Microsoft®
COBOL Compiler

for the MS™-DOS Operating System

User's Guide

Microsoft Corporation

If you have comments about the software or this documentation, please complete the Software Problem Report at the back of this manual and return it to Microsoft Corporation.

Document Number 8302A-200-03
Part Number 011-014-020

Contents

Runtime Licensing Information

The distribution policy for parts of the Microsoft COBOL Compiler is as follows:

> Neither the RUNCOB.EXE runtime module, nor the REBUILD or INSTALL Utility can be distributed without first entering into a license agreement with Microsoft Corporation for such distribution. A copy of the license agreement can be readily obtained by writing to Microsoft. In addition, a copyright notice reading "PORTIONS COPYRIGHTED BY MICROSOFT CORPORATION, 1984" must be displayed on the media.
>
> No other software in your Microsoft COBOL Compiler package can be duplicated, except for purposes of backing up your software. Other duplication of any of the software in the Microsoft COBOL Compiler package is illegal.

System Requirements

Microsoft COBOL requires a minimum of 192K bytes of memory. Additional memory may be required to run large programs. Each MS-COBOL program is limited to 64K of code. An MS-COBOL system consisting of a main program and one or more subprograms may contain more than 64K of code, but each program is limited to 64K of code. The entire system is limited to approximately 60K of working storage data.

Two disk drives are recommended, although you can use MS-COBOL with just one double-sided disk drive.

MS-DOS 2.0 or higher is required.

An MS-DOS CONFIG.SYS file should be present, and the FILES command should specify FILES = 10 or more files.

About These Manuals

The following manuals document Microsoft COBOL:

Microsoft COBOL Compiler User's Guide

Provides information on a particular implementation or operating system. Included is a list of system requirements and a description of disk contents. The *User's Guide* also provides general instructions on compiling, loading, and executing programs on your operating system.

Microsoft COBOL Reference Manual

Contains detailed descriptions of the Microsoft COBOL language. Information in the *Reference Manual* applies to most implementations of the Microsoft COBOL Compiler. (Exceptions are noted in the *User's Guide.*)

Microsoft COBOL Quick Reference Guide

Outlines the COBOL program structure and gives the formats of individual statements.

Chapter 1
Introduction to the Microsoft COBOL Compiler

The Microsoft® COBOL Compiler (MS™-COBOL), version 2.0, runs under versions 2.0 and later of the Microsoft Disk Operating System (MS-DOS), and accepts programs written in the language defined in the *Microsoft COBOL Reference Manual.* The Microsoft COBOL language conforms to the ANSI X3.23-1974 COBOL requirements.

Thousands of existing COBOL applications written for larger computers can be adapted and transported to Microsoft COBOL. Through the INSTALL Utility you can configure MS-COBOL to a large number of terminals, and take advantage of MS-COBOL's advanced interactive screen-handling features.

The Microsoft COBOL Compiler gives you COBOL's high-level programming language advantages, as well as many extensions to the full language standard COBOL. The purpose of this *User's Guide* is to help you get a Microsoft COBOL program up and running on your computer.

Two important features of this version of COBOL are the ability to run programs configured for a particular terminal, and the ability to run programs that make use of indexed (ISAM) files. Compiled programs that use either of these features require special consideration before they will run.

In the case of a program that depends on a particular terminal configuration, you will need to first run the INSTALL program to configure the MS-COBOL Runtime Executor. For more information, see Chapter 9, "INSTALL Program."

For a program that manipulates ISAM files, you will need to first load the Microsoft Multi-Key ISAM Facility, *before* the given program is executed, because the MS-COBOL Runtime Executor expects to find Microsoft ISAM resident in main memory at runtime. For more information about loading Microsoft-ISAM, see Appendix G, "Loading the Indexed File Handler."

1.1 Features and Benefits

1. Microsoft COBOL can define entire interactive terminal screens, and permits single-statement data entry and display. MS-COBOL supports interactive data entry with:

   a. automatic cursor positioning
   b. automatic numeric field editing
   c. data field screen attributes such as REVERSE-VIDEO and HIGHLIGHT

   These capabilities come from Microsoft extensions to the DATA DIVISION (SCREEN SECTION) and the PROCEDURE DIVISION (ACCEPT and DISPLAY statements). (See Section 6.4.4, "SCREEN SECTION," and Formats 1, 3, and 4 of the ACCEPT and Format 3 of the DISPLAY statement in Chapter 7, "PROCEDURE DIVISION," in the *Microsoft COBOL Reference Manual.*)

2. COMP-0, COMP-3, and COMP-4 data formats are available. COMP-0 and COMP-4 are two- and four-byte binary integers, respectively. COMP-3 format packs numeric data two digits per byte. All three data types can be used to reduce DATA DIVISION memory requirements, reduce data-file storage requirements, and increase the execution speed of certain operations.

3. Lowercase characters are treated as upper case, unless they are part of a non-numeric (quoted) literal.

4. The dynamic debugging statements READY TRACE, RESET TRACE, and EXHIBIT allow the display of procedure names or data-items during program execution. The Interactive Debug Facility, a standard COBOL feature, provides extensive runtime debugging functions.

5. The SELECT clause of the ENVIRONMENT DIVISION supports a split-key option with both the RECORD KEY and the ALTERNATE RECORD KEY clauses.

6. A CHAIN statement and a CHAINING phrase extend the scope of interprogram communication and allow any program to be loaded into memory and executed.

7. A file sharing construct supports file processing in multi-user/multi-tasking systems. The new syntax applies in the OPEN, READ, START, and UNLOCK statements, and the SELECT clause. This syntax is supported by the MS-COBOL Compiler, but record locking is not supported at runtime under MS-DOS version 2.xx.

8. Sort file-status reporting is implemented through the SORT STATUS clause in the FILE-CONTROL entry for a sort file.

9. The built-in extension subroutines, EXIST, RENAME, REMOVE, COMMAND, UPCASE, LOCASE and EXCODE, described in Chapter 10, "Interprogram Communication," provide additional facilities not normally available to COBOL programs.

Microsoft COBOL supports four file types: Sequential, Line Sequential, Relative, and Indexed. A Microsoft COBOL Indexed file is created by the Microsoft ISAM Facility and restored by the Rebuild Utility. The Microsoft ISAM Facility, which is included with Microsoft COBOL, provides fast, random retrieval of Indexed data records. Refer to Appendix G, "Loading the Indexed File Handler," for more information on the ISAM facility.

1.2 How to Use This Manual

This manual provides information on compiling Microsoft COBOL source programs with the MS-COBOL Compiler, and executing the generated code with the runtime executor, RUNCOB.EXE.

Chapter 1 describes the capabilities of the Microsoft COBOL Compiler.

Chapters 2 through 4 explain how to compile, load, and execute an MS-COBOL program. These chapters also contain information specific to your MS-COBOL implementation.

Chapter 5 tells you how to set up a batch command file to "compile, load, and go."

Chapter 6 explains the definition of search paths for COPY files. This discussion includes the use of library-names in the COPY statement, and the compiler command line /S switch with full pathnames.

Chapter 7 explains the four disk file organizations: Sequential, Line Sequential, Relative, and Indexed. It also describes how to use disk input/output files and other types of files.

Chapter 8 tells you how to use the Interactive Debug Facility to correct program errors at runtime.

Chapter 9 tells you how to configure MS-COBOL to your terminal (required before executing compiled programs that use MS-COBOL interactive screen extensions).

Chapter 10 explains interprogram communication with the CALL and CHAIN statements. Included in this discussion is an explanation of MS-COBOL's Extension Subroutines.

Chapter 11 describes the Rebuild Utility, which allows you to recover or restore Indexed file information.

Appendix A explains the differences between the Microsoft COBOL Compiler 2.0 and previous versions.

Appendix B describes the five phases of the MS-COBOL Compiler. This appendix may be useful if your program generates a "Compiler Phase Error."

Appendix C shows you how to modify MS-COBOL's tab stops.

Appendix D lists key assignments for special functions that apply to your terminal. It also contains a list of terminals supported by the INSTALL Utility.

Appendix E describes the demonstration programs included with the MS-COBOL Compiler: a test program for the INSTALL terminal interface, a simple MS-COBOL program, and three programs that demonstrate the MS-COBOL screen and Indexed file-handling capabilities.

Appendix F lists the compiler and runtime error messages. Compiler error messages are arranged alphabetically within four sections: command input and operating system I-O errors, program syntax errors, file usage errors, and warnings. Runtime errors are in a separate alphabetical list.

Appendix G describes how to load and use the Microsoft Multi-Key ISAM facility (ISAM). ISAM performs Indexed file input and output.

1.3 Notational Conventions

The following textual conventions are used throughout this manual to represent elements of programming syntax. Microsoft COBOL syntax requirements for individual language elements can be found in the *Microsoft COBOL Reference Manual.*

[ ]	Square brackets indicate that the enclosed text is optional.
{ }	Braces indicate that the user has a choice between two or more entries. At least one of the entries enclosed in braces must be chosen unless the entries are also enclosed in square brackets. Braces also delimit the portion of a statement that is referred to by an ellipsis.
|	Vertical bars separate the choices within braces. At least one of the entries must be chosen unless the entries are also enclosed in square brackets.
. . .	Ellipses indicate that an entry may be repeated as many times as needed or desired.
Italics	Italics represent user-entered data, and will appear in two forms. *Lowercase* italics indicate the type of text a user must enter. For example, CALL *filename* prompts the user to enter the name of a file. *UPPERCASE ITALICS* represent data which must be entered exactly as shown.
INPUT/ OUTPUT	This typeface represents input and output as well as computer displayed text. All caps in a command line represents text that must be entered exactly as shown.

CAPS	Small caps represent a key or combination of keys.

All other punctuation, such as commas, colons, slash marks, and equal signs must be entered exactly as shown.

1.4 Learning More About COBOL

If you are new to COBOL programming, you may want to learn more about the language before using this manual. The following texts are COBOL tutorials, written for the novice programmer:

Abel, Peter. *COBOL Programming: A Structured Approach.* Reston, VA.: Reston Publishing Co., 1980.

McCracken, Daniel D. *A Simplified Guide to Structured COBOL Programming.* New York: John Wiley and Sons, Inc., 1976.

Parkin, Andrew. *COBOL for Students.* London: Edward Arnold Ltd., 1975.

Seidel, Ken. *Microsoft COBOL.* Beaverton, OR.: Dilithium Press, 1983.

Welburn, Tyler. *Structured COBOL — Fundamentals and Style.* Palo Alto: Mayfield Publishing Co., 1981.

Chapter 2
Getting Started

The purpose of this *User's Guide* is to help you get a Microsoft COBOL program up and running on your computer. To do this, you need to perform two initial tasks, as well as understand the major steps involved in using MS-COBOL. This chapter begins by listing the contents of your disks and by telling you how to perform disk backup and terminal configuration—two tasks generally performed only once. It then presents an overview of the compilation process and a sample program development session.

2.1 Microsoft COBOL System Software

This package contains one or more Microsoft COBOL distribution disks. Disk 1 contains file FILES.DOC, which lists the contents of the distribution disks.

Important

Disk 1 files FILES.DOC, UPDATE.DOC and INSTALL.DOC may contain information that was unavailable when this manual was printed. Locate and read these files before attempting to use MS-COBOL.

The following files are included on your disk(s):

The MS-COBOL Compiler

COBOL.EXE	the main compiler program
COBOL0.OVR	overlay 0
COBOL1.OVR	overlay 1
COBOL2.OVR	overlay 2
COBOL3.OVR	overlay 3
COBOL4.OVR	overlay 4

Runtime System

RUNCOB.EXE	the common runtime executor
DEBUGCOB.EXE	the Interactive Debug Facility
ISAM.EXE	the Indexed File Handler

Utility Software

REBUILD.EXE	a program that recovers damaged Indexed files
INSTALL.COM	a program that performs terminal interface configuration
INSTALL.MSG INSTALL.OVL INSTALL.SPC	files needed by the INSTALL program
INSTALL.DAT	terminal data file for the INSTALL program

Demonstration Programs

CRTEST.COB	a test program for the terminal interface, as customized by INSTALL
CENTER.COB	a test program for the MS-COBOL Compiler and runtime system
CENTER.INT	the compiled version of CENTER

The MS-COBOL Demonstration System

DEMO.COB	a program to demonstrate the MS-COBOL SCREEN SECTION, to CALL the subprogram BUILD, and to CHAIN to the program UPDATE
DEMO.CPY	a file used by the COPY verb in DEMO.COB

BUILD.COB	a program to create an Indexed (ISAM) file of names, addresses, and telephone numbers
UPDATE.COB	a program to list or update the ISAM file created by BUILD
CLDEMO.BAT	a batch command file for compiling the Demonstration System

Assembly Language Subroutine Libraries

COBOL1.LIB COBOL2.LIB DEBUG.LIB	MS-COBOL runtime libraries
COBOL1.OBJ COBOL2.OBJ DEBUG.OBJ ASM.ASM USERPROG.MAC USERSEG.MAC PSEG.MAC	modules needed for incorporating assembly language routines into the MS-COBOL runtime executor
MAKERUN.BAT	batch file for linking new runtime executor
MAKERUN2.BAT	batch file for linking new runtime executor on floppy disk systems

Miscellaneous Files

FILES.DOC	a file listing the contents of the distribution disks
UPDATE.DOC	an optional file containing update information
INSTALL.DOC	an optional file containing INSTALL update information

2.2 Disk Backup

When you first receive your disk(s), make copies to work with, and save the original disk(s) as backups. Use the COPY or DISKCOPY utilities supplied on your operating system disk for this task.

