COBOL Program."

Please keep in mind that the default locking described in the manual *High Performance LEVEL II COBOL™ Language Reference* is EXCLUSIVE locking. EXCLUSIVE locking is obtained by specifying the "NOFILESHARE", "NOAUTOLOCK" and "NOWRITELOCK" compiler directives.

Using Syntax Extensions to Specify Locking Mode. See the *High Performance LEVEL II COBOL™ Language Reference Manual*, Chapter 7, "Indexed Input and Output", for a description of the full specification of FILE-CONTROL paragraph and PROCEDURE DIVISION syntax extensions.

You may specify a locking mode for a file within your COBOL program by using the LOCK MODE clause extension to the FILE-CONTROL entry as shown below. Note that the LOCK MODE clause is optional. If the LOCK MODE clause is left out of the FILE-CONTROL paragraph, then the locking mode is the default as specified by the compiler directives when you compile your program. If you do not specify compiler directives when you compile your program, the defaults are the directives specified when the compiler was built.

See page 7-11 of the *High Performance LEVEL II COBOL Language Reference Manual* for an example.

AUTOMATIC or Manual Record-Level Locking can both be done for single or multiple records. To obtain a lock the file must be opened I-O.

- To use AUTOMATIC single record locking you must specify LOCK MODE IS AUTOMATIC. Once read, a record remains locked until the file is closed or a another record is read. No locks are acquired when the WRITE or REWRITE commands are used.

- To use AUTOMATIC Multiple record locking you must specify LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS. Once accessed, records remain locked until the file is closed or a COMMIT or UNLOCK statement is executed. Locks are acquired if the compiler directive WRITELOCK is specified at compile time and the WRITE or REWRITE statement is executed.

- To use MANUAL single record locking you must specify LOCK MODE IS MANUAL. A lock is acquired by executing a READ WITH LOCK clause. Once a record is locked, it remains locked until the file is closed or a another record is read. No locks are acquired when a WRITE or REWRITE statement is executed.

- To use MANUAL multiple record locking you must specify LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORDS. A lock is acquired by executing a READ WITH LOCK. Locks are also acquired if the compiler directive WRITELOCK is specified at compile time and a WRITE or REWRITE statement is executed. Once a record is locked, it remains locked until the file is closed or a COMMIT or UNLOCK statement is executed.

To support a multi-user environment, extensions have been added to the PROCEDURE DIVISION. They include the additional statements COMMIT and UNLOCK, and additions to the READ statement. When writing programs for a multi-user environment, errors involving locking conditions must be taken into consideration.

To understand the function of the COMMIT and the UNLOCK statement, one must first understand the concept of a *Transaction* in a shared access environment. In Convergent ISAM a Transaction is the smallest unit of work that involves transfer and use of data (this may consist of a number of reads and writes). During a transaction the records accessed are locked to prevent anyone else from reading or changing one of those records until the data transfer is completed. At the end of a transaction all the records locked during the transaction are unlocked making them available to other users. For a full discussion of the transaction mechanism see the Convergent *ISAM Manual*, chapter 2.

In COBOL transactions are not limited to the "smallest unit of work". Instead, the default transaction in COBOL is the program as a whole. When the first COBOL organization indexed file is opened, a transaction is begun; the default is for it to be completed when the program ends. The COMMIT and UNLOCK statements are provided to enable the user to have more control of when records are released by an application so others can use them.

- The COMMIT statement signifies the successful completion of a transaction. It first unlocks all records and data sets locked by the application system, completes the transaction and then starts a new transaction for the application system.

- The UNLOCK statement releases all locks on the specified data set without ending the current transaction.

- The READ statement also affects locking.

  When "LOCK MODE IS MANUAL" is specified for a file that is opened I–O, a record is locked ONLY if the read statement has the form "READ ... WITH LOCK". The record locked will be unlocked when the next record is read.

  When "LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS" is specified for a file that is opened I–O, a record is locked ONLY if the read statement has the form "READ ... WITH KEPT LOCK". To unlock records use the COMMIT or UNLOCK statements depending on whether the user wishes to unlock all records or just records in one data set.

For details on the syntax and semantics of the COMMIT, UNLOCK, and additions to the READ statement, see Chapter 7 of the *High Performance LEVEL II COBOL Language Reference Manual*.

Table 2-1. Effect of NOFILESHARE Directive and LOCK Modes on how Records and Files are locked

OPEN Mode	SELECT... LOCK MODE IS	Command to Lock: single record	Command to Lock: multiple record	Command to Lock: whole file
INPUT				
	Not specified	none	none	none
	EXCLUSIVE	none	none	OPEN
	AUTOMATIC	none	none	none
	AUTOMATIC LOCK ON MULTIPLE	none	none	none
	MANUAL	none	none	none
	MANUAL LOCK ON MULTIPLE	none	none	none
OUTPUT				
	Not specified	none	none	OPEN
	Any locking mode	none	none	OPEN
I–O				
	Not specified	none	none	OPEN
	EXCLUSIVE	none	none	OPEN
	AUTOMATIC	READ	none	none
	AUTOMATIC LOCK ON MULTIPLE	none	READ	none
	MANUAL	READ WITH LOCK	none	none
	MANUAL LOCK ON MULTIPLE	none	READ WITH KEPT LOCK	none

The "LOCK MODE IS ..." clause is a LEVEL II COBOL extension to the 1974 ANSI standard X3.23.

Table 2–2. Effect of NOFILESHARE AUTOLOCK Directives and LOCK Modes on how Records and Files are locked

		Command to Lock		
OPEN Mode	**SELECT... LOCK MODE IS**	**single record**	**multiple record**	**whole file**
INPUT				
	Not specified	none	none	none
OUPUT				
	Not specified	none	none	OPEN
I–O				
	Not specified	**READ**	none	none

The "LOCK MODE IS ..." clause is a LEVEL II COBOL extension to the 1974 ANSI standard X3.23.

Table 2–3. Effect of NOFILESHARE WRITELOCK Directives and LOCK Modes on how Records and Files are locked

		Command to Lock		
OPEN Mode	**SELECT... LOCK MODE IS**	**single record**	**multiple record**	**whole file**
INPUT				
	Not specified	none	none	none
	EXCLUSIVE	none	none	OPEN
	AUTOMATIC	none	none	none
	AUTOMATIC LOCK ON MULTIPLE	none	none	none
	MANUAL	none	none	none
	MANUAL LOCK ON MULTIPLE	none	none	none
OUPUT				
	Not specified	none	none	OPEN
	Any locking mode	none	none	OPEN
I–O				
	Not specified	none	none	OPEN
	EXCLUSIVE	none	none	OPEN
	AUTOMATIC	READ	none	none
	AUTOMATIC LOCK ON MULTIPLE	none REWRITE	READ WRITE	none
	MANUAL	READ WITH LOCK	none	none
	MANUAL LOCK ON MULTIPLE	none	READ WITH KEPT LOCK WRITE REWRITE	none

The "LOCK MODE IS ..." clause is a LEVEL II COBOL extension to the 1974 ANSI standard X3.23.

Error Conditions While Using Locks.

When the Run-Time System detects an error, the application is notified in the file status data item that is specified in the FILE STATUS IS clause of the FILE-CONTROL paragraph. The following program fragment shows an example of how you may test for errors that can occur in a multi-user environment. The manual *High Performance LEVEL II COBOL Operating Guide* contains a description of status-2 error numbers.

```
FILE-CONTROL.
    SELECT ...
         ASSIGN TO ...
         STATUS RTS-FileStatus.
         . . .
WORKING-STORAGE SECTION.
         . . .
01  RTS-FileStatus          PIC X(02).
01  RTS-FileStat-Redefined
             REDEFINES RTS-FileStatus.
   03  RTS-FileStatus-1-X    PIC X(01).
   03  RTS-FileStatus-2      PIC 9(02)      COMP.
         . . .
PROCEDURE DIVISION.
         . . .
   READ ...
   IF RTS-FileStatus-1-X = "0" (means no exception status)
         . . .
   ELSE
   IF RTS-FileStatus-1-X = "1" (means 'end of file')
         . . .
   ELSE
   IF RTS-FileStatus-1-X = "2" (means 'invalid key')
         . . .
   ELSE
   IF RTS-FileStatus-1-X = "9" (means look at status-2)
   IF RTS-FileStatus-2 = 065 (means the file is locked)
         . . .
   ELSE
   IF RTS-FileStatus-2 = 068 (means the record is locked)
         . . .
   ELSE
   IF RTS-FileStatus-2 = 213 (means too many locks acquired)
         . . .
   ELSE
         . . .
```

See Appendix B of the *High Performance LEVEL II COBOL Operating Guide* for examples that show detection of other Run-Time errors. The sample program ByteStreamErrors.cbl on the release diskette illustrates differences between detecting Run-Time errors and CTOS errors.

3 Compiling a COBOL Program

You use the Executive command, **COBOL Compile**, to compile your COBOL source code. You can specify, in the command form, the source file name, intermediate file name, listing file name, and whether or not you want to use the ANIMATOR. Additional compiler directives, such as listing width, whether or not to print line numbers or the date in the listing, and whether or not to include the contents of files named in COPY statements can also be specified.

This chapter assumes that you know how to use an Executive command and how to use an at–file with an Executive command. If you do not, see the *Executive Reference Manual* for details.

Note that this chapter should be used as a compiler reference rather than Chapter 9 in the *High Performance LEVEL II COBOL Operating Guide.*

Using the COBOL Compile Command Form

To invoke the COBOL compiler from the Executive, type **COBOL Compile** on the command line and presss **Return** to display the command form.

Command Form

Command form changed

```
COBOL Compile
  Source file            ________________________________
  [Intermediate file]    ________________________________
  [Listing file]         ________________________________
  [For animation?]       ________________________________
  [Other directives?]    ________________________________
```

Parameter Fields

Source file

Enter the name of the source file to be compiled. The file name can be entered with or without a file extension (suffix beginning with "."). If you enter a file name without an extension and the file cannot be found, the COBOL Run-Time System adds the extension .cbl to the file name and searches for that file.

[Intermediate file]

Default: Sourcefilename.int

Enter the name of the file to which the compiler writes the intermediate code. The default is to write the intermediate code to a file name that is constructed by replacing the extension of the source file name with .int.

[Listing file]

Default: Sourcefilename.lst

Enter the name of the file to which the compiler writes the compilation listing. The default is to write the compilation listing to a file name that is constructed by replacing the extension of the source file name with .lst.

[For animation?]

Default: Yes

The default compiles the program and creates the additional files you need if you want to use the ANIMATOR later for debugging. This causes the compiler to generate extra files with extensions, such as .ANM and .D00, during compilation. These files, plus the source and intermediate files, must be present to use animation. (For details on animation, see Chapter 6, "Debugging a Program with the ANIMATOR.")

All the directives need to be checked against Micro Focus manual. We have made no changes to them.

[Other directives]

There are a number of other directives that you can specify, in the form

[No]*keyword[argument]*

where

No	Turns off the effect of the directives. No can adjoin *Keyword* or can be separated from it by one or more spaces. No is permitted where specified in "Other Compiler Directives," later in this chapter.
Keyword	Is the name of the compiler directive.
Argument	Qualifies (where applicable) the effect of *Keyword*. *Argument* must appear in parentheses or in double quotes, as follows: *"argument"* *(argument)*

Argument can adjoin *Keyword* or be separated from it by one or more spaces. An argument enclosed in double quotes can contain spaces; an argument enclosed in parentheses cannot.

You can specify the same directive more than once. The directives are processed from left to right, so the rightmost occurrence of a directive is the one that will take effect. The exception to this rule is where the rightmost occurrence of a directive is excluded by the value of a previous directive. (For details, see "Excluded Combinations of Compiler Directives," later in this chapter.)

If you have too many entries to place in the command form, you can the entries in an at-file. Then, use the at-file name in the command form prefixed by an at sign (@). (See the *Executive Reference Manual* for details on the use of at-files.)

If you have too many entries to fit on the command line you can also enter as many as fit, then type an ampersand (&). After you press Go the compiler prompts you to enter additional directives interactively. In this case the ampersand is considered a directive.

If you enter a period (.) the compiler stops and returns control to the Executive. You might want to do this if you wanted to check the compiler version number.