Next, check your copy of the compiler and runtime system by compiling and executing the test program, CENTER.COB. To do this, refer to the sample program development session in Section 2.4, "Sample Session."

Finally, if your programs use the interactive ACCEPT and DISPLAY facility in MS-COBOL, you must use the INSTALL program to make MS-COBOL compatible with your terminal. You must also run INSTALL before you can use the MS-COBOL Demonstration System. See Chapter 9, "INSTALL Program," for more information.

2.3 Compilation and Execution

MS-COBOL programs are first compiled and then run. The MS-COBOL Compiler consists of the main module (COBOL.EXE) and five overlays (COBOL0.OVR through COBOL4.OVR). The compiler routines analyze your COBOL program and produce an object code file. This file will have a .INT file extension.

Compilation is performed in two passes. The first creates an intermediate version of the program, which is stored in a temporary work file. The second creates the final version of the object code.

The runtime system (RUNCOB.EXE), hereafter referred to as the "runtime executor," loads and runs the compiled program.

Figure 2.1 outlines the "compile, load, and run" process.

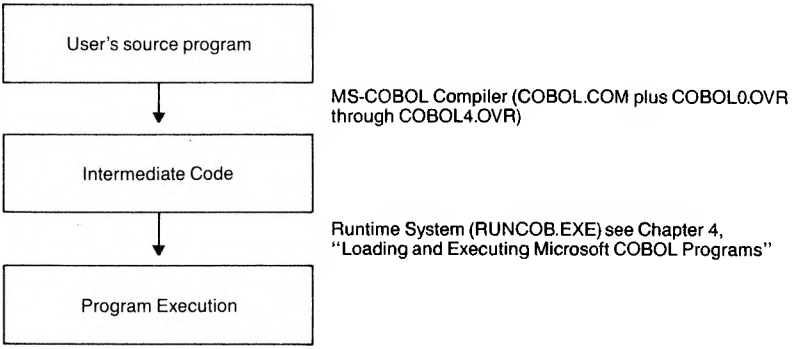


Figure 2.1 Major Steps in Compiling and Executing an MS-COBOL Program

2.4 Sample Session

The compilation and execution of an MS-COBOL program are described in detail in Chapter 3, "Compiling Microsoft COBOL Programs," and Chapter 4, "Loading and Executing Microsoft COBOL Programs." To give you an overview of the MS-COBOL system, however, the following sample session is provided. We recommend that you work through the sample session and then read Chapters 3 and 4 before beginning to compile your own programs.

The examples in this sample session are designed for systems with two disk drives of at least 160K capacity. The program development steps, however, are appropriate for all implementations of MS-COBOL.

Organize Your Disks (Step 1)

Organize the files on your disks to minimize disk-swapping and "Disk Full" errors during program development. Usually, MS-COBOL program development will require three working disks: one for your source and object programs, one for the MS-COBOL Compiler, and the other for the runtime executor, the text editor, and any other necessary utilities.

For example, your three working disks might contain the following files:

a. Program disk

   Your MS-COBOL program (source and object files) will be placed on this disk.

b. Compiler disk

   COBOL.EXE
   COBOL0.OVR
   COBOL1.OVR
   COBOL2.OVR
   COBOL3.OVR
   COBOL4.OVR

c. Utility disk

   A text editor (e.g., EDLIN)
   RUNCOB.EXE

The compiler disk should be copied from the distribution disk (see Section 2.1, "Microsoft COBOL System Software"). It should contain only the compiler.

During development, the program disk will be kept in drive B:, and either the compiler disk or the utility disk (depending on which disk is required at the time) will be in drive A:.

Drive B: should be selected as the default drive (where new files are placed unless otherwise specified in the command). This arrangement simplifies access to the program files by placing them all on the same disk.

The MS-DOS PATH command should be executed now, or as part of the AUTOEXEC.BAT file, to place drive A: on the system search path. To do this, type

```
PATH A:\
```

Setting the path will allow programs such as COBOL.EXE to be found on the root directory of drive A: without an explicit drive specification, and will allow MS-COBOL to find its overlays on the root directory of drive A:.

See Section 3.1, "Invoking the Compiler," for information on compiler overlay searching. See your *Microsoft MS-DOS User's Guide* for information on the PATH command and AUTOEXEC.BAT file.

Create the Source Program (Step 2)

In this sample session, we'll use the sample program CENTER.COB for the source program. CENTER asks you to enter a line of text, and lets you choose whether to center the text, or align it at the left or right margin. (CENTER can be converted into a subroutine for your own use later.)

You may either transfer CENTER.COB from the MS-COBOL distribution disk to your program disk, or use EDLIN or another text editor to create your own source program. This source program should also have the name CENTER.COB.

Transferring CENTER.COB

a. After booting the system as usual, place your program disk in drive B. Then place the disk containing CENTER.COB in drive A:. Copy it to your program disk by typing

   ```
   COPY A:CENTER.COB B:
   ```

b. Select B: as the default by typing

   ```
   B:
   ```

Creating a Source Program with EDLIN

a. After booting the system as usual, place your utility disk in drive A:. Then place the program disk in drive B: and select B: as the default drive by typing

```
B:
```

b. Type the command

```
EDLIN A:CENTER.COB
```

to run the EDLIN editor program so you can write your own MS-COBOL program.

c. When you have finished writing the program, use the EDLIN "E" command to place the file CENTER.COB on your program disk and exit to the operating system.

Check Program Syntax With Trial Compilation (Step 3)

Check your program for syntax errors with a "quick" compilation: compile the program and display the error listing on the screen. No object or listing files are created, so compilation is faster than usual.

a. Remove the utility disk and place the compiler disk in drive A:.

b. To compile CENTER.COB and display a list of errors on the terminal, type the command

```
COBOL CENTER,NUL;
```

The compiler first looks for its overlays (COBOL0.OVR through COBOL4.OVR) in the current directory of the default drive (drive B: in this example). If the overlays are not in the current directory, the compiler uses either the MS-DOS search path (defined by the PATH command) or a directory specified on the command line by the /C switch (see Section 3.1.3, "Using Compiler Switches") to find them.

The previous example and those in Step 4 assume that drive A: is on the search path as a result of your issuing the PATH command, "PATH A:\". When the PATH command has been

issued and the disks have been arranged as described in this example, COBOL.EXE and its overlays will be found on drive A:.

If errors occur during the trial compilation, go back to Step 2 and correct the source file (with the utility disk in drive A:). See Appendix F, "Error Messages," for a list of error messages and explanations. When the trial compilation is error free, you are ready to proceed to Step 4.

Compile the Source Program (Step 4)

Now the program is ready to be compiled. Compilation produces the object file. First, make sure the compiler disk is in drive A: and you are logged onto drive B:.

To compile the program and produce an object file (named CENTER.INT), enter one of the following commands:

`COBOL CENTER;`	produces just the object file
`COBOL CENTER,,PRN`	produces an object file and printed listing
`COBOL CENTER,,CENTER`	produces an object file and a list file (named CENTER.LST)

When compilation is successful, the message "No Errors or Warnings" is displayed, and the compiler exits to the operating system.

Your program disk now contains the following files: CENTER.COB, CENTER.INT, CENTER.DBG, and, if you requested a list file, CENTER.LST. The .DBG file will not be used in this session, and may be deleted.

Load and Execute the Program (Step 5)

To run a program, you need the object file (CENTER.INT) and the common runtime executor (RUNCOB.EXE). RUNCOB.EXE may be in either drive. The compiler will search for it first on the default drive, then on drive A:.

In this example, CENTER.INT is on the program disk and RUNCOB.EXE is on the utility disk in drive A:. Since we are keeping the program disk in drive B:, and drive B: is selected as the default drive, type

```
RUNCOB CENTER
```

Even though you've been very careful to remove all compile time errors, runtime errors may still occur when the program is run. Error messages are described in Appendix F of this manual. If runtime errors occur, return to Step 2 and edit the program to correct the errors.

Chapter 3
Compiling Microsoft COBOL Programs

As in Chapter 2, the sample commands in this chapter assume that

1. The Microsoft COBOL Compiler disk is in drive A:.
2. Your program disk is in drive B:.
3. Drive B: has been selected as the default drive.
4. Drive A: has been placed on the MS-DOS search path using the PATH command.

In the following examples, the file CENTER.DBG will be produced in addition to the files specified. Files with a .DBG extension are used by the Interactive Debug Facility (see Chapter 8, "Interactive Debug Facility"). Use the /D compiler switch to suppress .DBG files. However, since the .DBG file assists in debugging, we recommend that you produce it during program development.

3.1 Invoking the Compiler

Two methods may be used to invoke the MS-COBOL Compiler. Note that the discussions in Sections 3.1.1, "Filenames," and 3.1.2, "Partial Command Strings," apply to both. Therefore, you should read both of these sections before you begin to compile your own programs.

Note

The MS-COBOL Compiler first searches for its overlays (COBOL0.OVR through COBOL4.OVR) in your current directory. If the overlays are not found there, the directories defined by your MS-DOS PATH environment are searched. If the overlays are still not found, an error results. You can also use the /C switch to specify the directory or drive in which the overlays are to be found. The following examples assume that an MS-DOS PATH A:\ command has been issued, so that drive A: is on the system search path, and the compiler and its overlays will be found in the current directory on drive A:.

---

1. You may invoke the compiler by entering the command

   COBOL

   and replying to the following prompts. (Filenames are discussed in Section 3.1.1.)

   Source filename [.COB]:

   Name of your source program. A filename must be specified. If no extension is specified, .COB will be appended by default.

   Object filename [*source filename*.INT]:

   Name of the object file to be created. The source filename is the default filename. .INT is the default extension, and the object file must *always* be given the extension .INT.

   Source listing [NUL.LST]:

   Name of the file to which the program listing is to be written. If a filename is entered, its default extension is .LST. If no filename is entered, the default is NUL (no list). See Section 3.2, "The Source Listing File," for further discussion of the list file.

   Example: The following series of responses compiles the source file CENTER.COB, producing the object file CENTER.INT, the debug file, CENTER.DBG, and a listing file CENTER.LST on the default drive:

   ```
   COBOL
   Source filename [.COB]:      CENTER
   Object filename [CENTER.INT]: RETURN
   Source listing [NUL.LST]:    CENTER
   ```

2. The compiler can also be invoked with the command

   COBOL *command string*

   where the command string contains

   source filename , *object filename* , *source listing*[;]

   as explained above and in Section 3.1.1, "Filenames."

   The separator character is the comma (,). No spaces are allowed.

When compilation is complete, you will be notified of any errors. If errors exist, you must locate and correct them in the source program and recompile the program before you run it. If the compiler detected no errors, the message

```
No Errors or Warnings
```

will appear, and you may proceed.

If fatal errors were detected, MS-COBOL will set the MS-DOS exit code to 255. This can be tested in a batch file with the MS-DOS "IF ERRORLEVEL" command. See Chapter 5, "Batch Command Files," for more information.

3.1.1 Filenames

When you use either of the above methods to invoke the compiler, the source, object, and listing files that you specify can be the name of a disk file and/or a system device. The format is

device path filename extension

device is the name of a system device. This can be a disk drive, terminal, line printer, or other device supported by the operating system. If no device is specified, the current disk drive will be used. If the device is a disk drive, the filename must also be given, unless a default filename is available (see the final example in Section 3.1.2, "Partial Command Strings"). If the device is not a disk drive, only the device name is required. The device may be followed by a colon (:) for readability (it is required only for disk drives). MS-COBOL recognizes the following device names:

NUL	Do not create
CON or USER	Display on terminal
A: or B: ...	Disk drive
PRN or LPT1	Printer
LPT2...	Additional printer(s)
AUX or COM1	RS232

path is a valid MS-DOS path name. See the *Microsoft MS-DOS User's Guide* for information on paths. If *path* is not specified, the current directory will be used.

filename is the name of the file on disk. If the filename is specified without a device, the default disk drive is assumed as the device. Maximum length of the filename is eight characters.

extension is a period (.) followed by a one to three-character suffix to the filename. If an extension is not specified, the following defaults are assumed:

.COB	for the source program file
.INT	for the object file
.LST	for the list file

3.1.2 Partial Command Strings

You may also enter a partial command string when invoking the compiler. Note that the default *object* filename may be specified by entering only the comma which normally follows the filename. Also note that entry of a comma after the object filename causes a default to the source filename. You will be prompted for any files not specified in the command string. For example, the command

```
COBOL CENTER,,
```

would (1) prompt you for the source listing filename (with the default name CENTER.LST), (2) compile the source from CENTER.COB, and (3) produce the object file CENTER.INT.

Each prompt displays its default, which you may accept by pressing RETURN or override by entering another filename or device name.

If you enter an incomplete command string followed by a semicolon (;), default entries will be assumed for the unspecified files.

The following examples assume the compiler is in drive A: and that drive B: has been selected as the default drive. Note that in all the following examples the debug file, CENTER.DBG, will also be produced on the default drive.

1. `COBOL CENTER;`

   Compiles the source from CENTER.COB and produces the object file CENTER.INT. No listing file is produced.

2. `COBOL CENTER,;`

   See Example 1.

3. `COBOL CENTER,,;`

   Compiles the source from CENTER.COB and produces the files CENTER.INT and CENTER.LST. (The second comma tells the compiler to use the source filename as the default list filename.)

4. `COBOL CENTER,,CON`

   Compiles the source from CENTER.COB and places the source listing file on the terminal. The object program is CENTER.INT.

5. `COBOL CENTER,CENTOBJ,PRN`

   Compiles the source from CENTER.COB, sends the list file to the printer, and places the object in CENTOBJ.INT.

6. `COBOL A:CENTER,CENTOBJ,A:`

   Compiles CENTER.COB from drive A:, places the object into CENTOBJ.INT on drive B:, and places the listing into CENTER.LST on drive A:.

3.1.3 Using Compiler Switches

You can add one or more switches to the compiler command string or at the end of any interactive response. A switch is indicated by a slash (/). Square brackets ([ ]) delimit optional entries.

The format for a command string with switch(es) is:

drive`:COBOL` *command string/switch(es)*

Switches

/C[*drive***[:]][***directory***]**

Ordinarily, the compiler looks for the five overlay files (COBOL0.OVR through COBOL4.OVR) first in the current directory, then in the directories specified in the MS-DOS PATH command. The /C compiler switch specifies the drive and/or directory in which the overlays will be found. If a single character is specified after /C, as in "/CA", the character is assumed to be a drive name, and the current directory of the disk in that drive is used. For example,

```
A:COBOL TEST.COB/C\USR\LEON\WORK
```

would direct the compiler to the first overlay in file

A:\USR\LEON\WORK\COBOL0.OVR

/D

This switch suppresses generation of both the debug information file (with a .DBG extension) and source line numbers, which are normally placed in the object file. The result is a shortened (by 16 percent) PROCEDURE DIVISION code. However, when this switch is used, the runtime system will be unable to note the line number at which an error occurs. (See Chapter 8, "Interactive Debug Facility," for more information.) In this example,

```
A:COBOL CENTER/D;
```

the object file will contain no source line numbers and CENTER.DBG will not be produced.

/Fn

Fn (FIPS) flagging lets you tell the compiler to output a warning for each COBOL statement above the Federal Information Processing Standard level (n). n must be a digit from 0 through 4 (4 is the default):

0 Flag everything above low level.

1 Flag everything above low-intermediate level.

2 Flag everything above high-intermediate level.

3 Flag everything above high level.

4 No flagging.

In this example,

```
A:COBOL CENTER/F1;
```

the compiler will display a warning for each COBOL statement above low-intermediate level. If you create a source listing file, the warning will be included with the error messages.

/O (letter O)

This switch restores the default (old) tab settings of 7, 11, 19, 27, 35, 43, 51, 59, 67, and 72 that were used in previous versions of MS-COBOL. See Appendix C, "Tab Stop Customization," for more information.

/Pn

This switch sets the compiler listing page length to the value of n lines. n must be an integer from 10 to 100. The default page length is 66 lines.

/S[*drive*[:]][*directory*]

If the /S switch is used on the MS-COBOL Compiler command line, the default directory and default drive may be changed for COPY file searches.

For example, suppose that your copy file, FILE1, is in directory \TEST on drive B: (not the current drive). The command line that provides the compiler with the initial information about FILE1's location is

```
A:COBOL SAMPLE/SB:\NEW
```

SAMPLE.COB includes this COPY statement which gives the final information about FILE1's location:

```
COPY TEST\FILE1.
```

The compiler finds FILE1 by constructing this path using the following elements:

— the command line specified drive designator, B:

— the command line specified path, \NEW

— the COPY statement specified path, TEST

— the filename, FILE1

The final search path on which FILE1 is found is

```
B:\NEW\TEST\FILE1.
```

See Chapter 6, "COPY Files and Libraries," for more information on COPY files.

/T*drive*

This switch lets you specify the location of the compiler's intermediate file. (Default is the current directory on the default drive.) The disk in the drive you specify must not be write protected.

This option is particularly helpful for compiling very large programs on systems with more than two drives (see Section 3.3, "Compiling Large Programs").

In this example,

```
A:COBOL CENTER,,A:CENTER/TA
```

the intermediate file is placed in the current directory on drive A:. A colon is not allowed in the switch. (This example assumes the default drive is B:.)

3.2 The Source Listing File

The source listing file is a line-by-line account of the source file(s) with page headings and error messages. Each source line is preceded by a line number, which will be referenced by any error messages pertaining to that source line.

Files included in the compilation through source file COPY statements are also included in the listing.

Compiler error messages are shown at the end of the listing file and displayed on the terminal. See Appendix F, "Error Messages," for a listing and explanation of error messages.

3.3 Compiling Large Programs

Occasionally, an MS-COBOL program may be too large to compile in the available memory space or may exhaust the available disk space. There are several ways to take care of this problem:

1. Use the /D switch in your command string (see Section 3.1.3, "Using Compiler Switches") to prevent generation of a debug information file and suppress generation of line numbers in the object file.
2. Use the /T switch in your command string (see Section 3.1.3, "Using Compiler Switches") to place the intermediate files on a separate disk.
3. Place the MS-COBOL Compiler (COBOL.EXE) and its overlays (COBOL0.OVR through COBOL4.OVR) on two separate disks. Then load each portion into memory only as needed:
    a. With the program disk in drive B:, place the COBOL.EXE disk in drive A: and invoke the compiler by typing "A:COBOL".
    b. When you receive the first prompt "Source filename [.COB]:", take out the COBOL.EXE disk and place the overlay disk in drive A:. Then respond to the compiler prompts as usual.

This method allows the space normally used by COBOL.EXE to be available for the intermediate files.

4. Break the program into several program modules. These modules can be separately compiled and then run as a single program system. See Chapter 10, "Interprogram Communication," for information on using program modules.

5. Break the large program into several smaller chained programs. These programs are separately compiled. See Chapter 10, "Interprogram Communication," for information on chaining programs.

6. Check the contents of your disk to make sure that the compiler's intermediate files, STX.$$$, DTA.$$$, and LNK.$$$, have been deleted. Use the DIR operating system command for this task. (Refer to the *Microsoft MS-DOS User's Guide* for more information.)

   If these files were not deleted at the end of compilation because of abnormal termination, you can delete them by typing "DEL *.$$$".

   To make sure that the space has been released, use the CHKDSK program supplied with the Microsoft Disk Operating System. CHKDSK reclaims available space from unclosed files and tells you the total amount of available space on the disk.

Chapter 4

Loading and Executing Microsoft COBOL Programs

Once your Microsoft COBOL program has been compiled successfully, use the runtime executor (RUNCOB.EXE) to load and execute your program. Before you begin, be sure RUNCOB.EXE is on one of the disks in your system.

To run your program, enter RUNCOB followed by the name of your object file (omit the .INT extension). For example, to run the demonstration object file CENTER.INT you would type

```
A:RUNCOB CENTER
```

Execution of CENTER.INT would begin immediately.

If any nonrecoverable errors are detected during execution, the MS-DOS exit code will be set to 255. This can be tested with the MS-DOS "IF ERRORLEVEL" command. See Chapter 5, "Batch Command Files," for more information.

4.1 Finding .INT Files

The runtime executor normally uses your MS-DOS PATH environment to find your object (.INT) file. So, if your PATH were

```
".\BIN;\USR\BIN;\USR\LEON\BIN;USR\COBOL\BIN"
```

the runtime executor would check your current directory, and then each of the directories specified in your PATH to find the .INT file. There are two exceptions to this rule:

1. If your object file begins with the explicit path symbol (\), or a drive designator (letter followed by a colon (:)), only the directory specified by this path is searched. For example,

   ```
   RUNCOB \TEST\TEST1
   ```

   finds the run file TEST1.INT in the directory TEST.

2. If the /S runtime switch is used in conjunction with a valid pathname, that pathname becomes the default search path. For example,

```
RUNCOB TEST/SB:\USR\LEON\WORK\BIN
```

finds TEST.INT on the search path

B:\USR\LEON\WORK\BIN.

These searching conventions also apply to the loading and execution of files called by the MS-COBOL CALL statement and to the files loaded by the CHAIN statement.

4.2 Using Runtime Switches

You can add one or more switches to the runtime command line, immediately after the object filename. The switches and their parameters must be contiguous (without intervening spaces).

The format for a runtime command string with switch(es) is

[*drive*:]RUNCOB *object-file/switch(es)*

Switches

/P Causes output files assigned to PRINTER to be appended instead to file PRINT.SPL, in the current drive and directory. For example,

```
RUNCOB TEST1/P
```

sends a PRINTER file to file PRINT.SPL.

/S[*drive*[:]][*directory*]

Provides a replacement search path for your object files (.INT files). The runtime executor will search the specified path if the .INT file is not in the current directory. See Section 4.1, "Finding .INT Files," for more information.

/V Changes the Format 2 ACCEPT statement (described in Section 7.6.1.2 of the *Microsoft COBOL Reference Manual*) to treat numeric receiving fields as alphanumeric, in conformance with the ANSI-74 COBOL standard.

Chapter 5
Batch Command Files

MS-DOS supports your use of batch command files to compile and run your Microsoft COBOL programs.

For example, you could use the EDLIN editor to create a batch file CLGO.BAT ("compile, load, and go") containing the following text:

```
B:
COBOL %1,,;
PAUSE     Insert RUNCOB.EXE in drive A:
RUNCOB %1
```

The first line of the batch file will compile the program, producing a listing file; the second will cause a pause followed by the reminder to insert the runtime disk; the third runs the compiler-generated file.

To run the batch file, you would type

CLGO *sourcefile*

sourcefile is the name of the source program you want to compile, load, and run.

You may use symbols in the batch file to represent command line parameters. In this case, the symbol "%1" refers to the first parameter, *sourcefile*, on the CLGO command line.

MS-DOS supports the need to pause, display a prompt, and wait for you or an operator to continue. The PAUSE command, followed by the user-defined text of the prompt, performs this function.

The MS-DOS exit code will be set to the value of 255 if the MS-COBOL Compiler or runtime system encounter fatal errors. It may also be set to a value from 0 to 255 with the EXCODE extension subroutine, described in Section 10.5,

"Calling MS-COBOL Extension Subroutines." The exit code may be tested in batch files by using the "IF ERRORLEVEL" command to stop a long batch job if errors are found.

For example, in the CLGO batch file you could test for compile errors before running by adding an IF ERRORLEVEL statement:

```
B:
COBOL %1,,;
IF ERRORLEVEL 255 GOTO END
PAUSE     Insert RUNCOB.EXE in drive A:
RUNCOB %1
:END
```

Note

Your batch file must have the extension .BAT and should be kept on either your program disk or the utility disk. A .BAT filename must be unique: if the .COM file or .EXE file have the same name as .BAT, the batch will not be executed.

For more information about batch command files, see your *Microsoft MS-DOS User's Guide.*

Chapter 6
COPY Files and Libraries

This chapter describes how a COPY statement file-name may be qualified by a library-name, and how it may be found using the library-name and an MS-DOS drive designator and path.

The COPY statement is a compiler-directing statement rather than an executable statement. It logically embeds the text of a disk file (other than the source file) in the source program. It may be used anywhere in the ENVIRONMENT, DATA, or PROCEDURE DIVISIONS. The general format of the COPY statement is

COPY *file-name* [{OF | IN} *library-name*]

Note

The following discussion also applies to COPY statements that are modified with the REPLACING phrase.

6.1 Drive Designators and Paths

An MS-DOS drive designator is a letter followed by a colon; for example, A:. It specifies the disk drive on which a file will be found. If no drive designator is specified in a file-name, the current drive will be used during a file search.

A path is a series of directory names separated by backslashes (\). If the first character in the path is a backslash, the search path begins at the root directory; otherwise it begins at the default directory (usually your current working directory). For example:

\TEST\COBOL	starts from the root directory.
NEW\COBOL	starts from the default directory (if your default directory were "USER", the full path would be \USER\NEW\COBOL).

The default directory for COPY files is normally the current directory on the current drive. A /S switch, if used on the MS-

COBOL Compiler command line, changes the COPY file default directory for a specific drive. If no drive designator is specified on the command line, the default directory for the current drive is changed.