[NO] ALTER

Default: ALTER

Enter NO ALTER to prohibit ALTER statements within the program being compiled. This allows the compiler to operate more efficiently.

[NO] ANIM

Default: ANIM

The default compiles the program in a manner suitable for later animation. This directive takes effect over the parameter field [For animation?] described above. The default (in either case) compiles a program for animation. Entering NO ANIM for this directive, therefore, turns off the default.

[NO] AUTOLOCK

Default: NOAUTOLOCK

AUTOLOCK makes the default locking AUTOMATIC, rather then EXCLUSIVE for files opened I-O or EXTEND in a multi-user environment. See the section, "Indexed Sequential Files," in Chapter 2 for more information.

[NO BELL]
[BELL "integer"]

Default: BELL "07"

If you enter BELL "integer" (where integer is the ASCII character in decimal), that character is used to cause the bell (audio warning) to sound.

By default, this directive is on and uses the value specified when the workstation is configured to sound the bell. You can specify a different value for the program by using BELL "integer" for this directive.

Turning the directive off (NO BELL or BELL "0") causes no bell character to be set.

[NO] BRIEF

Default: NO BRIEF

Enter BRIEF to suppress the text of error messages (that is, error numbers only are produced on the listing and console). The default is NO BRIEF (unless no error message is found).

[NO] COMP

Default: NO COMP

Enter COMP to cause the compiler to produce much more compact and efficient code for certain statements involving PIC 9(2) COMP and PIC 9(4) COMP data items. This is described in Chapter 4 of the *High Performance LEVEL II COBOL Operating Guide.*

The more efficient code leads to nonstandard behavior in cases of numeric overflow. Specify this directive if you know that your statements will not lead to numeric overflow (in which case the semantics of your program will remain strictly in accord with ANSI standard while giving you the advantage of the extra efficiency), or if you mean to take advantage of the defined but nonstandard behavior on overflow.

[NO] COPYLIST ["integer"]

Default: NO COPYLIST

Enter COPYLIST to list the contents of any files named in COPY statements.

Whatever the state of this directive, the name of any copy file that is open when a page heading is written is listed as part of the heading.

The optional integer (which must be 0 or 50–99) allows the selection of particular segments with this directive. Zero means all root segments. For example,

    COPYLIST "53"

causes COPYLIST to be set in the IDENTIFICATION DIVISION and in Segment 53 but not otherwise.

    NO COPYLIST "53"

causes COPYLIST to be set in Segment 53 only.

[NO CRTWIDTH]
[CRTWIDTH "integer"]

Default: CRTWIDTH "128"

If you enter CRTWIDTH "integer," you are specifying the width of the user video display in characters (indicated by integer). This is used in Format 1 (standard ANSI) DISPLAY statements to allow you to plan points of data–items too long to fit on one physical video display line.

If you enter NO CRTWIDTH, the directive is turned off, causing Format 1 DISPLAY statements to be rejected. Not using DISPLAY statements saves memory space, because control tables are not required.

[NO DATE]
[DATE "string"]

Default: DATE

The default causes the system date to be entered into the comment entry in the DATE–COMPILED paragraph (if present). If the system date is not available, you can specify the date with DATE "string" for this directive.

This directive also causes the system date or the string you enter to be written at the top of each page of the listing.

[NO DICDEV]
[DICDEV "drive"]

Default: NO DICDEV

During a run of the compiler, a work file (normally COBOL V00) is produced that is used in building the dictionary. This directive specifies the volume and directory to which the dictionary file is to be directed.

If you enter NO DICDEV (the default) or if you do not include an entry for this directive, the dictionary file is directed to the current (default) directory.

[NO] ECHO

Default: ECHO

By default, this directive causes error lines and flags to be echoed to the console. For each error, the source line producing it, the error number, and (unless BRIEF is set) an explanatory message are displayed on the screen.

[NO] ECHOALL

Default: NO ECHOALL

Enter ECHOALL to send a full string to the video display as well as to the default listing device (or to a device specified with the LIST or PRINT directive).

[NO]ERRLIST

Default: NO ERRLIST

Use ERRLIST if you want the listing to only show the lines of COBOL that have systax errors or flags. In this case, the listing also shows the associated error messages for each line.

[NO] ERRQ

Default: NO ERRQ

If you enter ERRQ, you are asked whether to stop compilation at the error or to continue when an error occurs.

[NO]FILESHARE

Default: NO FILESHARE

Entering FILESHARE, makes default locking AUTOMATIC, rather than EXCLUSIVE, for files opened I-O in a multi-user environment. It also automatically locks records on a WRITE or REWRITE statement when the program is locking multiple records.

This directive works as though you specified the WRITELOCK and AUTOLOCK directives.

[NO FLAG]
[FLAG (LOW, L-I, H-I, HIGH, L/II, IBM)]

Default: NO FLAG

LOW, *L-I*, *H-I*, *HIGH*, *L/II*, and *IBM* are levels of General Services Administration (GSA) compiler verification flags. Verification flags are extra lines in the listing that indicate the level of a COBOL source statement and are written to the listing for all features higher than the specified level.

LOW produces flags for all features higher than the LOW level of GSA compiler certification.

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

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

HIGH produces flags for all features higher than the High level of GSA compiler certification.

L/II produces flags for only the LEVEL II COBOL extensions to standard COBOL as it is specified in the ANSI COBOL standard X3.23 1974.

IBM produces flags for only IBM-compatible non-standard COBOL.

[NO FORM]
[FORM "integer"]

Default: FORM "60"

Enter FORM "integer" (where integer is at least 3) to specify the number of lines per page of the listing.

A form feed character is always produced at the head of the listing file unless NO FORM is entered. Enter NO FORM to specify no form feed characters or page headings anywhere in the listing.

If you direct the listing to the console (using the LIST directive), form feed character interpretation depends on your video display.

[NO] IBM

Default: NO IBM

The default allows the use of IBM extensions. When the IBM directive is enabled, ID, GOBACK, PASSWORD, ENTRY, SKIP-1, SKIP-2, SKIP-3, and EJECT are all reserved words.

[NO INT]
[INT "filespec"]

Default: INT

If you enter NO INT, no intermediate code file is produced (that is, the compiler is used only as a syntax checker).

Enter INT "filespec" to specify the file to be used to hold the intermediate code produced by the compiler. If the specified file exists, it is overwritten. The compiler adds ".int" to the source file name, replacing any existing file name extension.

This directive overrides any intermediate code file name that you may have entered in the *Intermediate file* field.

[NO LIST]
[NO PRINT]
[LIST "filespec"]
[PRINT "filespec"]

Default: List file to source file name.LST.

If you enter NO LIST, no compilation listing is produced.

Enter LIST "filespec" to specify the destination of the listing file. If an existing file is specified, it is overwritten.

Enter PRINT "devicespec" to print the listing directly to a valid device. "Devicespec" can be any valid device specification, for example [VID], *[LPT]*, or *[GPSPrinterName]*.

This directive overrides any compilation listing file name that you may have entered in the *Listing file* field.

[LISTWIDTH "nnn"]
[LW "nnn"]

Default: LISTWIDTH "80"

Enter either LISTWIDTH *nnn* or LW *nnn* (where *nnn* is the number of character positions) to set the width across the listing page.

OVERRIDE "implementor–name" = "new name"

Default: Implementor–name is unchanged

Enter OVERRIDE "implementor–name" = "new name" to change the implementor–names permitted in the SPECIAL–NAMES paragraph to the new–names specified. "CONSOLE" = "CRT" is an example.

[NO] QUAL

Default: QUAL

If you enter NO QUAL, qualified data or procedure names are prohibited in the program being compiled. This allows the compiler to operate more effectively.

[NO] QUERY

Default: QUERY

If the compiler is unable to find the COPY file, the default causes the operator to be prompted to supply the file.

If you enter NO QUERY, an error is produced if a COPY file is missing.

[NO] REF

Default: NO REF

Enter REF to include four digit location addresses on the right side of the listing file. Note that a listing with location addresses may be required to identify the locations reported in COBOL Run–Time System error messages.

[NO] RESEQ

Default: NO RESEQ

Enter RESEQ to cause the compiler to produce COBOL line sequence numbers in increments of 10, starting at 10.

[NO] SEG

Default: SEG

If you enter NO SEG, the compiler ignores segmentation by treating all section numbers within the code as if they were 0. As a result, a huge program is produced.

[NO] TIME

Default: TIME

The default inserts the current system time following the system date (inserted by the DATE directive) into the source program and the listing. TIME can only be used with the DATE directive.

[NO] WRITELOCK

Default: NOWRITELOCK

If you use WRITELOCK, the WRITE and REWRITE statements acquire a record lock when the program is locking multiple records in a shared data file in a multi-user environment.

[NO] XREF

Default: NO XREF

Enter XREF to produce a cross referenced listing. The listing consists of an alphabetized list of all the data items and an associated sequence number that shows the line where the data item (marked with a #) is defined. Further sequence numbers show each time the data item is used. The listing also shows the kind of data item and the length in bytes of group items. The listing continues with a similar description of paragraph names.

NOTE: The XREF directive requires some work space on the disk. While the source code is compiling, work files are created that are later deleted. The amount of disk space required depends on the size of your source program and is related to the size of the data dictionary and the number of data items referenced. As a result, the XREF directive may need up to twice the space required by the source file. One work file (filename.5X1) is not deleted by the compiler. You can delete this file immediately following compilation.

Notes on Combining Directives

The use of certain directives implies that certain other directives are ignored, even when they are specified. Table 3–1 shows the combinations that are not permitted.

Table 3-1. Excluded Combinations of Directives

Directive	Excluded Directives
ERRLIST	COPYLIST [NO] REF RESEQ
NOLIST	COPYLIST ERRLIST [NO] FORM LIST (or LIST "Filespec") PRINT [NO] REF RESEQ ECHOALL
ANIM	[NO]CRTWIDTH

Default Compiler Directives

If no directives are specified, the compiler runs as if the following directives had been specified:

file name
ALTER
ANIM
AUTOLOCK
BELL "07"
NO BRIEF
NO COMP
NO COPYLIST
CRTWIDTH "128"
DATE
NO DICDEV
ECHO
NO ECHOALL
NO ERRQ
NO ERRLIST
NO FILESHARE
NO FLAG
FORM "60"
NO IBM
INTERMEDIATE FILE
  (filename.int)
LISTING FILE
  (filename.lst)
LISTWIDTH "80"
QUAL
QUERY
NO REF
NO RESEQ
SEG
TIME
NO WRITELOCK
NO XREF

Changing the Compiler Defaults

You may require that the default settings of the compiler directives be different from those described above. To rebuild the compiler with the defaults of your choice, use the following procedure:

1. Path to the <COBOL> directory.
2. Enter **COBOL Compile** on the Executive command line.
3. In the *Source file* field, enter = (equal sign).
4. In the *Other directives* field, enter the name of the file to contain the rebuilt compiler and the new defaults for the directives. Any directives you do not specify retain their original defaults.