See your *Microsoft MS-DOS User's Guide* for more information about drive designators, tree-structured directories, and paths.

6.2 Specifying Paths to COPY Files

If no drive designator or path is given for a filename, the MS-COBOL Compiler searches the default directory for a COPY file. This default directory is normally the current directory on the current drive, so when the compiler encounters a COPY statement such as

```
COPY FILE1.
```

it searches the current directory for FILE1. If the COPY filename is defined with a path

```
COPY \TEST\FILE1.
```

the compiler will search the directory TEST for FILE1 on the current drive.

If the /S switch is used on the MS-COBOL Compiler command line, the default directory and default drive may be changed for COPY file searches.

For example, suppose that your copy file, FILE1, is in directory \TEST on drive B: (not the current drive) and that \TEST is also a subdirectory of \NEW\COBOL. The MS-COBOL command line that informs the compiler of the initial information about FILE1's location is

```
COBOL SAMPLE/SB:\NEW\COBOL
```

SAMPLE.COB includes this COPY statement which gives the final information about FILE1's location:

```
COPY TEST\FILE1.
```

The compiler finds FILE1 by constructing this path using the following elements:

— the command line specified drive designator, B:

— the command line specified path, \NEW\COBOL

— the COPY statement specified path, TEST

— and the file-name, FILE1

The final search path on which FILE1 is found is

```
B:\NEW\COBOL\TEST\FILE1.
```

In searching for file names, the MS-COBOL compiler will first look for the name exactly as specified (e.g. FILE1). If no file with that name is found, and the specified file name contains no extension, the extension ".COB" will be added to the file name, and a search will be made for the new file name. For example, if FILE1 is specified, both FILE1 and FILE1.COB will be tried. If FILE1.CPY is specified, only that file name will be used.

Examples — COPY Files With the Current Directory Used as the Default Directory

For the following examples, assume that

1. The current drive is A:.
2. The current directory on drive A: is \TEST.
3. The current directory on drive C: is \TEST2.
4. The COBOL command line is

   ```
   COBOL SAMPLE2
   ```

   (No default directory is specified on the command line.)

In the following table, each COPY statement gives the compiler some or all of the path information needed to find FILE1. The compiler will use the specified file-name (lefthand column) and the current drive and current directory values to produce the full file-name (righthand column). Using any of these file-names, the compiler will find FILE1.

COPY statement	Resulting file-name
COPY FILE1.	A:\TEST\FILE1
COPY COBOL\FILE1.	A:\TEST\COBOL\FILE1
COPY \COBOL2\FILE1.	A:\COBOL2\FILE1
COPY C:FILE1.	C:\TEST2\FILE1
COPY C:COBOL\FILE1.	C:\TEST2\COBOL\FILE1
COPY C:\COBOL2\FILE1.	C:\COBOL2\FILE1

In the first two examples, both the drive and current directory were picked up from the current system values, A: and \TEST. (In the third example, only the current drive was picked up.)

In the fourth and fifth examples, the current directory for drive C:, \TEST2, was used, and the specified drive superseded the current drive A:. In the last example, where both the drive designator and a path from the root directory were specified, the file-name used was exactly the same as the file-name specified in the COPY statement.

Examples — COPY Files With a Specified Default Directory

Suppose the COBOL command line were

```
COBOL SAMPLE3/SB:\NEW
```

specifying the default drive B:, and the default directory \NEW. The resulting filenames for FILE1 would be

COPY statement	Resulting file-name
COPY FILE1.	B:\NEW\FILE1
COPY COBOL\FILE1.	B:\NEW\COBOL\FILE1
COPY \COBOL2\FILE1.	B:\COBOL2\FILE1
COPY C:FILE1.	C:\TEST2\FILE1
COPY C:COBOL\FILE1.	C:\TEST2\COBOL\FILE1
COPY C:\COBOL2\FILE1.	C:\COBOL2\FILE1

The default path and drive designator are used only when the COPY file-name specifies neither a drive designator nor a path starting from the root directory. When specified, the values in file-name override the default drive and directory. The specified default directory affects only drive B:, the default drive

used in the first three examples. Drive C:, specified in the last three examples, still uses its own default directory, \TEST2.

6.3 Library-Names as Paths to COPY Files

COBOL allows file-names used with the COPY statement to be qualified by a library name. Under MS-COBOL for the MS-DOS operating system, library names are implemented as part of the MS-DOS subdirectory structure.

When a COPY statement in the following format is encountered with a library-name specified:

COPY *file-name* [{OF | IN} *library-name*]

the compiler takes library-name to be the immediate directory of file-name.

If file-name contains a drive designator, a library-name may not be specified. If file-name contains a path from the root directory, only library-names consisting entirely of a drive designator are allowed.

Full paths to COPY files using library-names are built by first prepending library-name, then a backslash (if required) to file-name. The resulting filename is then modified as necessary by the default drive and directory.

Examples — COPY Files With Library-Names

For the following examples, assume that

1. The current drive is A:.
2. The current directory on drive A: is \TEST.
3. The current directory on drive C: is \TEST2.

4. The COBOL command line is

   ```
   COBOL SAMPLE4
   ```

   (No default directory is specified on the command line.)

In the following table, each COPY statement gives the compiler some or all of the path information needed to find FILE1. The filenames are all specified without drive designators or paths.

COPY statement	Resulting file-name
COPY FILE1 OF LIB1.	A:\TEST\LIB1\FILE1
COPY FILE1 OF COBOL\LIB1.	A:\TEST\COBOL\LIB1\FILE1
COPY FILE1 OF \COBOL2\LIB1.	A:\COBOL2\LIB1\FILE1
COPY FILE1 IN C:.	C:\TEST2\FILE1
COPY FILE1 IN C:LIB1.	C:\TEST2\LIB1\FILE1
COPY FILE1 IN C:COBOL\LIB1.	C:\TEST2\COBOL\LIB1\FILE1
COPY FILE1 IN C:\COBOL2\LIB1.	C:\COBOL2\LIB1\FILE1

Note that library-name may consist of only a drive designator.

If a default directory is specified in the command line, or filename contains a drive designator or path, the resulting filename is built according to the rules outlined in Section 6.2, "Specifying Paths to COPY Files."

Chapter 7
Data Input and Output

A Microsoft COBOL program can read or write data to files on disk or to other MS-DOS devices. The instructions for creating and using these files are entered as part of the MS-COBOL source program. This section explains disk files and other types of files, and tells you how to use them with your MS-COBOL programs. See the *Microsoft COBOL Reference Manual* for more information.

7.1 Using Disk Files

To specify that a disk file is to be used in a program, include the ASSIGN TO DISK clause in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

The file-name of the disk file must be declared in the VALUE OF FILE-ID clause in a File Description (FD) paragraph, in the FILE SECTION of the DATA DIVISION. The FD paragraph must also include the clause LABEL RECORDS ARE STANDARD. BLOCK clauses are checked for syntax, but they have no effect on any file type. If a disk file is to be used, the FILE-ID clause should not be given an MS-DOS device name. (See Section 7.3, "Using MS-DOS and Nondisk Files," for a list of MS-DOS device names.) This would cause the file to appear on the specified MS-DOS device rather than on a disk drive.

There are four types of disk file organization:

Sequential
Line Sequential
Relative
Indexed

Disk files are assumed to be Sequential unless they are declared otherwise in the ORGANIZATION clause in the FILE-CONTROL paragraph of the program's ENVIRONMENT DIVISION.

Note also that only Line Sequential files can be created with an editor. All others must be created by an MS-COBOL program or assembly language program. See the *Microsoft COBOL Reference Manual* or one of the tutorials recommended in Section 1.4, "Learning More About COBOL," for more information about creating disk files.

7.2 Disk File Organization

The following paragraphs describe the four types of disk files. (All formats are subject to change without notice.)

1. Sequential files have a two-byte count of the record length, followed by the actual record, for as many records as are in the file.
2. In Line Sequential files, each record is followed by a carriage return/line feed pair as the delimiter for as many records as are in the file. No COMP-0, COMP-3, or COMP-4 fields should be written into a Line Sequential file because these data-items may contain the same binary codes used for carriage return, line feed, and end-of-file. This would cause problems in reading the file.

Note

If a Sequential or Line Sequential file created by a program other than MS-COBOL version 2.0 or later is to be OPENed with the EXTEND option, the file must first be copied using one of the following methods:

a. a program written in MS-COBOL version 2.0 or later, or

b. a utility that will produce a file with MS-DOS 2.0 end-of-file format

This will ensure that the data file end-of-file format will be acceptable to MS-COBOL.

Warning

Files created by line editors and non-COBOL programs are often in Line Sequential format. If you wish to use such a file as input to an MS-COBOL program, you must include the ORGANIZATION IS LINE SEQUENTIAL clause in its FILE-CONTROL paragraph. If the clause is not included, MS-COBOL assumes the file is in Sequential format, and stops with a runtime error when the Line Sequential file is input.

3. Relative files always have fixed-length records of the size of the largest record defined for the file. Since no delimiter is needed, none is provided. Deleted records are filled with binary zeros. Additionally, six bytes are reserved at the beginning of the file to contain system bookkeeping information.

4. Each Indexed file declared in an MS-COBOL program will generate two disk files: a key file and a data file. The file specification in the VALUE OF FILE-ID clause specifies a file containing data only. The file-name included in the file specification is concatenated with an extension .KEY to form the file specification of the key file.

   The key file contains keys, pointers to keys, and pointers to data. The format of this file is very complicated, but follows the guidelines for a prefix B+ tree.†

   The data file consists of data records and a data dictionary that describes the data fields contained in the file.

†See Douglas Comer, "The Ubiquitous B-Tree," in *Computing Surveys of the ACM*, Vol. 11, no. 2 (June 1979), pp. 121-137.

7.3 Using MS-DOS and Nondisk Files

You do not need to place files on a disk if they will only be output. You can consider these files as a stream of characters going to a printer or other device; no permanent file need be created, and no extra characters are needed in the record for carriage control. Carriage return, line feed, and form feed are sent to the output device between lines. Note, however, that blank characters (spaces) on the end of a print line are truncated to improve printing speed.

To send an output file to the printer, use the SELECT *filename* ASSIGN TO PRINTER clause. Then, in an associated FD, specify the clause LABEL RECORD IS OMITTED. Do not specify the VALUE OF FILE-ID clause.

MS-DOS provides special device names for character devices. Data may be sent to or read from

CON or USER	display on terminal
AUX or COM1	serial port (RS232)
PRN or LPT1	printer
LPT2...	additional printer(s)

If you assign these names to the VALUE OF FILE-ID clause, MS-COBOL treats the files as disk files (see Section 7.1, "Using Disk Files"). That is, you assign the files to disk with the SELECT clause, but the operating system uses the designated device instead of a disk drive.

ORGANIZATION IS LINE SEQUENTIAL should be specified in the FILE-CONTROL paragraph for files being sent to MS-DOS devices.

Chapter 8
Interactive Debug Facility

The MS-COBOL Interactive Debug Facility allows you to control the execution of a program and to examine or change data-items in an MS-COBOL program. When a program is compiled, a "debug information file" is created along with the object file, unless you have suppressed its creation with the /D compiler switch. (see Section 3.1.3, "Using Compiler Switches").

This information file contains line numbers and data-names from the DATA DIVISION and PROCEDURE DIVISION of your MS-COBOL program. The debug commands listed in Section 8.2, "Debugging Commands," can use these line numbers and data-names to affect data-items and program execution in a number of ways.

The compiler will create the debug information file with the file-name of the MS-COBOL object program plus .DBG extension. For example, compilation of a source file named MYFILE would produce MYFILE.INT (object file) and MYFILE.DBG (debug information file).

The debug facility, DEBUGCOB.EXE, should be configured to your terminal using the same INSTALL utility option that is used with RUNCOB.EXE. Prepare the INSTALL files as described in Chapter 9, "INSTALL Program," type INSTALL DEBUGCOB.EXE, then select the option desired. Failure to run INSTALL will cause the debug facility commands to be overprinted, but the facility may still be used.

8.1 Using the Debug Facility

To use the debug facility DEBUGCOB.EXE, type DEBUGCOB in place of RUNCOB, and add the name of the file to be run. This invokes the interactive debugging executor instead of the runtime executor. For example, to debug the file CENTER.INT, type

```
DEBUGCOB CENTER
```

and the following message will appear:

```
MS-COBOL Interactive Debug Facility v. xxx
Program: CENTER
  Type help for list of commands
*
```

The asterisk prompt (*) indicates that the debug facility is ready to accept any of the debug commands listed in Section 8.2, "Debugging Commands." The debug information file should be on the current disk and directory. If it is not, the message

```
No debug information file found
```

will follow the messages already displayed.

Note that without a debug information file, limited debugging is possible. By simply running your program with DEBUGCOB, you can enable the debug facility and execute any of the debug commands listed in Section 8.2 except Change, Exhibit, and Goto *line-number*. However, without the debug information file, the debug facility cannot verify that line numbers specified in the breakpoint command (see Section 8.2, "Debugging Commands") are valid PROCEDURE DIVISION line numbers that contain statements, or section or paragraph names.

8.2 Debugging Commands

Debug commands may be typed in full or may be abbreviated to the first letter of the command name (shown in boldface). Uppercase and lowercase characters are equivalent. Arguments to the commands (line numbers, data-names, ALL, OFF) must be given in full. Arguments can be separated from commands by any nonalphabetic character: spaces are used in these examples. When a numeric argument is expected, the debug facility will scan to the first digit on the line. For example, the following commands are equivalent (i.e., set a breakpoint at line 100):

```
Breakpoint 100
BREAK @ 100
b100
break for me at line 100, if you would, please
```

Pressing your terminal's interrupt key suspends program execution at the next statement, as if a breakpoint had been set at the next line. The key used as the interrupt key may vary according to your type of terminal, but is usually CONTROL-C, ALT-C, or CTRL-BREAK.

The following functions are available with the Interactive Debug Facility:

Address *data-name*

Displays absolute address (hexadecimal) of a data-item in memory.

Breakpoints

Lists all breakpoints. (A breakpoint is a point at which execution is interrupted so that you can insert a debug command.)

Breakpoint *line-num*

Sets breakpoint at *line-num*. You may have up to eight breakpoints set at any given time. The Debug Facility verifies that *line-num* is a PROCEDURE DIVISION line that contains a statement or paragraph name.

Change *data-name*

Displays the contents of *data-name* and allows a new value to be entered. Change cannot be used on index-names or on subscripted or qualified variables.

Dump [*data-name-1*[*data-name-2*]]

Displays memory addresses (hexadecimal equivalents) from the start of *data-name-1* through the start of *data-name-2*.

Both data-names are optional. If *data-name-2* is omitted, 128 bytes are dumped, starting at *data-name-1*. If both data-names are omitted, 128 bytes are dumped, starting at the last data-name dumped.

If *data-name-1* is specified, Dump locates the data-item specified, and begins the dump from its starting location.

In addition to using data-names, explicit addresses may be used, replacing *data-name-1* with *addr1* and *data-name-2* with *addr2*. *addr1* and *addr2* are hexadecimal numbers preceded by

the "at" symbol (@). For example, @1A0E is a valid value for *addr1* or *addr2*.

Exhibit *data-name*

Displays contents of *data-name*.

Data-items of less than 77 characters are displayed within brackets. For data-items greater than 77 characters, the field length of the contents is displayed without brackets.

Group names may be displayed, although some components (e.g., binary characters) may not be displayable.

Exhibit cannot be used on index-names or on subscripted or qualified variables.

Go

Resumes execution from the last breakpoint or current program position until a breakpoint or end-of-program is encountered.

Goto *line-num*

Begins execution at *line-num*; continues until breakpoint or end-of-program is encountered.

This command may be used to branch anywhere within a program, even from one overlay segment to another.

If a PERFORM is active when Goto is issued, the debug session may terminate.

Help

Displays the list of debug commands.

Kill *line-num*

Removes the breakpoint at *line-num*.

Kill ALL

Removes all breakpoints from the breakpoint list.

Line

Displays the current *line-num*.

Quit

Terminates the program (closing all open files).

Step [*count*]

Executes one or *count* statement(s).

Trace

Sets trace mode. When trace is set, each line number will be displayed as the line is executed.

Trace OFF

Turns off trace mode. (See Trace.)

8.3 Debugging Subprograms

The Interactive Debug Facility allows you to debug systems of programs consisting of a main program and any number of subprograms. However, there are some limitations on what can be debugged in such a system:

1. Assembly language subroutines may be called, but none of the debugging features will be in effect while the subprogram is executing. For example, no breakpoints can be set in an assembly language subroutine.
2. If subroutines are nested to more than five levels without a return to an earlier subprogram, the debug facility will not open the debug files for subprograms beyond the fifth: the message "No debug information file found" will be generated, even though the information file may actually be present. You may still set breakpoints and use the trace mode at these deeper levels of nesting, but you may not examine or change variables. Upon return to subprograms nested less than five levels deep, the full debug facilities will again be available.

   This limitation does not hold for systems where a program calls a large number of subprograms but returns to the main program before calling each subprogram.

Chapter 9
INSTALL Program

Microsoft COBOL can define screen attributes and have these attributes and other screen definitions displayed on your terminal's screen in an interactive mode.

This capability comes from Microsoft extensions to the DATA DIVISION (SCREEN SECTION) and the PROCEDURE DIVISION ACCEPT and DISPLAY statements (see the SCREEN SECTION in Chapter 6, "DATA DIVISION," and Formats 1, 3, and 4 of the ACCEPT and Format 3 of the DISPLAY statements in Chapter 7, "PROCEDURE DIVISION," of the *Microsoft COBOL Reference Manual*).

This chapter tells you how to configure the MS-COBOL runtime system using the INSTALL program.

9.1 Overview of INSTALL

In order for the Microsoft COBOL extensions for interactive screen-handling and keyboard interpretation to run correctly on your terminal, the MS-COBOL runtime system needs to be configured to the characteristics of your system.

INSTALL is run once to reconfigure a specified runtime executor (RUNCOB.EXE, DEBUGCOB.EXE, or any custom runtime executor created for assembly language interface) to the characteristics of your terminal. See Chapter 10, "Interprogram Communication," for a discussion of the assembly language interface that is created by customizing the new MS-COBOL runtime executor. INSTALL can be run more than once if you wish to change some of your terminal's function key assignments.

When you run INSTALL, the necessary information about the screen-handling characteristics of your terminal is placed in the common runtime executor (RUNCOB.EXE or another specified file). This information comes from a data file which contains terminal descriptions.

If you have a serial terminal that is not listed in the data file, you can respond to a series of questions, and your answers will then be put into the common runtime system.

Throughout this chapter, the word "terminal" will refer to the components of your computer system affected by INSTALL. A terminal consists of the keyboard and screen, either separate from, or part of a host computer, and any hardware or software used to communicate with the keyboard or screen.

Terminals may be described as "serial" or "nonserial." Serial terminals perform screen functions (e.g., moving the cursor or turning highlighting on or off) by interpreting special character strings sent from the host computer. Nonserial terminals perform screen functions by executing special assembly language routines rather than by interpreting character strings. These assembly language routines differ from machine to machine, and are inserted by INSTALL into the Microsoft COBOL runtime module.

Note

Many traditionally nonserial terminals and computers can be treated as serial terminals, because the operating system will interpret character sequences as output and perform the corresponding screen functions. Refer to your system's documentation for specific information.

When you run the INSTALL program, it shows a list of terminals that are already defined in the INSTALL.DAT file (the input file for the INSTALL program). If your terminal is on the list, you can quickly modify MS-COBOL by selecting your terminal from the list. INSTALL then automatically loads and saves the information from your selection. When INSTALL is finished, you are ready to run MS-COBOL programs.

If your terminal is not included on the list, INSTALL provides you with a list of questions for defining a serial terminal. You will need the technical manual for your terminal to answer most of the questions. Your answers are then collected into the INSTALL.DAT file (see Appendix D, "INSTALL Terminal Database," for the terminal characteristics known to Microsoft COBOL). Answers collected in INSTALL.DAT can be modified later if some answers are incorrect or unsatisfactory. See Section 9.3, "Defining a Terminal," for details.

Warning

You may define only a serial terminal in this manner. Nonserial terminals not on the terminal list in Appendix D cannot take advantage of the MS-COBOL screen extensions unless a serial driver for one of the terminals on the list is provided by the terminal manufacturer. The ANSI-standard terminal driver is often provided.

Throughout the program, INSTALL displays explanatory text to help you complete the answer file.

To run INSTALL, you need these files contained on the MS-COBOL distribution disks:

INSTALL.COM	The INSTALL program file.
INSTALL.MSG INSTALL.OVL INSTALL.SPC	The files needed by the INSTALL program.
INSTALL.DAT	The file that contains descriptions of common terminals. If INSTALL lists your terminal, INSTALL.DAT contains a description of your terminal.

The MS-COBOL files to be modified include RUNCOB.EXE, DEBUGCOB.EXE, and, potentially, any custom runtime executors created by linking in assembly routines.

INSTALL consists of the steps shown in Figure 9.1. Specifically, these steps are:

1. Starting INSTALL and determining if a description of your terminal is included.
2. Answering questions about your terminal if it is not included in INSTALL.
3. Reviewing and editing your answers to the questions in Step 2.

4. Running the terminal tests.

Each of the steps is described more fully in Sections 9.2 through 9.5.

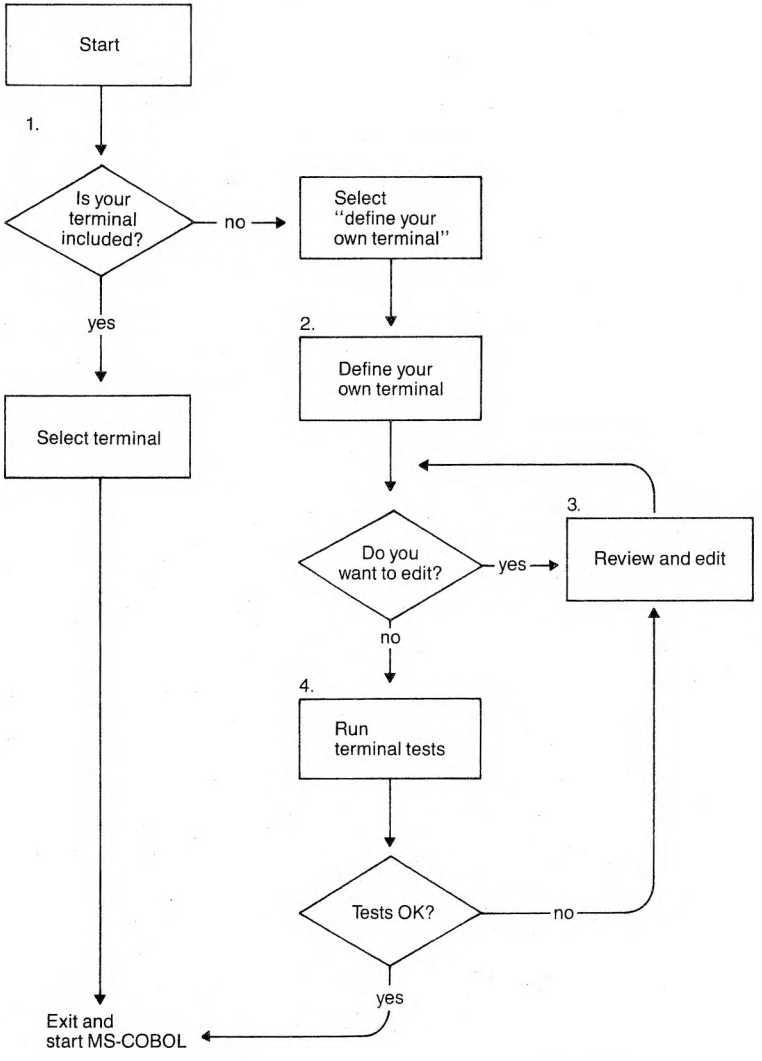


Figure 9.1 INSTALL Program Steps

9.2 Starting INSTALL: Is Your Terminal Included? (Step 1)

Before starting the INSTALL program, make sure that MS-DOS is up and running. Then put your program disk containing the INSTALL files in drive A: and make sure that the file to be reconfigured is on this or another disk in your system.

To start the INSTALL program, type

INSTALL *filename*

then press RETURN. *filename* is the name of the file to be reconfigured with your terminal's characteristics. If *filename* is not specified, RUNCOB.EXE on the default drive is assumed.

Once the INSTALL program and the necessary files are loaded into memory, the screen gives instructions about using the basic keys you will need. These basic keys are:

1. CTRL-C

   to terminate the INSTALL program at any time

2. BACKSPACE

   to correct your answers

3. RETURN

   to proceed with the INSTALL program at any time. Press RETURN after each answer.

When you have read the initial instructions, press RETURN and INSTALL displays the list of terminals from INSTALL.DAT. The terminals listed are the ones for which INSTALL already has data. INSTALL then asks if your terminal is listed.

If your terminal is listed, enter the number corresponding to your terminal and press RETURN. INSTALL finishes the program automatically, adding the description of your terminal to the file specified. When INSTALL is finished, the message "Install complete" appears. At this point, you can copy the "installed" runtime executor to your utility disk, and skip

Steps 2, 3, and 4. (Steps 2, 3, and 4 are for defining a terminal not on the list.) Then you can begin running your MS-COBOL programs. The demonstration program CRTEST may now be compiled and run. (See Appendix E, "Guide to the MS-COBOL Demonstration Programs," for information on CRTEST, and Section 2.4, "Sample Session," for instructions on compiling.) CRTEST tests several of the screen features established by INSTALL.
If your terminal is not listed, press *1* ("define your own terminal") and RETURN. This starts Step 2, in which you define a serial terminal. Step 2 is applicable only if you have a serial terminal system.