For example, to rebuild the COBOL compiler with default animation, you would complete the command form as follows:

This is not supported now. [illegible] because [illegible] only modify cobol.dir file

```
COBOL Compile
  Source file            _=______________________________
  [Intermediate file]    ________________________________
  [Listing file]         ________________________________
  [For animation?]       ________________________________
  [Other directives?]    _COBOL_anim_____________________
```

Compiler Examples

The following form compiles the file MYPROG.ABC and produces a list file MYPROG.LST, which is a resequenced version of the source file. No intermediate code file is produced.

```
COBOL Compile
  Source file            _MYPROG.ABC_____________________
  [Intermediate file]    ________________________________
  [Listing file]         ________________________________
  [For animation?]       ________________________________
  [Other directives?]    _NO FORM_RESEQ_NOINIT___________
```

The following form compiles MYPROG.CBL, produces a list file MYPROG.LST with flags indicating all features higher than the LOW level of certification, echoes error messages to the video display, and produces an intermediate code file named MYPROG.INT. Note the precedence of the later directives over previous different settings of the same directives.

```
COBOL Compile
  Source file           _MYPROG.CBL________________________________
  [Intermediate file]   ___________________________________________
  [Listing file]        ___________________________________________
  [For animation?]      ___________________________________________
  [Other directives?]   _NO ECHO_FLAG(LOW) ECHO________________
```

The following form compiles MYPROG.CBL but produces no intermediate file and no list file. The compiler is used to check the program's syntax and to report errors to the console only. Note that, because NOLIST is specified, NOFORM and ERRLIST are ignored.

```
COBOL Compile
  Source file           _MYPROG.CBL________________________________
  [Intermediate file]   ___________________________________________
  [Listing file]        ___________________________________________
  [For animation?]      ___________________________________________
  [Other directives?]   _NOFORM_ERRLIST_NOINT______________________
```

The following form compiles MYPROG and inserts the system date and time into the DATE-COMPILED paragraph (if present).

```
COBOL Compile
  Source file           _MYPROG____________________________________
  [Intermediate file]   ___________________________________________
  [Listing file]        ___________________________________________
  [For animation?]      ___________________________________________
  [Other directives?]   _DATE_TIME_________________________________
```

In the following example, entries for the [Other directives?] field are placed in an at-file named DirList. "DirList" contains the following entries:

```
BRIEF NOCRTWIDTH NOECHO NOFORM
```

This example compiles PI.CBL, does not compile DISPLAY statements, does not produce error messages or page headings, suppresses echo of errors to the console, and directs the compiler listing to the console.

```
COBOL Compile
  Source file            _PI_______________________________
  [Intermediate file]    __________________________________
  [Listing file]         _VID______________________________
  [For animation?]       __________________________________
  [Other directives?]    _@DIRLIST_________________________
```

Summary Information Shown on the Screen Display

After the command form has been completed and you press **Go**, the compiler replies with:

```
 *MICRO FOCUS COBOL  Copyright(c) 1985, 1987 Micro Focus Ltd. Ref xxx-nnn
*Convergent Technologies version vvv
*LEVEL II COBOL V2.5.4 Copyright(c) 1985, 1987 Micro Focus Ltd.
```

where

xxx–nnn	Is the compiler reference number
vvv	Is the compiler version number

After all the directives have been acknowledged, the compiler opens its files and starts to compile. At this point, it displays the message:

```
*compiling filename
```

If any file fails to open correctly, the compiler displays:

```
Open fail: filename
```

The compilation is aborted, returning control to the operating system.

When compilation is complete, the compiler displays the message:

```
*Errors=nnnnn Data=nnnnn Code=nnnnn Dict=mmmmm:nnnnn/ppppp FIPS flagging off
```

where

Errors	Shows the number of errors found.
Data	Shows the size of RAM required (that is, data area of the program produced).
Code	Shows the number of bytes of code in the disk file (that is, code area of the program produced).
Dict	Where *mmmmm* shows the number of bytes used in the data dictionary. Where *nnnnn* shows the number of bytes remaining in the data dictionary. Where *ppppp* shows the total number of compiler validation flags found or OFF if the directive NO FLAG was entered or assumed.
FIPS flagging off	Shows either the number of compiler validation flags found or OFF (if the directive NOFLAG was entered or assumed).

The data division and code sizes displayed by the compiler do not include the overhead that is needed for segmented programs. The overhead is variable, but a rough guide is to allow another 60 bytes for the root segment and 30 bytes for each overlay.

NOTE: You should not use the intermediate code produced from an unsuccessful compilation. Change the source, and then compile the program again.

Listing Formats

The general layout of the list file is as follows:

```
*LEVEL II COBOL  V2.5       filename          Page nnnn

                            nn-mth-88 nn:nn*

*Options:

ssssss                      statement 1           hhhh
.                           .                     .
.                           .                     .
.                           .                     .
ssssss                      statement n           hhhh

*LEVEL II COBOL V2.5  revision n   URN xxxx/xxxxx/xxxxx

*Compiler Copyright(c) 1981, 1985 Micro Focus Ltd.  REF XXX-nnnnnnnnn

*

*Errors-nnnnn Data-nnnnn Code-nnnnn Dict-mmm:nnn/ppp FIPS flagging off
```

where

ssssss	is the sequence number of the statement.
hhhh	is a reference number (in hexadecimal) showing the address of each data name or procedure statement. This number appears in the listing only if you specified REF in the command line before compiling.
XXX–nnnnnnnnn	is the compiler reference number.

The first two lines of the title information are repeated for each page. The final line is the same as on the display.

Addresses of data names 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:

```
ssssss                        illegal statement

**eee***...                    ...*** (nnnn)**
**   compile error statement
```

where:

ssssss	is the sequence number of the line containing the error.
eee	shows the error number.

The asterisks (*) following the error number indicate the character position of the error in the preceding incorrect source line. The asterisks at the end of the line simply highlight the error line.

The number in parentheses at the end of the line shows the page of the listing on which the previous error occurred. This enables the programmer to trace back from one error to the previous error.

If errors occur, the information at the end of the list file contains a line:

*Last error on page: nnnn

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

```
ssssss    flagged feature

**level---     ...  ...---    ------(nnnn)
```

where

ssssss	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 form (when setting the lowest required flagging level):

LOW	Low level
L-1	Low-intermediate level
H-1	High-intermediate level
HIGH	High level
L/II	LEVEL II COBOL extensions
IBM	IBM-compatible extensions

The flagged feature is directly above the end of the dashes. The dashes at the end of the line highlight the flagging line.

The number in parentheses at the end of the line shows the page of the listing on which the previous flag occurred. This enables the programmer to trace back from one flag to the previous flag.

If flags occur, the information at the end of the list file contains a line:

```
*last flag on page: nnnn
```

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

Effect of Incorrect Data Items

Errors in data declarations are shown by an error message at compile time. Subsequent data declarations may then be ignored by the compiler, and can result in spurious error messages if these data items are referred to in the program.

4 Using the Native Code Generator

The Native Code Generator is used to produce a program which the Run–Time System can execute at optimum speed. The Native Code Generator produces machine instructions from the intermediate code produced by the compiler, the **COBOL Compile** command. When you first compile your program, you will want to use the Run–Time System (the **COBOL Run** command) to execute the intermediate code directly. The Native Code Generator is usually used after you have completed debugging your program.

The Native Code Generator is described in detail in Chapter 10 of the *High Performance LEVEL II COBOL Operating Guide.* The command form, directives, and customization of the defaults are described here.

Using the Native Code Generator Command Form

Command Form

To generate native code from an intermediate file, on the Executive command line type **COBOL Generate** and press **Return** to display the command form shown below.

```
COBOL Generate
  File Name    ________________________________________
  [Directives?]________________________________________
```

Parameter Fields

File Name

Enter the name of the intermediate code file for which the native code is to be generated. It is assumed that the file has the .INT extension.

[Directives]

There are a number of directives that you can specify, in the form

[No]*keyword[argument]*

where

No	Turns off the effect of the directives. No can adjoin *Keyword* or can be separated from it by one or more spaces. No is permitted where specified in "Other Compiler Directives," later in this chapter.
Keyword	Is the name of the compiler directive.
Argument	Qualifies (where applicable) the effect of *Keyword*. *Argument* must appear in parentheses or in double quotes, as follows: *"argument"* *(argument)*

Argument can adjoin *Keyword* or be separated from it by one or more spaces. An argument enclosed in double quotes can contain spaces; an argument enclosed in parentheses cannot.

You can specify the same directive more than once. The directives are processed from left to right, so the rightmost occurrence of a directive is the one that will take effect.

If you have too many entries to place in the command form, you can the entries in an at-file. Then, use the at-file name in the command form prefixed by an at sign (@). (See the *Executive Reference Manual* for details on the use of at-files.)

If you have too many entries to fit on the commandline you can also enter as many as fit, then type an ampersand (&). After you press Go the compiler prompts you to enter additional directives interactively. In this case the ampersand is considered a directive.

Directives

[NO]ASM

Default: NO ASM

Use this directive when you want an assembly listing. The listing file includes all details in mnemonic form. If NOASM is used with LIST, it includes only the Native Code Generator identity, command line details, and summary statistics.

[NO BELL]
[BELL "integer"]

Default: BELL "07"

If you enter BELL "integer" (where integer is the ASCII character in decimal), that character is used to cause the bell (audio warning) to sound.

By default, this directive is on and uses the value specified when the workstation is configured to sound the bell. You can specify a different value for the program by using BELL "integer" for this directive.

Turning the directive off (NOBELL or BELL "0") causes no bell character to be set.

[NO]CHECK

Default: CHECK

If you want to supress checking of the Run–Time limit violations, use NO CHECK.

[NO FORM]
    [FORM "integer"]

Default: FORM "60"

Enter FORM "integer" (where integer is at least 3) to specify the number of lines per page of the listing.

A form feed character is always produced at the head of the listing file unless NO FORM is entered. Enter NO FORM to specify no form feed characters or page headings anywhere in the listing.

If you direct the listing to the console (using the LIST directive), form feed character interpretation depends on your video display.

[NO]GNT ([device]external–filename])
    "[device]external–filename]"

Default: .GNT extension is added.

Use this field to specify the name of the file the native code should be written to. If you give an existing filename it will be overwritten.

If you specify NO GNT then only an assembly code listing is generated.

[NO LIST]
    [NO PRINT]
    [LIST "filespec"]
    [PRINT "filespec"]

Default: List file to source file name.LST.

If you enter NO LIST, no compilation listing is produced.

Enter LIST "filespec" to specify the destination of the listing file. If an existing file is specified, it is overwritten.

Enter PRINT "devicespec" to print the listing directly to a valid device. "Devicespec" can be any valid device specification, for example [VID], *[LPT]*, or *[GPSPrinterName]*.

This directive overrides any compilation listing file name that you may have entered in the *Listing file* field.

Customizing the Defaults

You may require different default settings for the compiler directives than those described here. To rebuild the Native Code Generator with the defaults of your choice, use the following procedure:

1. Path to the <COBOL> directory.
2. Type **COBOL Generate** on the Executive command line, and press **Return**.
3. In the *File Name* field, enter = (equal sign).
4. In the *Directives* field, enter the Generate, and the new defaults for the directives. Any directives you do not specify retain their original defaults.

```
COBOL Generate
  File Name  _=___________________________________________
  [Directives?]_Generate_ASM_No_Check_____________________
```

5 Running a COBOL Program

The COBOL Run–Time System can execute the intermediate code produced by the **COBOL Compile** command or the native code produced by the **COBOL Generate** command.

Use this chapter to learn about the Run–Time System, rather than Chapter 11 in the *High Performance LEVEL II COBOL Operating Guide*.

Using the COBOL Run Command Form

To run an intermediate or native code program from the Executive, type **COBOL Run** on the command line and press **Return** to display the command form.

Command Form

```
COBOL Run
  [File name]  ____________________________________
  [Parameters]_____________________________________
  [Switches]   ____________________________________
```

Parameter Fields

[File Name]

Enter the name of the intermediate or native code file to be run, with or without a file name extension. If you do not enter a file name extension, the Run–Time System searches for a file to load and execute as follows:

1. The Run–Time System looks for a file with the name that you specify.

2. If the Run–Time System cannot find this file, it adds the extension .gnt to the file name and searches for the native code file.

3. If the Run–Time System cannot find the native code file, it adds the extension .int to the file name and searches for the intermediate code file.

4. If the Run–Time System cannot find the intermediate code file, it reports an error.

[Parameters]

Enter any parameters required by the program here. Parameters can be read in on the console file device [KBD] as well as from the command form. There can be as many parameters as the programmer requires, and they can be text or numbers in any format, provided they do not exceed 80 characters.

There are two methods of accessing the program parameters: READ or ACCEPT.

You may declare [KBD] as a sequential or line sequential file. (Line sequential is preferable, since fixed length records are not then expected.) This file may then be accessed by a READ; READ returns the command line program parameters until the command line is exhausted. Subsequent READs expect console input. If there are no command line parameters, the first READ returns spaces.

The first ACCEPT FROM CONSOLE statement in a program suite returns the program parameters from the command line. The CRTWIDTH directive (described in Chapter 3) affects the behavior of the ACCEPT statement; ACCEPT is compiled as a sequence of READs, each of CRTWIDTH characters, sufficient to fill the data item specified.

Subsequent ACCEPT statements expect console input.