Note that some terminals "waste" or use up a character for turning screen attributes on and off by filling a character position on the screen with a special marker. This marker appears as a space on the screen when the screen attribute is turned on or off.

Some of these terminals, such as the TeleVideo®, are predefined in the INSTALL data file. For these terminals, the INSTALL program does not use the screen attributes such as blink, underline, highlight, and reverse-video, to avoid confusing the compiler's internal mapping of what is on the screen.

However, some MS-COBOL programs may use these screen attributes despite the wasted characters. If you are sure that your programs will run with this condition, you may want to try the "define your own terminal" selection (number 1 on the menu). This allows you to use your terminal's screen attributes rather than using those in the INSTALL data file.

9.3 Defining a Terminal (Step 2)

This step is applicable only if you have a serial-mapped terminal.

When you press *1* and RETURN, INSTALL displays

```
Would you like to redefine this terminal (Y/N?)
```

If you press *N* (for no), INSTALL automatically finishes the program and loads the default terminal characteristics into the specified runtime executor file.

Important

The initial default values (see Table D.2, "Default System Interface") in the INSTALL.DAT file are for a general purpose terminal and probably do not apply to your terminal. Consult your terminal's technical manual for the correct values.

If you press *Y* (for yes), INSTALL will ask if you want to go through the questions sequentially or if you want to use the shorter menu form. Both forms provide default answers.

The sequential questions are displayed one at a time, along with their default answers. If you are using INSTALL for the first time, we recommend using the sequential form to reduce the chance of skipping an important question.

The menu form is for reviewing and selecting individual questions. It displays some of the questions by groups. For example, it displays MS-COBOL function keys as a group instead of showing all the individual function keys. Once you choose the function key selection, INSTALL will display all of the function key questions sequentially.

INSTALL asks questions about key assignments and terminal characteristics. Key assignment questions ask which keys are assigned to the different MS-COBOL functions such as character delete, forward space, Function 1, etc. Terminal characteristic questions pertain to the character sequences needed to perform such functions as clearing the screen and initializing the terminal.

When INSTALL asks a question, the default answer is displayed under the question. To accept the default answer, simply press RETURN. If you want to change the answer, backspace over the default answer and enter the new answer. Note

that when INSTALL asks a question, it will prompt you for the type of answer you should enter. The prompt characters are

1. (I) Integer

   Use only number keys for this type of answer.

2. (Y/N) Yes or No

   Answer with Y or N (either uppercase or lowercase).

3. (S) Character string

   Enter a sequence of characters. Special keys can be coded with one of the two prefix characters: ^ and &.

   The ^ is a prefix for coding control characters. (Example: INSTALL asks, "What sequence(s) of characters (S) represents DELETE?" If the answer is CONTROL-U, type "^U".)

   The & is a prefix for coding the characters shown in the following menu. These characters may be typed in lowercase or uppercase. (Example: INSTALL asks, "What sequence of characters (S) starts high intensity?" If the answer is ESCAPE followed by "P", type "&EP".)

   When you see the "S" prompt, you can type "&M" to display a menu of special keys and character sequences that are coded with the & prefix character. The menu will read:

```
&E - escape         &R - return       &^ - ^
&N - newline        &T - tab          && - &
&F - formfeed       &B - backspace    &X - rubout
&, - ,              &+ - +

&Dxxx     - 3-digit decimal (less than 256)
&Oxxx     - 3-digit octal   (less than 0400)
&Hxx      - 2-digit hex
&P&Hxx    - pause xx (hex) milliseconds
&Iy&Dxxx  - pad character "y", xxx times
&Y        - used to code a Y after a CONTROL-C
```

Note

On some terminals, certain commands take longer to execute than others. To compensate for this difference, it may be necessary to make the computer wait until the terminal has finished executing the command. By inserting a pause of xx milliseconds, you can indicate how long it must wait.

The millisecond timing of a pause is for an 8 MHz clock. If you have a 4 MHz clock, divide the value by 2. If you have a 2 MHz clock, divide the value by 4. For example, for a 40 millisecond pause (with a 8 MHz clock), enter 40. To get the same 40 millisecond pause from a 4 MHz clock, enter 20.

Consult your terminal's technical manual to see if such a pause is necessary.

9.3.1 Hints on Defining Key Assignments

To use all of your terminal capabilities, you should understand how INSTALL recognizes character sequences.

All keystrokes send character values to the MS-COBOL program. Some keys, such as *cursor up*, send a multiple character sequence (&E[A). On the other hand, CONTROL-*key* and SHIFT-*key* are considered single keystrokes. Pressing either CONTROL or SHIFT by itself signifies nothing to the program, but when combined with another key, the combination will send a single character. MS-COBOL recognizes both single and multiple-character sequences as well as multiple-key sequences such as "^R+^E" (page up), and multiple-key, multiple-character sequences such as "^R+&E[A".

To take advantage of MS-COBOL's ability to recognize the different types of character sequences, use the following guidelines:

1. Functions requiring a multiple-key sequence are separated with the "+" character between each key character sequence. This tells MS-COBOL to expect more than one keystroke for a given function. For

example, if you want to use the key sequence "CONTROL-R CONTROL-E", you would enter "^R+^E".

2. The first keystroke of a multiple-key sequence cannot be defined as a key sequence of its own. For example, if the character sequence "&E[OM" is a value for a single key, then, "&E[OM+^P" cannot be used as a multiple-key sequence.

   The reason for this rule is that MS-COBOL does not know if it has received the whole command or if it is waiting for another character to finish the command. For example, if a portion of a multiple character sequence is typed as input to an interactive ACCEPT statement, the MS-COBOL program waits for the sequence to be completed before continuing processing.

3. Keys in a multiple-key sequence can consist of multiple characters. For example, the "page up" sequence for certain terminals is "&E[11~+&E[A".

4. When setting up new input key definitions to convert a character or character sequence to your terminal, make sure the changes do not result in conflicting sequences. If there is a conflict, INSTALL will display the message "Your input keys are ambiguous. No two functions may share the same string" when you try to exit INSTALL. If you get this message, check your work and try again.

9.3.2 Screen Attributes

Screen attributes have a hierarchy of use: blink, underline, highlight, and reverse-video. If the terminal doesn't support the attribute requested, it will try the next attribute in the hierarchy. For example, if you request the underline attribute and your terminal doesn't support it, INSTALL uses the next attribute (highlight) instead. If highlight isn't available, reverse-video is used. If reverse-video isn't available, normal video is used.

In order for MS-COBOL to put a character on the screen, it must know the cursor position. On some terminals, a character is used to turn on and off a screen attribute. By using a character to turn on the attribute, the terminal moves the

cursor over one position. The same is true for turning off the attribute. Unfortunately, this characteristic is usually not documented in the terminal's technical manual. "Standouts" are the character or characters to which a screen attribute has been applied. If you notice spaces before standouts, your terminal probably has this characteristic. If it does, disable the screen attributes by entering "blank" answers for the questions about blink, underline, highlight, and reverse-video. (A "blank" answer is an empty string.) If the default answer to the question is "blank," simply press RETURN. If there is a character sequence for the default answer, backspace over the answer, and press RETURN.

When you have seen and answered all questions, you are ready to begin Step 3, reviewing and editing your answers.

9.4 Reviewing and Editing Answers (Step 3)

INSTALL allows you to review and edit all answers before saving your terminal characteristics in the INSTALL.DAT file. Before saving the answers, INSTALL will display a menu of your terminal characteristics with either the default answer (from the previous session) or the answer you supply during this session.

To change an answer on the menu, select the number of the terminal characteristic and press RETURN. INSTALL will then display the question again. When it appears, backspace over the old answer and enter the new answer. Then, press RETURN to keep the new answer.

You may continue reviewing and editing the characteristics until you are satisfied that everything is correct. When you are through reviewing the answers, press *D* and RETURN. You are now ready to run the terminal tests.

9.5 Running the Terminal Tests (Step 4)

The terminal tests let you test your terminal characteristics before you run your programs. The tests aren't mandatory, but we recommend running them to verify your selections. INSTALL offers the terminal tests shown in the following list. Three of the test selections, ("initialization," "graphics characters," and "all of above") are not applicable to MS-COBOL. During the tests, pressing CONTROL-C will return you to the editing menu.

If you don't want to run any of the tests, press *D* and RETURN when you see the test list. INSTALL will save your answers and exit automatically.

Note

The answers you save become the default answers for the INSTALL program.

The tests are

1. cursor positioning
2. clearing the screen
3. initialization
4. function keys
5. screen attributes: blink, underline, highlight, reverse-video
6. cursor and keyclick options
7. sounding the bell
8. graphics characters
9. all of the above

If the terminal tests that you select end successfully, INSTALL adds the description of your terminal to MS-COBOL. Then it

displays the message "Install complete." You are now ready to run MS-COBOL.

If any of the tests fail, you will receive a message on your terminal. Press CONTROL-C to return to Step 2. Check your terminal's technical manual and change any incorrect responses.

Chapter 10
Interprogram Communication

Interprogram communication is accomplished using the CALL or CHAIN statement. CALL temporarily transfers control to a subprogram, and CHAIN permanently transfers control to another program. The communications options available with CALL and CHAIN are

1. Temporary transfer of control from one MS-COBOL program to a subprogram (CALL).
2. Temporary transfer of control from an MS-COBOL program to a non-COBOL subroutine (CALL).
3. Permanent transfer of control from one MS-COBOL program to another (CHAIN).
4. Permanent transfer of control from an MS-COBOL program to a non-COBOL program (CHAIN).

In addition to transferring program control, these statements can transfer data between programs. This is done with the USING and CHAINING phrases. In a CALL statement, the USING phrase lists parameters which give the addresses of data to be acted on within the called program. These data are specified in a corresponding USING phrase in the PROCEDURE DIVISION statements of the called program. The called program makes any necessary changes and then returns control to the calling program.

In a CHAIN statement to an MS-COBOL program, the USING phrase also contains parameters, but in this case the actual values of the parameters in the chaining program are substituted for those of the chained program. The runtime system copies the data values listed in the chaining program to a safe place in memory, loads the chained program into memory, and copies the data values into their corresponding parameters in the chained MS-COBOL program. These parameters are specified by a chaining phrase in the PROCEDURE DIVISION header of the chained MS-COBOL program.

Parameters may be passed to called or chained non-COBOL programs. Such programs must be designed to accept such data. See Sections 10.6, "Calling Non-COBOL Subroutines," and 10.7, "Chaining Non-COBOL Programs," for more information.

Note

MS-COBOL programs are limited to passing 60 parameters. See Section 6.3, "DATA DIVISION Limitations," of the *Microsoft COBOL Reference Manual* for more information.

10.1 PROCEDURE DIVISION Header With CALL and CHAIN

A called or chained MS-COBOL program may specify data-items to be passed from the invoking program. These data-items are specified in the USING and CHAINING phrases of the PROCEDURE DIVISION header.

The format of the PROCEDURE DIVISION header of an MS-COBOL program is

```
PROCEDURE DIVISION
    [ {USING | CHAINING} data-name-1
    [, data-name-2]...].
```

The data-names in the optional USING phrase must be defined in the LINKAGE SECTION of a called subprogram, in the same order specified in the USING phrase. The data-names in the optional CHAINING phrase must be defined, in any order, in the WORKING-STORAGE SECTION of the chained-to MS-COBOL main program.

10.2 Calling Microsoft COBOL Programs

You can transfer control to a subprogram in a COBOL run unit by using the CALL statement. Control is returned to the calling program when an EXIT PROGRAM statement is executed.

The format of the CALL statement is

```
CALL {literal-1 | identifier-1}
     [USING data-name-1 [, data-name-2]...]
     [; ON OVERFLOW imperative-statement]
```

Literal-1 is a non-numeric literal whose value is the PROGRAM-ID defined in the IDENTIFICATION DIVISION of a called MS-COBOL program. Identifier-1 must be defined as an alphanumeric data-item whose value can be a program-name.

Note

MS-COBOL loads all subroutines from disk when they are first called. To do this, it appends the .INT file extension to literal-1 or identifier-1, then searches for a file of that name, using the .INT file searching mechanism described in Section 4.1, "Finding .INT Files." To be consistent with the ANSI-74 Standard, the name of the source file for a called program should match its PROGRAM-ID, at least in the first six characters.

The data-names in the optional USING phrase refer to data whose addresses are passed to the called program. For example, a program needing inventory totals could call another program to calculate the totals and place them into designated data-names in the calling program.

If, during the execution of a CALL statement, it is determined that the available memory can't accommodate the specified program, and the ON OVERFLOW phrase is specified, the call is not made, and the imperative statement contained in the ON OVERFLOW phrase is executed. If the ON OVERFLOW phrase is not specified, and memory for the subprogram is unavailable, the program will terminate with a runtime error.

Warning

A chained or called program should have neither a chaining list nor items in the USING list unless the invoking CHAIN or CALL statement has a USING list. (The special case of chained programs that are invoked directly will be discussed later.) Furthermore, the number of entries in the lists should be equal, and entries with corresponding positions in the two lists should reference data-items of the same size and USAGE. Failure to conform to these rules will not be diagnosed and may cause unpredictable results at runtime.

Data-item values named in the PROCEDURE DIVISION header are established at program initialization time by the contents of corresponding data-items in the invoking CALL or CHAIN statement. In the case of CALL, the identification is made by passing data addresses. Therefore, if the value of a data-item named in a PROCEDURE DIVISION USING phrase is changed during subprogram execution, the corresponding data-item in the calling program will reflect the change after control is returned from the subprogram.

When the USING phrase is used, the following must be true:

1. Within the Calling Program

   The data-names listed in the USING phrase must be declared in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION of the DATA DIVISION.

2. Within the Called Program

   The data-names corresponding to those in the USING phrase of the calling program must be declared in the LINKAGE SECTION of the DATA DIVISION and in a USING phrase after the PROCEDURE DIVISION header. The names in the LINKAGE SECTION and in the PROCEDURE DIVISION header must be in the same order.

Control is returned to the calling program by an EXIT PROGRAM statement in the PROCEDURE DIVISION.

The programmer must make sure that the data-items listed in the calling program and in the called program are equivalent. See the *Microsoft COBOL Reference Manual* for more detailed information on data-items.

Example of a Calling Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      PROG1.
ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01   PARAMETER-1      PIC X(20)  VALUE SPACES.

PROCEDURE DIVISION.
100-MAIN.
     MOVE "HELLO" TO PARAMETER-1.
     CALL "PROG2" USING PARAMETER-1.
     DISPLAY "GOODBYE".
     STOP RUN.
```

Example of a Called Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      PROG2.
ENVIRONMENT DIVISION.

DATA DIVISION.
LINKAGE SECTION.
01   LOCAL-PARAM      PIC X(20).

PROCEDURE DIVISION USING LOCAL-PARAM.
100-MAIN.
     DISPLAY LOCAL-PARAM.
     EXIT PROGRAM.
```

After PROG1 and PROG2 are compiled, and PROG1 is executed, the following will be displayed on the screen:

```
HELLO
GOODBYE
```

10.3 Canceling the Called Program

The CANCEL statement releases previously called MS-COBOL object programs from memory. The format for the CANCEL statement is

```
CANCEL {identifier-1 | literal-1}
       [, {identifier-2 | literal-2}] ...
```

The identifiers or nonnumeric literals specify the subprograms to be released by the CANCEL statement. The identifiers must be defined as alphanumeric data so that the value can be a program name.

Subprograms are released from the memory occupied during the execution of a CALL operation. These subprograms return to their initial states after execution and are in that state when subsequently called.

A subprogram named in the CANCEL statement must not be in the process of being executed; an EXIT PROGRAM instruction must have been executed in the subprogram before a valid CANCEL operation can be performed.

10.4 Chaining MS-COBOL Programs

The CHAIN statement is used to permanently transfer control to a separately compiled program, which is loaded into memory and executed. The chained program can issue its own CHAIN statement or may even issue a CHAIN statement to its original chaining program, but it cannot otherwise return to the original program.

The format of the CHAIN statement is

```
CHAIN {literal-1 | identifier-1}
      [USING data-name-1 [, data-name-2]...]
```

Literal-1 or identifier-1 is the filename of an executable program or the name of an MS-COBOL main program. Literal-1

is a nonnumeric literal, and identifier-1 must be defined as an alphanumeric data-item whose value can be a valid MS-DOS filename.

Note

To find the file to chain to, MS-COBOL takes the name specified by literal-1 or identifier-1 and strips off any file extension. It then appends the .INT extension to the resulting name, and attempts to find and open the file using the .INT file searching mechanism described in Section 4.1, "Finding .INT Files." If the file can be opened, that MS-COBOL program is chained to. If the file cannot be opened, MS-COBOL attempts to open the file using the filename specified in literal-1 or identifier-1, and if the file can be opened, the program is chained to.

The data-names in the optional USING list are data-items defined in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION of the chaining program.

For more details about the CHAIN format, see the *Microsoft COBOL Reference Manual.*

If the USING phrase is included, the values of the data-items listed there will be copied to a safe place in memory, and when the chained program is loaded and run, they will be substituted for the equivalent values in the chained program. This allows the user to run a new program using values established in an earlier program. When this phrase is used, the following requirements must be met:

1. Within the Chaining Program

   The data-items listed in the USING phrase must be declared in the FILE SECTION, WORKING-STORAGE SECTION, or LINKAGE SECTION of the DATA DIVISION.

2. Within the Chained Program

   The data-items corresponding to those in the USING phrase of the chaining program must be declared in the WORKING-STORAGE SECTION of the DATA DIVISION and in a CHAINING phrase after the PROCEDURE DIVISION header.

An MS-COBOL program with a CHAINING phrase specified in its PROCEDURE DIVISION header may be executed directly, rather than being chained by another MS-COBOL program. If this is done, the values to be used by the executed program may be passed on the command line that invokes the program. This is one way in which runtime command line parameters may be passed to MS-COBOL programs; another way is through use of the built-in subroutine COMMAND, as described in Section 10.5, "Calling MS-COBOL Extension Subroutines."

If this scheme is used, the amount of data that may be passed is limited by MS-DOS to 128 bytes.

Example of a Chaining Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      PROG3.
ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01   PARAMETER-1      PIC X(20)  VALUE SPACES.
01   CHAIN-FILE       PIC X(20)  VALUE SPACES.

PROCEDURE DIVISION.
100-MAIN.
     DISPLAY "HELLO".
     MOVE "GOODBYE" TO PARAMETER-1.
     MOVE "PROG4" TO CHAIN-FILE.
     CHAIN CHAIN-FILE USING PARAMETER-1.
     STOP RUN.
```

Example of a Chained Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      PROG4.
ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01   LOCAL-PARAM      PIC X(20).

PROCEDURE DIVISION CHAINING LOCAL-PARAM.
100-MAIN.
     DISPLAY LOCAL-PARAM.
     STOP RUN.
```

After PROG3 and PROG4 are compiled, and PROG3 is executed, the following will be displayed on the screen:

```
HELLO
GOODBYE
```

PROG4 may also be executed directly, rather than being chained. If PROG4 were to be invoked with the command line

```
RUNCOB PROG4 EUREKA
```

it would display

```
EUREKA
```

10.5 Calling MS-COBOL Extension Subroutines

Several callable subroutines have been added to the MS-COBOL runtime system. They enhance file handling from within MS-COBOL programs, and act as extensions to normal COBOL statements. These routines are invoked with the COBOL CALL statement, with a USING list containing the required parameters.

The names and functions of the available routines are

EXIST	Checks to see whether a specified file exists
RENAME	Renames a specified file
REMOVE	Deletes a specified file
COMMAND	Gets the command line given to the runtime executor
UPCASE	Turns alphabetic characters within a data-item into upper case
LOCASE	Turns alphabetic characters within a data-item into lower case
EXCODE	Sets the MS-DOS exit-code to a specified value

The routines are called as follows:

```
CALL "EXIST" USING file-name, status.
CALL "RENAME" USING old-file-name, new-file-name,
     status.
CALL "REMOVE" USING file-name, status.
CALL "COMMAND" USING command-line.
CALL "UPCASE" USING data-name, length, status.
CALL "LOCASE" USING data-name, length, status.
CALL "EXCODE" USING exit-code.
```

Arguments in the USING list should be defined in the WORKING-STORAGE or LINKAGE sections as follows:

file-name	Alphanumeric data-names in the format of a valid MS-DOS file name (full MS-DOS path names may be used).

old-file-name	Same as file-name. This is the original name of a file to be renamed.
new-file-name	Same as file-name. This is the new name of a file to be renamed.
status	An alphanumeric two-character data-item (PIC XX). Returned status codes are described below.
command-line	An alphanumeric data-item with PIC X(128). The MS-DOS command line passed to the runtime executor will be loaded here, and command line will be padded on the right with spaces.
data-name	An alphanumeric data-item of length 9999 or smaller. Alphabetic data within data-name will have its case modified by UPCASE or LOCASE.
length	An unsigned numeric data-item with USAGE DISPLAY (no COMP type data-items may be used). PIC can be from PIC 9 through PIC 9(4). The largest allowable value for length is 9999.
exit-code	An unsigned numeric data-item with USAGE DISPLAY (no COMP type data-items may be used). PIC can be from PIC 9 through PIC 9(3). The largest allowable value for exit-code is 255.

Returned Status Values

In general, the status codes are similar to those used by COBOL for file-handling status codes.

Status code:	"00"
Description:	Routine successful; no errors encountered.
Notes:	For EXIST, "00" indicates file found.

Status code:	"30"
Description:	Routine unsuccessful; no action taken.
Notes:	For EXIST, "30" indicates file not found. For UPCASE and LOCASE, it usually indicates an invalid length field.

Status code:	"92"
Description:	Duplicate name found; no action taken.
Notes:	Used in RENAME only, when a file with new-file-name already exists.

Programming Notes

These file-handling routines do not have the full MS-COBOL error-handling abilities, so they should be used carefully. Status values should always be tested. If the number of arguments received by any of these functions is not the expected number, the routines return immediately, without taking any action or reporting any error. Since status would not be changed in this case, the status data-item should always be initialized to a value such as SPACES (which would never be returned by these subroutines) before making the call. For example, an attempt to make the invalid call

```
CALL "RENAME" USING INFILE, IN-STATUS.
```

would result in no action, and IN-STATUS would be unchanged, since one of the expected arguments is missing.

It is usually possible to RENAME or REMOVE an open file, and problems may not arise until later in processing. If possible, check to see that files are closed before using RENAME or REMOVE. These routines may be used on files not defined from within a COBOL program, so checking may not always be possible.

UPCASE and LOCASE check to see that the length specified is not greater than the size of data-name. If the length is larger, an error status "30" will be returned, and no action will be taken.

EXCODE can be used to set the MS-DOS exit code, which may be tested in batch files. See Chapter 5, "Batch Command Files," for more information.

The command line returned from COMMAND includes everything on the line after the name of the runtime executor. For example, if you enter

```
RUNCOB TEST1/P NEWVALUE,100
```

you will receive

```
TEST1/P NEWVALUE,100
```

Note that the COBOL UNSTRING command is an excellent tool for breaking a command line into its elements. In the above example, we could write

```
CALL "COMMAND" USING COMMAND-LINE.

UNSTRING COMMAND-LINE DELIMITED BY "," OR " "
    INTO
        PROGRAM-NAME,
            DELIMITER IN DEL1, COUNT IN  COUNT1,
        VALUE-ID,
            DELIMITER IN DEL2, COUNT IN  COUNT2,
        MAX-VALUE,
            DELIMITER IN DEL3, COUNT IN  COUNT3.
```

and the resulting fields would contain

PROGRAM-NAME	"TEST1/P"
VALUE-ID	"NEWVALUE"
MAX-VALUE	"100"
DEL1	" "
COUNT1	7
DEL2	","
COUNT2	8
DEL3	" "
COUNT3	3

10.6 Calling Non-COBOL Subroutines

A Microsoft COBOL program may call non-COBOL subroutines.

Note that several built-in subroutines may eliminate the need for writing some non-COBOL subroutines. They allow MS-COBOL programs to delete and rename files and check for a file's existence, among other functions. These subroutines are described in Section 10.5, "Calling MS-COBOL Extension Subroutines."

When a non-COBOL subroutine is called, the MS-COBOL runtime system searches through a user-supplied subroutine table for the desired routine. If it is found, the subroutine is called using the parameter passing conventions described in Section 10.6.1, "Passing Parameters on the Stack."

The following discussion will explain how subroutines written in Microsoft Macro Assembler language may be called from MS-COBOL programs. Subroutines written in other languages which match the calling and parameter passing conventions may also be called. Future versions of MS-COBOL will allow programs written in other Microsoft languages to be called as well, using the parameter passing conventions of those languages. When this feature becomes available, the details concerning calls to these languages will be included in the file UPDATE.DOC. Consult the Microsoft Macro Assembler manual for instructions on writing assembly language programs.

When an MS-COBOL program calls an assembly language subroutine, the runtime system transfers execution to the subroutine by means of a FAR CALL instruction. Execution should return via the FAR RET instruction.

10.6.1 Passing Parameters on the Stack

Parameters are passed by reference (i.e., by passing the address of the parameter). Parameter addresses are passed on the stack with the first parameter pushed to the stack first (see Figure 10.1).