[Switches]

COBOL enables you to control events in a program at run time depending on whether or not programmable switches are set by the operator. The operator sets these switches at run time by use of the Switches parameter. Switches that have been set at run time remain set when COBOL-called modules are processed.

The general format of the Switches parameter is

```
{±}S [,{±} S]...
```

where

[]	Shows an optional item.
{}	Shows a choice.
+ or −	Set the switch on or off, respectively. The default is that all switches are off initially.
...	Shows that the preceding options enclosed in the outermost brackets can be repeated.
S	Is a digit in the range 0 to 7 corresponding to the programmable switches you include in your program. You can specify them in any order, and the last setting of the switch is the one that is accepted. Initially, all these switches are off.
F	Flags illegal numeric comparisons in intermediate code with run−time error 163.

Run–Time Switches

The Run–Time switches that you can specify are

- the ANSI COBOL Debug switch
- the null switch
- the tab switch

The ANSI COBOL Debug Switch

You may include the ANSI COBOL debug switch as a parameter to invoke the standard ANSI COBOL Debug feature.

The ANSI COBOL Debug switch has the form:

```
±D
```

The switch is set to on with +D; –D sets it off. The default is off.

The Null Switch

You may need to use the null switch if the line sequential data file you want to work on was created with another version of COBOL. The format of the data files may be incompatible with this version of COBOL, so the null switch parameter is provided to allow you to make the files compatible.

The null switch has the form:

```
±N
```

The switch is set to on with +N; –N sets it to off. The default is on. This switch affects only the physical storage of data in a line sequential file. Records written with one setting and read back with the same setting are returned in exactly the same way.

NOTE: If you attempt to input a file with a +N setting where it was output with a –N setting, or vice versa, unpredictable results will occur.

When the null switch is set to –N, data records in a line sequential file must contain only ASCII text characters (those with a value greater than 1Bh, with the exception of the TAB character). However, if the null switch is set to +N, any character is allowed in the data record of a line sequential file.

The physical storage of data records is handled as follows:

With the null switch off:

- On output to disk, trailing spaces are removed, and a record terminator (0D0Ah) is appended to terminate the record. Printer control characters may also be written to disk.

- The file is read as data records with each record terminated by a record terminator. If the data in the record contains either Dh or 0Ah, results can be unpredictable. Tab characters (09h) are expanded to spaces; otherwise, the data record is returned intact.

With the null switch on:

- Before each record is written, it is examined, and all characters with values less than 1Bh are written with a preceding null character (00h) to form 2–byte pairs.

  On output to disk, trailing spaces are removed, and a record terminator is appended to terminate the record.

- The file is read as records (some of which may contain 2–byte pairs where the first byte is null) terminated by a record terminator. The leading null of each 2–byte pair is removed to give the final data record. Tab characters (09h) that do not form part of a 2–byte pair, are expanded.

To enable or disable null handling for all line sequential files in a run unit, a null switch is required. You can, however, change null handling for individual files with subprogram calls.

The Tab Switch

You may also include the tab switch as a parameter to enable or disable insertion of tab characters in a line sequential file. A tab switch is provided to allow compatibility with other versions of COBOL.

The tab switch has the form:

+T

The switch is set to on with +T; −T sets it to off. The default is off.

When −T is specified, all spaces in a line sequential file are written to disk as space characters.

When +T is specified, multiple spaces before a tab stop position are written as a tab character on output to disk.

On input, all tab characters are expanded to spaces to the next tab position, regardless of the setting of the run−time switches. Tab positions are every 8 character positions (for example, 9, 17, 25, and so on) and cannot be changed.

To enable or disable tab insertion in line sequential files, a tab switch is required. You can, however, change tab insertion for individual files with subprogram calls.

COBOL Run Examples

The following example loads the program MYPROG from the intermediate code file produced by the compiler and passes the application program parameters 1 and 2 to MYPROG. The parameters then are accessible to the ACCEPT statement.

```
COBOL Run
 [File name]     _MYPROG______________________________
 [Parameters]    _1_2_________________________________
 [Switches]      _____________________________________
 _
```

The following example loads the program SWITCHEDPROG from the intermediate code file produced by the compiler with programmable switches 3 and 6 on and 2 off. Note that the last setting of switch 6 is accepted. Switches 1,4,5, and 7 are off by default.

```
COBOL Run
 [File name]     _SWITCHEDPROG________________________
 [Parameters]    _____________________________________
 [Switches]      _−2_+3_−6_+6_________________________
```

The following example loads the program SWITCHEDPROG from the intermediate code file on volume [MYVOL] and directory <MYDIR> with the ANSI COBOL DEBUG module invoked.

```
COBOL Run
  [File name]     _[MYVOL]<MYDIR>SWITCHEDPROG____________
  [Parameters]    _______________________________________
  [Switches]      _+D____________________________________
```

The following example loads the program SWITCHEDPROG from the current directory and specifies that, in line sequential files, all characters less than 1Bh, except for Tab and the file control characters, are treated as data. Null is not inserted before data characters less than 1Bh on output.

```
COBOL Run
  [File name]     _SWITCHEDPROG__________________________
  [Parameters]    _______________________________________
  [Switches]      _-N____________________________________
```

The following example loads the file SWITCHEDPROG.INT from the current directory, and in line sequential files, outputs multiple spaces as TAB characters.

```
COBOL Run
  [File name]     _SWITCHEDPROG__________________________
  [Parameters]    _______________________________________
  [Switches]      _______________________________________
```

Running COBOL on an SRP

If you have an SRP as a master, you generally run COBOL at your local workstation through the Executive. Sometimes, however, you may want a program to execute at the SRP itself.

Your Executive manual describes the concepts behind executing a command at the SRP. Check it and your release notice for background information.

To run a command at the SRP you must use the SRP Command Line Interpreter (CLI).

With LEVEL II COBOL using the CLI commands is slightly differently than most other commands, because all the LEVEL II COBOL commands (**COBOL Compile, COBOL Run, COBOL Generate,** and **COBOL Animate**) are invoked using the same run file. This makes it necessary to enter the command name in the CLI Run statement, as well as the run file name.

To run an intermediate or native code program from the CLI, use the RUN statement

```
$RUN ([Sys]<Sys>COBOL.RUN,'COBOL Run'),Filename,Parameters,Switches
```

where, *Filename, Parameters,* and *Switches* correspond to the entries you would use when invoking **COBOL Run** from the Executive command line.

Subparameter must be delimited by placing them in single quotes. Note that if *Filename, Parameters,* or *Switches* has no entry you must still leave a placeholder for that parameter. For example

```
$RUN ([Sys]<Sys>COBOL.RUN,'COBOL Run'),MyProg, ,'+D''-N'
```

Setting up a Customized Command for your Program

You can create a customized Executive command to invoke your LEVEL II COBOL program in a similar manner to the way you would create any other Executive command. When you do so, users can just type *YourProgramName* on the Executive command line to invoke your program, rather than having to type **COBOL Run** and enter the run file name.

Your Executive manual goes into detail to describe how to create a new command using the **Command File Editor** or the **New Command** command. If you are unfamiliar with this process you should probably read that description before reading further. For simplicity, this manual describes only the use of the **New Command** command.

To create a customized command to execute the file *YourProgram*.Int on the Executive command line type **New Command** and press **Return.** Complete the command form as shown, then press **Go** to create the new command, *YourProgram*.

```
New Command
  Command name          _YourProgram_______________________
  Run file              _[Sys]<Sys>COBOL.Run________________
  [Field names]         _'Your Field Name 1'_'Your Field Name 2'_____
  [Description]         ___________________________________
  [Overwrite ok?]       ___________________________________
RØ  [Case (default '00']  ___________________________________
  [Command file]        ___________________________________
```

Notice that in the *Command name* field you enter the intermediate or native code file name, but *do not* include the suffix .Int or .Gnt.

Each of the field names you enter in the *Field names* field becomes a separate field of the Executive command you create. Your program can read the data the user enters in these fields by using LINE SEQUENTIAL input from the keyboard [KBD]. See "Reading the Fields of a Custom Command Form," below.

To invoke the new command, the user types *YourProgram* on the command line, presses **Return** to display the command form, completes the command form, then presses **Go** to execute the command.

For your new command to work properly, all the intermediate code or native code files that make up the program must be in the system directory, [Sys]<Sys>. If the program uses segmentation, do not forget to move the files containing independent segments and inter-segment reference information. These intermediate code files have extensions .I*xx* and .Isr, where *xx* represents a segment number. The native code files have extensions .G*xx* and .Gsr. If the program calls other modules, move these into the system directory as well.

Reading the Fields of a Customized Command Form

A COBOL program reads the data entered in the fields of a custom command form by reading sequential records of a file that is opened by using the filename [KBD], with LINE SEQUENTIAL organization, in INPUT mode.

The first record returned corresponds to the data entered in the first field. The second record corresponds to the second field, and so on. If the field contains no data, the corresponding record is empty. It contains only spaces.

When all the fields have been read, subsequent read operations will take input from the keyboard.

If the form for the command **YourProgram** as shown below is used

```
YourProgram
  Your Field Name 1       _First Entry_______________________________
  Your Field Name 2       _Second Entry______________________________
```

the following program example shows how to read from the form.

Program fragment from YourProgram:

```
    ...

   FILE-CONTROL.
       SELECT FieldNames
           ASSIGN "[KBD]"
           ORGANIZATION LINE SEQUENTIAL.
  *
   DATA DIVISION.
  *
   FILE SECTION.
   FD  FieldNames.
   01  FieldNames-Buffer      PIC X(80).
  *
   WORKING-STORAGE SECTION.
   01  Display-Names.
       03  Name-1             PIC X(20).
       03  Name-2             PIC X(20).
  *
   PROCEDURE DIVISION.
   p1.
  *  Read the two fields of the Update command FieldNames.
       OPEN INPUT FieldNames.
       READ FieldNames.
       MOVE FieldNames-Buffer TO Name-1.
       READ FieldNames.
       MOVE FieldNames-Buffer TO Name-2.
       DISPLAY Display-Names.
       STOP RUN.

  *end*of*program*
```

Using COBOL Communications

LEVEL II COBOL supports ANSI Standard COBOL communications facilities. This section describes the CTOS commands you use to create, update, and use an initialization file for the communications facilities. Operation of the communications facilities is described in Chapter 7 of the *High Performance LEVEL II COBOL Operating Guide*.

Note that the *High Performance LEVEL II COBOL Operating Guide* mentions that for MS–DOS and CPM–86 the *Physical Device* field does not apply. It is however used for CTOS and refers to the communications channel of the workstation. [COMM]A should be specified for channel A and Console Number 1. [COMM]B should be specified for channel B and Console Number 2.

The *High Performance LEVEL II COBOL Operating Guide* also mentions about device drivers that are required for MS–DOS. These are not required for CTOS.

Creating an Initialization File

To create an initialization file, on the Executive command line enter **COBOL Run**, and press **Return**. Complete the command form as shown below, then press **Go**.

```
COBOL Run
 [File name]    _[Sys]<COBOL>COMMS________________________
 [Parameters    __________________________________________
 [Switches]     __________________________________________
```

Operation instructions are in the *High Performance LEVEL II COBOL Operating Guide*

Note that the *High Performance LEVEL II COBOL Operating Guide* mentions that for MS–DOS and CPM–86 the *Physical Device* field does not apply. It is however used for CTOS and refers to the communications channel of the workstation. [COMM]A should be specified for channel A and Console Number 1. [COMM]B should be specified for channel B and Console Number 2.

Updating an Initialization File

You can update an initialization file in a similar manner to the way you create one. Complete the **COBOL Run** form in the same way as you do to create an initialization file, then press **Go**.

```
COBOL Run
  [File name]    _[Sys]<COBOL>COMMS______________________
  [Parameters    ________________________________________
  [Switches]     ________________________________________
```

At the first prompt, press **U**, then change parameters as required.

Using an Initialization File

When you have created an intialization file, this file can be used to define symbolic consoles and queues to the Message Control System before the program is run. The command to use an already existing file is shown below.

```
COBOL Message Control System
  [File name]    _InitializationFileName_________________
  [Parameters    ________________________________________
  [Switches]     ________________________________________
```

The default extension .MCS is often used for initialization file names, but is not required.

A lot of Animator menu options have changed.

6 Debugging a Program with the ANIMATOR

The LEVEL II COBOL ANIMATOR is a COBOL debugging tool that lets you watch your program execute. With the ANIMATOR, you can stop the execution of the program dynamically and modify the values of data items during execution. The ANIMATOR runs your compiled intermediate code.

Preparing to Run the ANIMATOR

Before you use the LEVEL II COBOL ANIMATOR, you must have the COBOL compiler and the ANIMATOR installed as described in the *LEVEL II COBOL Release Notice.*

The ANIMATOR files and the COBOL compiler files are all assumed to be in the directory [Sys]<COBOL> The directory of the source file you want to animate is assumed to be the same one you are pathed to, unless otherwise specified.

You should compile the program on the same disk as the one where you install the ANIMATOR, and you must use the ANIM directive. This causes the compiler to create additional files with the extensions .ANM, .ACP, .D00, and .Dnn (where n is the segment number of a segmented program). The ANIMATOR uses these files during animation.

Additional compiler directives you may want to set if you intend to use the ANIMATOR are RESEQ, REF, and COPYLIST. These supply you with more information to use during the animation session. For details about compiler directives, see Chapter 3, "Compiling a COBOL Program."

Any library files referenced by COPY statements in your program should also be present when you run the ANIMATOR.

Caution: *The ANIMATOR does not support screen displays that contain highlighting. Do not use the Amimator to run a program that highlights the user screen.*

Using the Animate Command Form

Command Form

To invoke the ANIMATOR from the Executive, type **COBOL Animate** on the Executive command line and press **Return** to display the command form.

```
COBOL Animate
  [File name]  ______________________________________
  [Parameters]______________________________________
  [Switches]   ______________________________________
```

Complete the parameter fields is described below. After you complete the form, press **Go.** The program is displayed on the top portion of the screen, and the cursor is shown on the first executable statement in the Procedure Division. ANIMATOR commands are displayed on a menu on the bottom of the screen below the dotted line.

Parameter Fields

File Name

Enter the name of the intermediate file (the file to which the compiler wrote the intermediate code).

[Parameters]

The parameters the program needs to run must be entered interactively after the ANIMATOR is invoked. Leave this field blank.

[Switches]

Enter switch parameters as required. For details, see Chapter 5, "Running a COBOL Program."

ANIMATOR Commands

When you use the ANIMATOR, the program you are executing is displayed on the top portion of the screen. ANIMATOR commands are displayed on a menu on the bottom of the screen below the dotted line.

You select a particular ANIMATOR command by pressing a single key, corresponding to the upper case letter in the command name as it appears on the video display menu. For example, if you press **X**, you select the eXchange command. You also can press **x,** since upper case and lower case letters are equivalent for this purpose.

If your keyboard has function keys, the ANIMATOR is configured so these keys correspond to some of the most commonly used ANIMATOR commands. You can therefore select these commands by pressing either the appropriate letter or the function key.

Several commands contain menus of subcommands. You can select these subcommands in the same way (that is, by function key or letter).

The rest of this section describes the ANIMATOR commands in the order that they appear on the video display.

Function Key/Letter Commands

The ANIMATOR command menu allows you to enter the following commands by pressing a function key or a letter:

F1 or H = help. Displays information about the ANIMATOR commands currently shown on the video display.

F2 or V – view. Displays the user screen. The user screen is what you see if you run the program without animation. Press any key to return to the ANIMATOR screen.

F3 or A = align. Aligns the display so that the line marked by the cursor becomes the third line of the display.

F4 or X = exchange. Allows you to move the cursor into the other part of the screen if you have a split screen. (For details on split screen display, see the Text command in "Letter Commands," later in this section.)

F5 or W = where. Moves the cursor to the statement that the ANIMATOR will execute next.

F6 or K = look–up. Prompts you to enter a line number. Then, that line becomes the third line of the display.

F9 or < = word–left. Moves the cursor to the previous word in the source code.

F10 or > = word–right. Moves the cursor to the next word in the source code.

Cancel. Has three different functions, depending on when it is pressed. When the main menu is displayed, you exit from the ANIMATOR. When any other menu is displayed, you return to the main menu. When a program is executing, the program immediately stops, and you return to the main menu.

Letter Commands

The rest of the ANIMATOR commands in the main menu can be entered only by letter. The letter you enter is the uppercase letter in each command described below. Some of the commands have subcommands, which are described along with their associated commands.

S(Step). Executes the statement marked by the cursor and then stops.

G (Go). Executes and animates the program. This means that the cursor moves from statement to statement as each one is executed. The Go command has the following subcommands:

> **0–9 = speed.** Sets the speed of animation where 0 is the slowest and 9, the fastest. You can enter the speed before you press G, or at any time during animation.
>
> **Z(Zoom).** Executes at the fastest speed, but does not animate. Press the **Cancel** key to interrupt execution and return to the main menu.
>
> The Go subcommands also offer function keys F1 to F4 and Cancel. If you choose to use the letter equivalents of F1 to F4, you will have to press two keys (:/) followed by the appropriate letter key.

Z(Zoom). Executes at the fastest speed, but does not animate. Press the Cancel key to interrupt execution and return to the main menu.

I (next–If). Executes, but does not animate, up to the next IF statement. Execution then stops.

P(Perform). Executes PERFORM paragraphs and CALLs within an animated program but does not animate them. This allows you to execute PERFORMs and CALLs that you know are free of bugs at the fastest speed. The Perform command has the following subcommands:

> **S(Step).** If the next statement to be executed is a PERFORM or CALL, this command executes the instructions in the PERFORMed paragraph or CALLed subprogram but does not animate the instructions. Otherwise this command acts like an ordinary Step command.
>
> **E(Exit).** When execution passes into a PERFORM paragraph, this command executes the rest of the statements in the paragraph but does not animate them. The cursor stops on the statement after the PERFORM statement.
>
> The Perform subcommands also offer function keys F1 to F10 and Cancel.

R(Reset). Allows you to specify which statement you want the ANIMATOR to execute next. The Reset command has the following subcommands:

> **C(Cursor–position).** Specifies the statement marked by the cursor to be the one the ANIMATOR executes next.
>
> **N(Next).** Moves the cursor to the next statement in the source code.
>
> **S(Start).** Moves the cursor to the first statement of the PROCEDURE DIVISION for execution to start again.
>
> **Q(Quit–perform).** Leaves the PERFORM paragraph currently being executed without executing any more statements. The cursor moves to the statement after the most recent PERFORM statement.
>
> The Reset subcommands also offer function keys F1 to F10 and Cancel.

B(Break). Allows you to set up to four breakpoints in the source program, so that when the program executes either with or without animation, execution stops when it reaches a breakpoint. The Break command has the following subcommands:

S(Set). Places a breakpoint on the statement pointed to by the cursor.

U(Unset). Removes the breakpoint at the statement pointed to by the cursor.

C(Cancel all). Removes all breakpoints.

E(Examine). Shows where you have set breakpoints. The cursor moves to the next breakpoint each time you enter E. If you press E after you have seen the last breakpoint, the first breakpoint is redisplayed.

I(If). Sets a breakpoint with an associated condition. The ANIMATOR prompts you to enter the condition, so that execution stops at this point only if the condition is fulfilled. The condition must comply with COBOL syntax rules, and you can set only one conditional breakpoint. The If command has the following subcommands:

F1 or /H = help. Displays information on conditional breakpoints.

F2 or /C = clear. Clears the current condition to spaces.

Cancel. Returns you to the main menu without setting a breakpoint.

The Break subcommands also offer function keys F1 to F10 and Cancel.

E(Env). Allows you to specify the environment of a program by setting parameters to control the way a program is animated, displayed, and executed. The Environment command has the following subcommands:

P(Program break). Specifies that a particular program within a suite is animated, regardless of whether the suite as a whole is executed with animation. The Program break command has the following subcommands:

T(This). Specifies the current program as the one to be animated. Execution stops when this program is reached.

S(Select). Prompts you to enter the name of the program you want animated. The Select command has the following subcommands:

F1 or /H = help. Displays information about the Select command.

F2 or /C = Clear. Clears the name currently displayed to spaces.

Cancel. Returns you to the main menu without naming a program.

C(Cancel). Cancels the command and allows execution of the whole suite without animation.

The Program break subcommands also offer function keys F1 to F10 and Cancel.

T(Threshold level). Allows you to set the level of PERFORM or CALL below which the PERFORMed paragraph or CALLed subprogram is not animated but executed as one instruction. The screen divider shows the current PERFORM level, and the Threshold level is set at this level until you change it. The Threshold level command has the following subcommands:

S(Set). Sets the level to the current level of PERFORM or CALL.

U (Unset). Animates all levels of PERFORM or CALL.

The Threshold level subcommands also offer function keys F1 to F10 and Cancel.

U(Until). Sets a general condition, which if it ever becomes true, causes the program to stop. This condition is not associated with a specific statement as the conditional breakpoint is. The Until command has the following subcommands:

S(Set). Allows you to enter the condition. The ANIMATOR prompts you to enter the condition, which must follow COBOL syntax. The Set command has the following subcommands:

F1 or /H = help. Displays information about setting a conditional breakpoint.

F2 or /C = clear. Clears the condition currently displayed to spaces.

Cancel. Returns you to the main menu without specifying a condition.

U(Unset)..Cancels any condition you have already set.

E(Examine). Displays the condition you have already set.

The Until subcommands also offer function keys F1 to F10 and Cancel.

Q(Query). Displays and allows you to change the contents of a data item, which you select either with the cursor or by entering the name of the data item. The Query command has the following subcommands:

C(Cursor name). Displays the contents of the data item marked by the cursor. You may need to move the cursor before you enter this command, so that the cursor is at the first letter of the data name.

E(Enter name). Displays the contents of the data item whose name you enter. The ANIMATOR prompts you to enter the name. The Enter name command has the following subcommands:

F1 or /H = help. Displays information about entering the name of the data item.

F2 or /C = clear. Clears the contents of the data item currently displayed to spaces.

Cancel. Returns you to the main menu without displaying the contents of the data item.

The Query subcommands also offer function keys F1 to F10 and Cancel.

R(Repeat). Displays the same data item as the ANIMATOR displayed the time before.

M(Monitor–off). Stops the constant display of the contents of the data item.

When you have selected a data item, the ANIMATOR displays another menu:

F1 or /H = help. Displays information about displaying data.

F2 or /C = clear. Clears the contents of the data item to spaces.

F3 or X – text or hex. Displays the contents of the data item as text or in hexadecimal format. The F3 or /X key allows you to switch between the two.

F4 or /M – monitor. Constantly displays the contents of the data item as the program executes.

F5 or /L = left. Allows you to move left in the data item if it is larger than 80 bytes of text or 16 bytes in hexadecimal format.

F6 or /R = right. Allows you to move right in the data item if it is larger than 80 bytes of text or 16 bytes in hexadecimal format.

F7 or /U = up–table. Allows you to move to the previous entry if the data item is subscripted or indexed.

F8 or /D = down table. Allows you to move to the next entry if the data item is subscripted or indexed.

F9 or /D = hex/ASCII. (Hexadecimal format only) moves the cursor between the hexadecimal and ASCII displays of the contents of the data item. You can change the contents of the data item simply by entering the new values.

Cancel. Returns you to the main menu without changing the data.

Depending on the type of data item you are displaying, not all of these options are applicable; only the appropriate options will be displayed.

F(Find). Allows you to search forward from the cursor position for a specified string. The ANIMATOR prompts you to enter the string, which you must enter exactly as it appears in the source code. The Find command has the following subcommands:

> **F1 or /H = help.** Displays information about entering a string.
>
> **F2 or /C = clear.** Clears the string currently displayed to spaces.
>
> **Cancel.** Returns you to the main menu without searching for a string.

I(Locate). Finds the declaration of a data item, file name, or procedure. The Locate command has the following subcommands:

C(Cursor name). Finds the declaration of the item pointed to by the cursor.

E(Enter name). Finds the declaration of the item whose name you enter. The Enter name command has the following subcommands:

F1 or /H = help. Displays information about locating the declaration.

F2 or /C = clear. Clears the data name currently displayed to spaces.

Cancel. Returns you to the main menu without finding a declaration.

The Locate subcommands also offer function keys F1 to F10 and Cancel.

T(Text). Allows you to control the format of the screen. The Text command has the following subcommands:

S(Split). Divides the screen at the current cursor position. You can enter commands to operate on the part of the screen containing the cursor, and, by using the F4 or X key, you can move the cursor between the two parts of the screen.

J(Join). Returns to an undivided display after a Split command.

R(Refresh). Repaints the screen display. When a program contains a Format 1 DISPLAY, the screen display may be corrupted.

The Text subcommands also offer function keys F1 to F10 and Cancel.

D(Do). Allows you to enter a COBOL statement that will be executed immediately. The statement has immediate effect on your program, but the statement is not permanently included in the source code. The ANIMATOR prompts you to enter the COBOL statement you want executed. The Do command has the following subcommands:

F1 or /H = help. Displays information about immediately executing a COBOL statement.

F2 or /C = clear. Clears the COBOL statement currently displayed to spaces.

Cancel. Returns you to the main menu without executing a COBOL statement.

0–9 = speed. Sets the speed at which your program is animated. The slowest speed is 0; the highest, 9. You can set the speed by entering a number before you enter the Go command, or at any time during animation.

Cursor Positioning Keys

In addition to the function key and letter commands already described, you have other keys that help you move around the source code. These keys are described below:

Left arrow	Moves the cursor one character to the left.
Right arrow	Moves the cursor one character to the right.
Up arrow	Moves the cursor up one line.
Down arrow	Moves the cursor down one line.
Code–Prev Page	Displays the previous 200 lines of source code.
Code–Next Page	Displays the next 200 lines of source code.
Next Page	Displays the next 20 lines of source code.
Prev Page	Displays the previous 20 lines of source code.
Tab	Moves the cursor to the next tab position right.
Shift–Return	Moves the cursor to the eighth character position of the next line.

7 Configuring COBOL

LEVEL II COBOL can be configured to include library procedures and other non–COBOL procedures, as is mentioned in Chapter 2. The CRT display can also be specially configured. Both of these types of configuration are described in this chapter.

Linking Procedures into the COBOL Run–Time System

LEVEL II COBOL is already set up so that you can easily link procedures into your program from the Forms, ISAM, Graphics, Sort/Merge, or CTOS.lib libraries. You can also link any procedures or libraries you have created yourself into the Run–Time System if they have been compiled using the medium model of computation.

Before you can link any procedures into the Run–Time System you must have your workstation environment set up for the link. You will need to have Standard Software, Graphics.lib, ISAM.lib, Forms.lib, and SortMerge.lib all present, whether or not you want to link procedures from these libraries into the Run–Time. See the *LEVEL II COBOL Release Notice* for detailed information on these and other required files.

If you want to link your own procedures into the Run–Time you must first edit COBOLGen.Asm to include them. Use the Editor to open the file [Sys]<MF–RTS>COBOLGen.Asm. Instructions for changing the file are included within it. Then go on to assemble the file.

If you only want to link standard procedures into the Run–Time you can go directly to assembling COBOLGen.Asm without editing.

Assembling LEVEL II COBOL

In the Executive, path to the [Sys]<MF-RTS> directory.

On the Executive command line, type **Assemble** and press **Return** to display the Assembler command form. Complete the form as shown below. (See the *Assembly Language Manual* for details on the Assembler.)

```
Assemble
 File                              _COBOLGen.Asm________________
 [Errors only?]                    _____________________________
 [GenOnly, NoGen, or Gen?]         _____________________________
 [Object file]                     _____________________________
 [Error file]                      _____________________________
 [List on pass 1?]                 _____________________________
 [:f0:]                            _____________________________
 [:f1: (Default [Sys]<Edf>]        _____________________________
```

During assembly, the assembler asks questions of this type:

```
Will you be calling the Graphics Manager?

Will you be calling FORMS (y/n)?

Will you be calling OpenFile or CloseFile  (y/n)?
```

If you answer y (for yes) to a question, the assembler creates an entry for each procedure in the corresponding software package. To answer no to a question, type **n RETURN** or just **RETURN**.

The procedure names and interfaces associated with each software package are also part of the file, COBOLGen.asm.

After you have answered all questions asked by the assembler, the video displays the following message:

```
.....configuring COBOL
```

Then, you are instructed to relink the COBOL Run-Time System to use the procedures for Forms, Graphics, and so on.

Linking COBOL .Run

To relink the COBOL Run–Time System to produce COBOL.run, use the submit file LinkCOBOL.sub. In the Executive, complete the **Submit** command, as shown below:

```
Submit
  File list              __LinkCOBOL.sub______________________
  [Parameters]           _____________________________________
  [Force Expansion?]     _____________________________________
  [Show Expansion?]      _____________________________________
```

LinkCOBOL.sub displays the Linker command form and fills in the fields below:

```
Link
  Object modules            COBOLHead.obj...COBOLTail.obj_obj.damutl.obj____
  Run file                  COBOL.run_______________________________________
  [Map file]                COBOL.map_______________________________________
  [Publics]                 ________________________________________________
  [Line numbers?]           ________________________________________________
  [Stack size]              ________________________________________________
  [Max memory array size]   ________________________________________________
  [Min memory array size]   ________________________________________________
  [System build?]           ________________________________________________
  [Version]                 ________________________________________________
  [Libraries]               [Sys]<Sys>SortMerge.lib...[Sys]<Sys>Graphics.lib__
  [DS allocation?]          ________________________________________________
  [Symbol file]             COBOL.sym_______________________________________
```

When the form is filled in, press **GO**.

Updating the [Sys]<Sys>Directory

To use the COBOL Run–Time System, copy your new version of COBOL.run to [Sys]<Sys>COBOL.run.

In place of 'CONFIG' utility we have a 'CONFIG' Submenu on 'Cobol Toolset' If needed working of that submenu could be covered here

CRT Configuration

If your application has special keyboard or CRT requirements for ACCEPT and DISPLAY statements, or if you want to configure COBOL to interpret certain keys differently than the LEVEL II COBOL defaults, you can tailor Run–Time System CRT handling capabilities to your workstation or your application requirements.

You make the changes to the COBOL ACCEPT and DISPLAY interface through a CRT configuration utility program called CONFIG. The CONFIG utility allows you to inspect and alter the CRT description in your ACCEPT/DISPLAY Module (ADIS). The Run–Time System uses the ADIS module as a reference for how to interpret keystrokes.

LEVEL II COBOL is released with a Run–Time System designed to work on your workstation. There is no need to run CONFIG unless you require the values within the CRT description to be changed for either a different set of CRT values or different application requirements such as different tabbing positions for ACCEPT input, or different keys to make the cursor move from field to field.

When you use CONFIG you can:

- Review a list of the currently defined ACCEPT/DISPLAY values of your ACCEPT/DISPLAY Module (ADIS).
- Tailor your ACCEPT/DISPLAY Module (ADIS), so that the interactive features of LEVEL II COBOL can be used with your CRT and your application.
- Form one or more libraries of CRT definitions that may be used for your ACCEPT/DISPLAY Module. Note that to use the library feature of the CONFIG utility, ISAM must be installed and the dataset CONFIG.TRM and its index set CONFIG.Ind must be reorganized using the ISAM reorganize command.

The subsections that follow describe

- the values that can be specified within a CRT description.
- the default values
- the procedure for running CONFIG

CRT Description

The CONFIG Utility requests information required for building an ACCEPT/DISPLAY Module suitable for your CRT and your applications. You can provide values that specify general configuration options, physical characteristics, console output, and function key configuration.

General Configuration Options

General configuration options refer to the type of CRT addressing. Addressing choices are the following:

- Direct to a particular screen position. If you make this choice you can further specify:
  - screen coordinate format (Absolute binary value, 1-digit, 2-digit, or 3-digit ASCII)
  - whether the first coordinate is line or column
  - increment to first and second coordinate (can be zero)
  - whether there are leading, intermediate and trailing control characters, and if so, their values
- Step addressing by single cursor moves. If you make this choice you are also requested to specify key or hexadecimal sequence of characters that move the cursor a step left, step right, step up, and step down.
- User subroutine addressing by call to routine. If you make this choice you are futher requested to specify:
  - decimal number (in the range 0–255) that is used to call the routine
  - up to three bytes of parameter information

Screen Display Options

You can also specify a number of CRT characteristics that govern screen display characteristics. They are the following:

- Number of columns and rows. Number of columns and rows is entered as a decimal. You must also specify whether it is safe to use the last position on the screen.

- Echo, which specifies whether you have automatic echo (when the CRT itself echoes characters) or whether it is the microcomputer that echoes characters. Specifying CRT echo requires that you also specify whether all characters (including control characters) are echoed.

- Wraparound, which can be forward only, backward only, or both. You can also specify whether highlighting is affected by wraparound.

  Forward wraparound means that the cursor moves to a new line if a character is written in the last column of the line

  Backward wraparound means that the cursor moves to the last position of the previous line if you backspace over the first column of a line

- Highlighting can be *Not Implemented*, or you can specify how it is implemented.

  A 2 digit character mask that you specify can be used to implement highlighting

  Start and stop control strings can be used to implement highlighting. If you choose this option you must specify the hexadecimal control character start sequence, the hexadecimal control character stop sequence, and the number of screen character positions used in the control string.

Console Output Options

Console output configuration parameters allow you to specify the key or hexadecimal sequences that clear the screen and sound the bell and the location on the screen where messages and indicators are displayed.

Message and indicator position on the screen is specified as a row number followed by a column number. You are requested to specify the coordinate position for

- Abort confirmation
- Insert/replace indicator
- "Off End of Field" indicator

Text of messages and indicators can also be explicitly specified.

These refer to the actual messages/indicators that appear on your screen. You are requested to specify the messages/indicators for:

- Abort confirmation.
- Indicating or clearing replace or insert mode. The insert/replace mode indicator shows whether you are inserting characters or typing over characters.
- Indicating "Off End of Field." When it is enabled, the message you specify is displayed whenever you try to enter data beyond the end of a data field.
- Clearing "Off End of Field" message.

Initialization and Reset Options

You can specify a control character string to initialize the CRT before ACCEPT and DISPLAY statements. This allows you to disable scrolling, etc. while executing the interactive facilities. You may specify the hexadecimal sequences for the following:

- Initialize CRT before Display.
- Initialize CRT before Accept.
- Initialize CRT before a single character is accepted from the keyboard.
- Initialize CRT before First Accept/Display. You can reset any of the functions disabled by initialization. You may also specify the hexadecimal sequences for:

- Reset CRT after Display
- Reset CRT after Accept
- Reset CRT after a single character is accepted from the keyboard

You may specify how you want the cursor to behave and what you want the fields to look like during the execution of an ACCEPT statement.

If you choose not to have automatic display of data fields, you must make sure that your programs execute appropriate DISPLAY statements before each ACCEPT statement. An appropriate DISPLAY statement displays an image on the screen that corresponds with the contents of the relevant data fields of the ACCEPT. The following options are available:

- Capability to specify whether you want the contents of data fields to be automatically displayed during an ACCEPT statement. No automatic display means that the screen remains as it is. Note that numeric fields with full editing enabled will always be pre–displayed when the cursor enters the field irrespective of the settings.

  Choices you have for display of data during an ACCEPT statement are:

  - no display except on entry to numeric edited fields with numeric editing enabled
  - no display except on entry to numeric edited/non–edited fields with numeric editing enabled
  - predisplay of field on entry (individually as the cursor is moved into them).
  - pre–display of all fields before first data entry

- Capability to specify whether full editing of numeric fields is required. This means that attempts to enter characters other than digits, signs and decimal points are rejected. As data is entered, fields are reformatted to show the editing symbols, such as insertion symbols.

- Capability to specify whether you want the cursor to automatically skip to the next field when the current data field is full. This applies only within multiple data fields of one ACCEPT statement.

- Capability to specify the following choices for which keys termination an ACCEPT (in addition to 'terminate accept' key specified in ADIS Function Key List):
    - no other method
    - field–tab in the last field key
    - data entry into the last position/field–tab in last field

You may enter up to 16 tab stop positions.

Function List Options

There are three function key list options that you can use. The ACCEPT/DISPLAY Module (ADIS) list must be configured. The Micro Focus list is required by the ANIMATOR. The User Function list configures the functions your program uses by default.

ADIS Function List. This list is required. If these function keys are not defined, results can be unpredictable.

The keys are defined to the CONFIG utility either by pressing the actual key, or by entering the hexadecimal equivalent of the key sequence.

In addition to defining these function keys, you can also map a second key onto one that is already defined, so that both keys return the same function key value to your program.

Following is the list of ADIS Function Keys:

Function number	Meaning
00	TERMINATE ACCEPT
01	ABANDON PROGRAM
02	MOVE TO START OF NEXT LINE
03	MOVE LEFT ONE CHARACTER
04	MOVE RIGHT ONE CHARACTER
05	MOVE UP ONE LINE
06	MOVE DOWN ONE LINE
07	MOVE TO START OF SCREEN
08	(HOME)MOVE TO NEXT TAB POSITION
09	MOVE TO PREVIOUS TAB POSITION
10	MOVE TO END OF SCREEN
11	MOVE TO NEXT FIELD
12	MOVE TO PREVIOUS FIELD
13	CHANGE CASE OF CHARACTER
14	UNTYPE CHARACTER (RUBOUT)
15	RETYPE CHARACTER
16	INSERT CHARACTER
17	DELETE CHARACTER
18	RESTORE DELETED CHARACTER
19	CLEAR TO END OF FIELD
20	CLEAR (RESET) FIELD
21	CLEAR TO END OF SCREEN
22	CLEAR (RESET) ENTIRE SCREEN
23	SET INSERT MODE
24	SET REPLACE MODE
25	RESET (UNDO) FIELD
26	MOVE TO START OF LINE/FIELD

Micro Focus Function List. The Micro Focus function list is required by the ANIMATOR. This list gives you the capability to set up the function keys **F1** to **F10**, **Escape/Cancel**, and the function keys that allow you to move around the screen text.

The MicroFocus Function List is:

Function number	Meaning
01	FUNCTION KEY 1
02	FUNCTION KEY 2
03	FUNCTION KEY 3
04	FUNCTION KEY 4
05	FUNCTION KEY 5
06	FUNCTION KEY 6
07	FUNCTION KEY 7
08	FUNCTION KEY 8
09	FUNCTION KEY 9
10	FUNCTION KEY 10
65	ESCAPE (Cancel key)
66	PAGE UP (PREVIOUS PAGE)
67	PAGE DOWN (NEXT PAGE)
68	UP (PREVIOUS PAGE * 10)
69	DOWN (NEXT PAGE * 10)
70	TOP
71	BOTTOM

NOTE: Some systems begin their control sequences with an escape character (1B hex). If your terminal does this, you should either configure Escape (function 65) as two presses on the escape key, or if possible, define a different key to perform the escape function. This means that sequences that begin with 1B hex do not cause an escape to be sent. If you configure a function key with a hexadecimal character that also starts a sequence, unwanted results may occur.

User Function List. This is the list of functions your program uses by default. To add functions to this list, you select a function number and enter it with its associated keys, or hexadecimal equivalent. You may define two or more keys with the same function.

Default CRT Handling Options

This section shows a sample default set of configuration options. You can check the current defaults for your version of LEVEL II COBOL by using the Review option of the CONFIG utility. CONFIG is described later in this chapter.

CRT General Configuration

1) Type of addressing	Direct
(A) Screen coordinate format	Absolute binary value
(B) First coordinate increment	00
(C) Second coordinate increment	00
(D) Leading control character sequence	FF, 43
(E) Intermediate control character sequence	Not Defined
(F) Trailing control character sequence	Not Defined
(G) Is the line coordinate output first?	No
(H) Home cursor to top left hand corner	FF, 43, 00, 00

CRT Physical Characteristics

2) Number of CRT columns	80
Is it safe to use the last position on the screen?	No
3) Number of CRT lines	29
Is it safe to use the last position on the screen?	No
4) Does your CRT echo characters?	No
5) Does your CRT support wraparound?	Yes – Forward only
Is highlighting affected by wraparound?	Yes
6) Does your CRT support highlighting?	Yes – Stop & start strings
(A) Control character start sequence	FF,41,45
(B) Control character stop sequence	FF,41,41
(C) Number of characters used in control string	00

Console Output Configuration

1) Clear screen procedure F,FF	FF,46,20,00,00,F
	FF,43,00,00
2) Sound bell sequence	07
3) Decimal coordinate positions for:	
(A) Abort confirmation	25, 01
(B) Insert/replace	25, 79
(C) End of field	25, 79
4) Messages/indicators for:	
(A) Request abort confirmation	Abort Y/N?
(B) Indicate replace mode	Not Defined
(C) Indicate insert mode	I
(D) Clear insert/replace mode	" "
(E) Indicate OFF END OF FIELD	>
(F) Clear OFF END OF FIELD	" "
5) Initialize CRT before:	
(A) Display	FF,50,46
(B) Accept	FF,50,46
(C) Get single character	Not Defined
(D) First Accept/Display	FF,56,4E,FF,50,46

6) Reset CRT sequences after:

(A) Display	Not Defined
(B) Accept	Not Defined
(C) Get single character	Not Defined

7) Accept configuration
Note: Numeric fields with full editing enabled are always be pre–displayed when the cursor enters the field irrespective of the settings.

(A) Display of fields before accept?	No
(B) Is full editing of numeric fields required?	Yes
(C) Within an accept do you wish to move to the next field when the current field is full?	Yes

(D) Termination of accept (in addition to 'terminate accept' key)
Data entry into the last position/field–tab in last field

8) Tab positions	8,16,24,32,40,48, 56,64,72

ADIS function key list:

Function	ASCII code	CT keyboard
00 TERMINATE ACCEPT	0A	Return
01 ABANDON PROGRAM	EB	code–K
02 MOVE TO START OF NEXT LINE	0D	Bound
03 MOVE LEFT ONE CHARACTER	0E	left arrow
04 MOVE RIGHT ONE CHARACTER	12	right arrow
05 MOVE UP ONE LINE	01	up arrow
06 MOVE DOWN ONE LINE	0B	down arrow
07 MOVE TO START OF SCREEN (HOME)	91	code–ScrollUp
08 MOVE TO NEXT TAB POSITION	09	Tab
09 MOVE TO PREVIOUS TAB POSITION	89	code–Tab
10 MOVE TO END OF SCREEN	93	code–ScrollDn
11 MOVE TO NEXT FIELD	09	Tab
12 MOVE TO PREVIOUS FIELD	89	code–Tab
13 CHANGE CASE OF CHARACTER	E6	code–f
14 UNTYPE CHARACTER (RUBOUT)	08	BackSpace
15 RETYPE CHARACTER	F9	code–y
16 INSERT CHARACTER	EF	code–o
17 DELETE CHARACTER	7F	Delete
18 RESTORE DELETED CHARACTER	EC	code–l
19 CLEAR TO END OF FIELD	FA	code–z
20 CLEAR (RESET) FIELD	F8	code–x
21 CLEAR TO END OF SCREEN	E5	code–e
22 CLEAR (RESET) ENTIRE SCREEN	F3	code–s
23 SET INSERT MODE	E9	code–i
24 SET REPLACE MODE	Not Defined	
25 RESET (UNDO) FIELD	E1	code–a
26 MOVE TO START OF LINE/FIELD	Not Defined	

Micro Focus Function list:

	CT Keyboard	ASCII code
01	FUNCTION KEY 1	15 or 00
02	FUNCTION KEY 2	16
03	FUNCTION KEY 3	17
04	FUNCTION KEY 4	18
05	FUNCTION KEY 5	19
06	FUNCTION KEY 6	1A
07	FUNCTION KEY 7	1C
08	FUNCTION KEY 8	1D
09	FUNCTION KEY 9	1E
10	FUNCTION KEY 10	1F
65	ESCAPE (Cancel key)	07
66	PAGE UP (PREVIOUS PAGE)	05
67	PAGE DOWN (NEXT PAGE)	0C
68	UP (PREVIOUS PAGE * 10)	C5
69	DOWN (NEXT PAGE * 10)	CC
70	TOP	85
71	BOTTOM	8C

User Function Key list:

Not Defined

Running the CONFIG Utility

General Procedure

To run CONFIG:

1. Path to [Sys]<COBOL>.

2. Copy the CONFIG* files from the release diskettes.

3. Make a copy of the previous version of [sys]<COBOL>ADIS.254 as a backup. For example, copy [sys]<COBOL>ADIS.254 to [Sys]<COBOL>Saved>ADIS.254.

4. To perform the library functions in CONFIG, the ISAM files CONFIG.TRM and CONFIG.Ind may have to be reorganized to run on your system.

   To reorganize, enter the **ISAM Reorganize** command as follows, and press **Go**:

```
ISAM Reorganize
  ISAM data set or DAM file                              _CONFIG.TRM_____
  [Index file]                                           _CONFIG.Ind_____
  [Work file 1]                                          ________________
  [Work file 2]                                          ________________
  [Use parameters from ISAM data set?]                   _y______________
  [Index keys (e.g., Byte:10.8.ANU.W)]                   ________________
  [B-Tree node size (2 sectors)]                         ________________
  [Data store file growth increment (30 sectors)]        ________________
  [Index file growth increment (30 sectors)]             ________________
  [Initial index file size (30 sectors)]                 ________________
  [Maximum initial B-Tree node fullness (80%)]           ________________
   [Overwrite ok?]                                       ________________
```

5. On the Executive command line, enter **COBOL Run** and press **RETURN**. Complete the COBOL Run command form as shown below, and press **Go** to start the CONFIG utility.

```
COBOL Run
  File Name       __Config__________________________________
  [Parameters?]   __________________________________________
  [Switches]      __________________________________________
```

CONFIG displays an introductory banner,

6. When the CONFIG utility prompts you for the name of the ADIS file recognized by the current release of the compiler, if you have LEVEL II COBOL version 2.5.4 release 1.0 or 2.0, type **ADIS.254**

   Otherwise, check your release notice for the name of the ADIS file.

   CONFIG utility displays a function menu and prompts you with a ? to enter the integer associated with the function required. After the specified function is completed, the menu is redisplayed and you are prompted to enter a number for the next function you require.

   The CONFIG functions are described below.

CONFIG Function Operation

Functions involving the ISAM dataset containing the library of ACCEPT/DISPLAY Modules require that ISAM be installed.

The Main Menu of CONFIG Functions is:

1) QUIT	Return control to the operating system.
2) UPDATE	Write the new ACCEPT/DISPLAY Module to disk.
3) INPUT	Enter a new set of accept/display values or retrieve an ACCEPT/DISPLAY Module from the ISAM dataset containing the library of ACCEPT/DISPLAY Modules.
4) ALTER	Change a single accept/display value.
5) REVIEW	List the currently defined accept/display values
6) LIBRARY	Review/save/delete/overwrite to/from the ISAM dataset containing the library of ACCEPT/DISPLAY Modules.

QUIT

Main Menu choice 1, QUIT, returns you to the Executive.

UPDATE

Main Menu choice 2, UPDATE, causes CONFIG to write your updated ACCEPT/DISPLAY Module to disk.

This function is used to update the ACCEPT/DISPLAY Module that you specify with the values you have changed using the ALTER function, or the INPUT function.

INPUT

Main Menu choice 3, INPUT, causes CONFIG to ask if you wish to input the configuration values from a CRT configuration file already existing on the disk.

If you respond with an N selecting no, CONFIG prompts you to alter or retain every value of your ACCEPT/DISPLAY Module. Default values are taken from the currently loaded ACCEPT/DISPLAY Module.

If you respond with a Y selecting yes, CONFIG prompts you to input a CRT configurations file from the library containing ACCEPT/DISPLAY Modules. Accessing this library requires that you have ISAM installed. The default file name for the library containing ACCEPT/DISPLAY Modules is the ISAM dataset CONFIG.TRM. Its index file is CONFIG.Ind.

After entering the library file name, CONFIG displays the following:

```
This file contains the following configurations:
00 :  Quit - CRT not present
nn :
.   \
.   / (pre-specified ADIS descriptions )
nn :
```

When you enter the number associated with the file you require, CONFIG loads the values from that file. If you wish to overwrite the existing ADIS file with the newly loaded library file, you must enter a 2 for INPUT when the Main Menu of CONFIG reappears.

ALTER

Main Menu choice 4, ALTER, gives you the capability to selectively change one or more ADIS values. The following ALTER Function Sub-Menu is displayed:

```
0) Terminate the alter process
1) Alter CRT Miscellaneous Values
2) Alter Console Output value
3) Alter ADIS/user function key lists
```

If you enter 0 for the ALTER Sub-Menu function Terminate the alter process, the alter process is terminated, and CONFIG returns to the Main Menu of CONFIG for another function entry.

If you enter 1 for the ALTER Sub-Menu function Alter CRT Miscellaneous Values, CONFIG displays the following list of Miscellaneous CRT Sub-Menu functions:

```
0) Terminate the alter process
1) Alter addressing mode supported by CRT.
2) Alter the number of CRT columns
3) Alter the number of CRT lines
4) Alter auto character echo
5) Alter auto wraparound
6) Alter character highlighting
```

Miscellaneous CRT Sub-menu options. CONFIG prompts you to enter the number for the required function from the ALTER CRT Miscellaneous Values Sub-Menu. Following is a brief description of each function of that Sub-Menu :

If you enter 0 selecting the ALTER CRT Miscellaneous Values Sub-Menu function Terminate the alter process, the alter process is terminated, and CONFIG returns to the Main Menu of CONFIG for another function entry.

If you enter 1 selecting the ALTER CRT Miscellaneous Values Sub-Menu function, you may alter the addressing mode to correspond to the one supported by your CRT. There are three addressing modes, direct addressing, step addressing and subroutine addressing. Following is a description of the information requested for each mode:

0) Direct addressing to a particular screen position requires the following information:

(A) Screen coordinate format (Either absolute binary value or 1-byte or 2-byte or 3-byte ASCII)

(B) First coordinate increment

(C) Second coordinate increment

(D) Leading control character sequence

(E) Intermediate control character sequence

(F) Trailing control character sequence

(G) Is the line coordinate output first ?

(H) Hexadecimal character sequence to home the cursor to top left hand corner

1) Step addressing by single position cursor moves requires that you specify which keys perform the following functions:

(A) Move cursor one position left

(B) Move cursor one position right

(C) Move cursor one position up

(D) Move cursor one position down

(E) Home the cursor to top left

2) User subroutine addressing by call to routine requires that you enter:

(A) Call code identifier number

(B) Parameter block – up to 3 2-digit

If you enter 2 selecting the ALTER CRT Miscellaneous Values Sub-Menu function, you may alter the number of CRT columns to correspond to your CRT by entering the number of columns in decimal. You are also asked if it is safe to use the last position on the screen.

If you enter 3 selecting the ALTER CRT Miscellaneous Values Sub–Menu function, you may alter the number of CRT lines to correspond to your CRT by entering the number of lines in decimal. You are also asked if it is safe to use the last position on the screen.

If you enter 4 selecting the ALTER CRT Miscellaneous Values Sub–Menu function, you may alter information about auto character echo to correspond to your CRT.

> You are asked: 'Does the CRT automatically echo input characters?'. If you answer yes, you are asked: 'Are all characters echoed (including control characters)?'

If you enter 5 selecting the ALTER CRT Miscellaneous Values Sub–Menu function, you may alter information about auto wraparound to correspond to your CRT.

> Does the CRT forward wraparound onto the next line?
> Does the CRT backward wraparound onto the previous line?
> Is highlighting affected by wraparound/repositioning?

If you enter 6 selecting the ALTER CRT Miscellaneous Values Sub–Menu function, you may alter information about character highlighting to correspond to your CRT. There are three options available for specifying highlighting capabilities of your CRT. Following is a description of the information required for each specification:

> 0) Not implemented requires no further information.
>
> 1) Implemented by a character mask requires a two digit hex highlight mask, e.b. '80'.2) Implemented by start and stop control strings requires the following information:
>
> > (A) Control character start sequence
> >
> > (B) Control character stop sequence
> >
> > (C) Number of characters at end of field used by the control string

If you enter 2 selecting the ALTER Sub–Menu function to specify 'Alter Console Output value', CONFIG displays the following list of Console Output Sub–Menu functions. CONFIG prompts you to enter the number for the required function.

```
0) Terminate the alter process
1) Alter the clear screen procedure
2) Alter the 'sound bell' sequence
3) Alter message positioning
4) Alter messages/indicators
5) Alter the initialize CRT sequences
6) Alter the reset CRT sequences
7) Alter accept configuration
8) Alter tab positions
```

Console Output Sub–Menu options.

If you enter 0 selecting the ALTER Console Output Value Sub–Menu function 'Terminate the alter process', the alter process is terminated, and CONFIG returns to the Main Menu of CONFIG for another function entry.

If you enter 1 selecting the ALTER Console Output Value Sub–Menu function, you may alter the clear screen procedure by entering the ASCII code to correspond to the one supported by your CRT.

If you enter 2 selecting the ALTER Console Output Value Sub–Menu function, you may alter the 'sound bell' sequence by entering the ASCII code that corresponds to the one supported by your CRT.

If you enter 3 selecting the ALTER Console Output Value Sub–Menu function, you may specify decimal coordinate positions to alter the message positioning for:
(A) Abort confirmation message
(B) Insert/replace mode indicators
(C) End of field indicator

If you enter 4 selecting the ALTER Console Output Value Sub–Menu function, you may specify the message you wish to see on the screen for:
(A) Request abort confirmation
(B) Indicate replace mode
(C) Indicate insert mode
(D) Clear insert/replace mode
(E) Indicate 'OFF END OF FIELD'
(F) Clear 'OFF END OF FIELD'

If you enter 5 selecting the ALTER Console Output Value Sub–Menu function, you may specify the hexadecimal value to initialize

the CRT before:
(A) Display
(B) Accept
(C) 'Get single character'
(D) First Accept/Display

If you enter 6 selecting the ALTER Console Output Value Sub-Menu function, you may specify the hexadecimal value to reset the CRT after:
(A) Display
(B) Accept
(C) 'Get single character'

If you enter 7 selecting the ALTER Console Output Value Sub-Menu function, you may alter information about your Accept configuration.
The following options are available for specifying how you wish your Accept statements to behave at execution time:

(A) Display of fields before accept has the following options available:

(0) No display except on entry to numeric edited fields with numeric editing enabled

(1) No display except on entry to numeric edited/non-edited fields with numeric editing enabled

(2) Pre-display of field on entry

(3) Pre-display of all fields before first data entry

(B) Is full editing of numeric fields required?

(C) Within an accept do you wish to move to the next field when the current data field is full?

(D) Termination of accept (in addition to 'terminate accept' key) has the following options available:

0) No other method

1) Field-tab in the last field key

2) Data entry into the last position/field-tab in last field

If you enter 8 selecting the ALTER CRT Miscellaneous Values Sub–Menu function, you may alter tab positions by entering up to 16 positions as 2 digit decimal numbers, separated by commas E.G. '08,16'.

If you enter 3 selecting the ALTER Sub–Menu function Alter ADIS/user function key lists', CONFIG displays the following list of ADIS/user function key lists Sub–Menu functions:

```
0) Terminate the alter process
1) ADIS  function key list
2) Micro Focus Function list
3) User Function Key list
```

ADIS/user function key Sub–menu options.

If you enter 0 selecting the ALTER ADIS/user function key lists Sub–Menu function 'Terminate the alter process', the alter process is terminated, and CONFIG returns to the Main Menu of CONFIG for another function entry.

If you enter 1 selecting the ALTER ADIS/user function key lists Sub–Menu function, you may alter ADIS function key list to correspond to your CRT and/or applications.

If you change an item of the ADIS function key list, you may enter the appropriate sequences as a string of two digit hex numbers separated by commas, E.G. '01,0A' or " to remove the entry or "Q" to quit from this list. You can define a key to return the value of a different function, by mapping the keystroke to another function. To do this, two values are required. Firstly you must enter the function number being mapped to in quotes (E.g. "05" maps a keystroke to function 05). Secondly the keystroke must be entered to the function prompt. You may enter a new value,"RETURN" to keep the current value, " to remove the entry or "Q" to quit from this list.

The following is the ADIS Function List:

00	TERMINATE ACCEPT
01	ABANDON PROGRAM
02	MOVE TO START OF NEXT LINE
03	MOVE LEFT ONE CHARACTER
04	MOVE RIGHT ONE CHARACTER
05	MOVE UP ONE LINE
06	MOVE DOWN ONE LINE
07	MOVE TO START OF SCREEN (HOME)
08	MOVE TO NEXT TAB POSITION
09	MOVE TO PREVIOUS TAB POSITION
10	MOVE TO END OF SCREEN
11	MOVE TO NEXT FIELD
12	MOVE TO PREVIOUS FIELD
13	CHANGE CASE OF CHARACTER
14	UNTYPE CHARACTER (RUBOUT)
15	RETYPE CHARACTER
16	INSERT CHARACTER
17	DELETE CHARACTER
18	RESTORE DELETED CHARACTER
19	CLEAR TO END OF FIELD
20	CLEAR (RESET) FIELD
21	CLEAR TO END OF SCREEN
22	CLEAR (RESET) ENTIRE SCREEN
23	SET INSERT MODE
24	SET REPLACE MODE
25	RESET (UNDO) FIELD
26	MOVE TO START OF LINE/FIELD

If you enter 2 selecting the ALTER ADIS/user function key lists Sub-Menu function, you may alter Micro Focus Function key list that is used by the ANIMATOR.

The Micro Focus Function list for the ANIMATOR includes:

01	FUNCTION KEY 1
02	FUNCTION KEY 2
03	FUNCTION KEY 3
04	FUNCTION KEY 4
05	FUNCTION KEY 5
06	FUNCTION KEY 6
07	FUNCTION KEY 7
08	FUNCTION KEY 8
09	FUNCTION KEY 9
10	FUNCTION KEY 10
65	ESCAPE (Cancel Key)
66	PAGE UP (PREVIOUS PAGE)
67	PAGE DOWN (NEXT PAGE)
68	UP (PREVIOUS PAGE * 10)
69	DOWN (NEXT PAGE * 10)
70	TOP
71	BOTTOM

If you enter 3 selecting the ALTER ADIS/user function key lists Sub-Menu function, you may alter User Function Key list to correspond to your CRT and/or applications.

The ACCEPT/DISPLAY Module can return to the user program an integer value which is associated with a function key sequence entered by the program operator. You may enter the value required and the associated key sequence.

REVIEW

Main Menu choice 5, REVIEW, lists all of the defined values of the ACCEPT/DISPLAY Module you have currently loaded.

LIBRARY

Main Menu choice 6, LIBRARY, causes CONFIG to ask if you wish to input the configuration values from the library containing ACCEPT/DISPLAY Modules in the ISAM dataset CONFIG.TRM. Its index file is CONFIG.Ind. This function requires that ISAM be installed.

If the file you request does not exist, CONFIG has the capability to create a library file using ISAM.

After entering the library file name, CONFIG displays the following LIBRARY function Sub-Menu:

```
0) Terminate SAVE routine
1) Review CRT configurations available
2) Save a new configuration
3) Delete an existing configuration
4) Overwrite an existing configuration
```

If you enter 0 selecting the LIBRARY Sub-Menu function Terminate SAVE routine, CONFIG returns the main menu of CONFIG functions.

If you enter 1 selecting the LIBRARY Sub-Menu function Review CRT configurations available, CONFIG displays the ACCEPT/DISPLAY Module entries in your library dataset as follows:

```
This file contains the following configurations :-
00 :  Quit - CRT not present
nn :
.   \
.   / (ADIS descriptions entered in RESPONSE '4' )
nn :
```

If you enter 2 selecting the LIBRARY Sub-Menu function Save a new configuration, CONFIG requests a descriptive name of up to 32 characters and enter the current ACCEPT/DISPLAY Module into the library.

If you enter 3 selecting the LIBRARY Sub-Menu function Delete an existing configuration, CONFIG displays the ACCEPT/DISPLAY Module entries in your library dataset, and the request you to enter the number associated with the dataset you wish to delete.

If you enter 4 selecting the LIBRARY Sub-Menu function Overwrite an existing configuration, CONFIG displays the ACCEPT/DISPLAY Module entries in your library dataset, and the request you to enter the number associated with the dataset you wish to overwrite.

Index