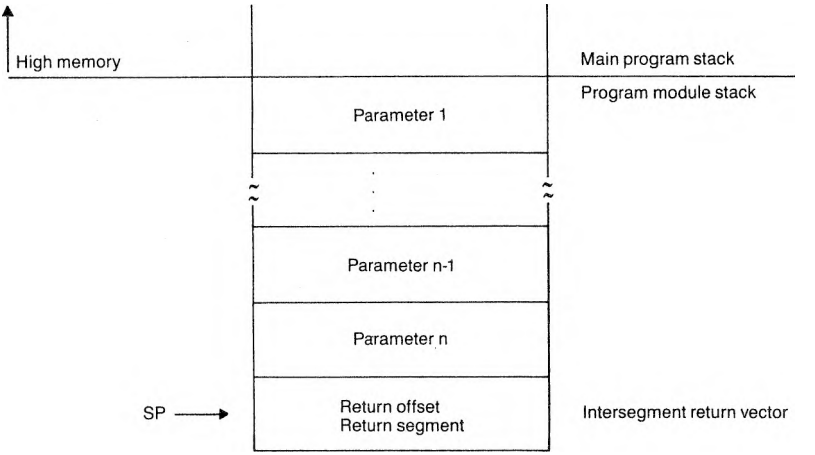


Figure 10.1 Contents of Stack at Entry to a Routine

Since MS-COBOL saves its registers and stack pointer before a call, and restores them upon return from the subroutine, the called routine may use registers freely. The called program does not need to remove the parameter addresses from the stack before returning, but may do so without causing damage.

The name of a Macro Assembler program module should be defined by a PUBLIC directive and be declared as PROC FAR. Data and code segments must be defined so they don't conflict with those used by the MS-COBOL modules. Macros START CSEG, END CSEG, START DSEG, and END DSEG may be used to establish correct segments. They are provided in the file USERSEG.MAC, which may be included in your Macro Assembler routines. The following examples show how these macros may be used.

Macro Assembler programs that ran with earlier versions of MS-COBOL may be used if the code and data segments are defined as previously described.

Names of user subroutines must not conflict with module names used by the MS-COBOL runtime executor. If a conflict occurs, an error will be noted by MS-LINK (the Microsoft linker for MS-DOS), and the user routine name should be changed.

Because the stack space used by an MS-COBOL program is contained within the program boundaries, assembler programs that use the stack must not overflow or underflow the stack. The best way to ensure safety is to save the MS-COBOL stack-pointer upon entering the routine and to set the stackpointer to another stack area. The assembler routine must then restore the saved MS-COBOL stackpointer or otherwise preserve the return address before returning to the main program.

Subroutines can expect only as many parameters as are passed: the calling program is responsible for passing the correct number. The user must determine that the type and length of arguments passed by the calling program are acceptable to the called subroutine; neither the compiler nor the runtime system checks for the correct number of parameters.

It is usually most convenient if numeric values to be passed are declared as binary (i.e., USAGE IS COMP-0 or COMP-4) in the WORKING-STORAGE SECTION of the calling program. Note, however, that the storage order of binary items passed from MS-COBOL programs may not follow the order expected by the hardware.

In the current MS-DOS implementation of MS-COBOL, data-items with USAGE COMP-0 are stored with the most significant and least significant bytes reversed; COMP-4 data-items have their most and least significant words reversed, with the bytes reversed within each word. This storage order may change in future implementations of MS-COBOL.

See Section 10.6.3, "Examples of Calls to Assembly Language Subroutines," for examples of how binary data-items may be used.

10.6.2 Creating and Linking Assembly Language Subroutines

The name of an assembly language subroutine to be called must be entered into a user-supplied table of subroutine names. The table and the subroutine(s) must then be linked with MS-COBOL object modules to produce a new runtime executor. The INSTALL Utility should be run on the new executor if interactive screen handling is desired. MS-COBOL programs that call the assembly language subroutine(s) should be run using the new runtime executor, rather than RUNCOB.EXE.

The following procedures should be followed when assembly language subroutines are used:

1. Make the required disk files available. The following files from the distribution disk must be available:

   COBOL1.LIB
   COBOL2.LIB
   DEBUG.LIB
   COBOL1.OBJ
   COBOL2.OBJ
   DEBUG.OBJ
   ASM.ASM
   USERPROG.MAC
   USERSEG.MAC
   PSEG.MAC
   INSTALL.COM
   INSTALL.MSG
   INSTALL.OVL
   INSTALL.SPC
   INSTALL.DAT
   MAKERUN.BAT
   MAKERUN2.BAT

   The following utilities are also required:

   ASM86.EXE
   LINK.EXE
   text editor

2. Edit USERPROG.MAC with EDLIN or another text editor. For each subroutine to be called, enter a line of the format

   asmnam *subroutine-entry-point-name, type*

   This will create the table of subroutine names, which will later be assembled with ASM.ASM, to be incorporated into the new runtime executor module. The second argument, type, tells the COBOL runtime the language in which the subroutine is written. Type may be:

ASM86
C
PASCAL
FORTRAN

In this example, ASM86, Microsoft Macro Assembler Language, would be used. Language support may not be implemented for some of these languages. If the language is not supported, a message will be printed during the assembly of ASM.ASM. When new language support is available, it will be documented in the file UPDATE.DOC.

3. Create and assemble your subroutines.

   Include file USERSEG.MAC or its equivalent in your program file to establish correct segmentation. If USEGSEG.MAC is included, begin each group of data with the START DSEG macro, and end each data group with END DSEG. Similarly, macros START CSEG and END CSEG should bracket all sections of code. These macros will create the needed PUBLIC declaration as well as set up the segments. Invoke each macro by entering

   macro-name subroutine-entry-point-name

   Example:

```
        START_DSEG DSPACE
savbc   dw    ?   ;bytes per disk cluster
savdx   dw    ?   ;temporary register area
        END_DSEG DSPACE
```

4. Use batch file MAKERUN.BAT to incorporate your subroutine name table and subroutines into a new MS-COBOL runtime executor. MAKERUN.BAT will

   a. Assemble ASM.ASM, incorporating your modified USERPROG.MAC, which contains the subroutine name table.

   b. Link the subroutine name table and any subroutines with the MS-COBOL runtime modules contained in COBOL1.LIB and COBOL2.LIB to produce the new runtime executor.

   MAKERUN.BAT allows up to eight Macro Assembler subroutines to be linked into the new runtime executor.

Programmers experienced with MS-DOS batch files may modify MAKERUN.BAT if more subroutine object files are needed. MAKERUN2.BAT may be used instead of MAKERUN.BAT on smaller floppy-disk based systems.

MAKERUN.BAT is invoked as follows:

```
MAKERUN runtime-executor-name object-file-1
        object-file-2 . . . object-file-8
```

Object files 1 through 8 are the names of the assembly language subroutines. If fewer than eight routines are needed, the latter names may be omitted from the command line.

If the MS-COBOL Interactive Debug Facility is to be used, the *first* object file specified in the command line should be DEBUG.

Note that most of the files specified in step 1 will be required for the execution of MAKERUN. In this example

```
MAKERUN RUNSUB SUBIT
```

the assembly language routine SUBIT.OBJ will be included in the runtime executor RUNSUB.EXE.

In this example

```
MAKERUN DEBTEST DEBUG DSPACE SUBIT
```

the assembly language routines SUBIT.OBJ and DSPACE.OBJ will be included in the runtime executor DEBTEST.EXE, which will invoke the MS-COBOL Interactive Debug Facility, since DEBUG.OBJ is also included.

5. Run the INSTALL utility if interactive screen handling is desired (see Chapter 9, "INSTALL Program").

10.6.3 Examples of Calls to Assembly Language Subroutines

The following are examples of COBOL programs containing calls to assembly language subroutines. It is assumed that the programmer is familiar with Macro Assembler programming and with MS-DOS function calls. The *MS-DOS User's Guide* and Macro Assembler manual should be used for reference, if needed.

Example 1:

This COBOL program calls the assembly language routine SUBIT, which subtracts two data-items and puts the result in a third data-item. The data-items are declared as binary with USAGE COMP-0. Note that the bytes of the binary data-items are reversed before and after use, since MS-COBOL and 8088/8086 storage order is different.

1. CALL1, MS-COBOL calling program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      CALL1.
ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.
01   WORK-AREAS.
     05 PARM1     PIC S9(5)  COMP-0 VALUE 511.
     05 PARM2     PIC S9(5)  COMP-0 VALUE 384.
     05 PARM3     PIC S9(5)  COMP-0 VALUE ZERO.
     05 PPARM-1   PIC -----9  VALUE ZERO.
     05 PPARM-2   PIC -----9  VALUE ZERO.
     05 PPARM-3   PIC -----9  VALUE ZERO.

PROCEDURE DIVISION.
MAIN.
     CALL "SUBIT" USING PARM1, PARM2, PARM3.
     MOVE PARM1 TO PPARM-1.
     MOVE PARM2 TO PPARM-2.
     MOVE PARM3 TO PPARM-3.
     DISPLAY PPARM-1 " - " PPARM-2 "
          = " PPARM-3.
     STOP RUN.
```

2. Table entry in USERPROG.MAC:

```
asmnam SUBIT,ASM86
```

3. Macro Assembler routine SUBIT.ASM:

```
TITLE   Assembly language test routine SUBIT

;***********************************************
; Contains a routine to subtract 2 numbers and
; return the result in a third. Called from an
; MS-COBOL program.
;***********************************************

        include userseg.mac
        PAGE

;
; Structure for accessing parameters
; on the stack

DYNS    STRUC
        DW      ?       ;Pushed BP
        DD      ?       ;Long return address.
PARM3   DW      ?       ;Parameter number 3.
PARM2   DW      ?       ;Parameter number 2.
PARM1   DW      ?       ;Parameter number 1.
DYNS    ENDS

START_CSEG SUBIT

push    bp
mov     bp,sp
mov     bx,[bp].parm1     ; get addr of parm1
mov     ax,[bx]
xchg    ah,al   ; COBOL COMP-0 bytes reversed
mov     bx,[bp].parm2     ; get addr of parm2
mov     cx,[bx]
xchg    ch,cl   ; COBOL COMP-0 bytes reversed
sub     ax,cx
xchg    ah,al ; restore byte order of result
mov     di,[bp].parm3     ; get addr of parm3
mov     [di],ax
pop     bp
ret             ; note COBOL will pop arguments
                ; off stack, but it can also
                ; be done here using ret 6
                ; with no bad results
END_CSEG SUBIT
end
```

Example 2:

This COBOL program calls the assembly language routine DSPACE, which makes a call to the MS-DOS Get Disk Free Space function described in the *Microsoft MS-DOS Programmer's Reference Manual.* This program uses both COMP-0 and COMP-4 binary storage, and attention should be paid to the byte and word reversal done in DSPACE. If the returned value of LPARM3 is ZERO, then an error is indicated.

1. CALL2, MS-COBOL calling program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      CALL2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WORK-AREAS.
    05 PARM1      PIC S9(5)  COMP-0  VALUE ZERO.
    05 LPARM2     PIC S9(9)  COMP-4  VALUE ZERO.
    05 LPARM3     PIC S9(9)  COMP-4  VALUE ZERO.
    05 PPARM-1    PIC -----9  VALUE ZERO.
    05 LPPARM-2   PIC --,---,---,--9  VALUE ZERO.
    05 LPPARM-3   PIC --,---,---,--9  VALUE ZERO.

PROCEDURE DIVISION.
MAIN.
    DISPLAY " Enter drive (0 = default, ",
            " 1 = a:,etc.)".
    ACCEPT PARM1.
    CALL "DSPACE" USING PARM1, LPARM2, LPARM3.
    MOVE PARM1 TO PPARM-1.
    MOVE LPARM2 TO LPPARM-2.
    MOVE LPARM3 TO LPPARM-3.
    IF LPARM3 NOT = ZERO
        DISPLAY "Drive ", PPARM-1
                "Free Space = ", LPPARM-2
                "Total Space = ", LPPARM-3 "
    ELSE
        DISPLAY "DSPACE: Invalid drive "
            " designation".
    STOP RUN.
```

2. Table entry in USERPROG.MAC:

```
asmnam DSPACE,ASM86
```

3. Macro Assembler routine DSPACE.ASM:

```
TITLE   Assembly language test routine DSPACE

;***********************************************
; Contains a routine to call the MS-DOS Get
; Free Disk Space function.  Zero is returned
; in the third parameter in case of error.
;***********************************************

        include userseg.mac

;
;Structure for accessing parameters on stack
;
DYNS2   STRUC
        DW      ?       ;Pushed BP.
        DD      ?       ;Long return address.
ARG3    DW      ?       ;Parameter number 3.
ARG2    DW      ?       ;Parameter number 2.
ARG1    DW      ?       ;Parameter number 1.
DYNS2   ENDS

MSDOS   EQU     21H     ;standard MS-DOS
                        ;entry point
SPACE   EQU     36H     ;get disk free space
                        ;function

        START_DSEG DSPACE
savbc   dw      ?       ;bytes per disk cluster
savdx   dw      ?       ;temporary register area
        END_DSEG DSPACE


START_CSEG DSPACE

push    bp              ;use bp to point to
                        ;arguments
mov     bp,sp
mov     bx,[bp].arg1    ;get addr of arg1
                        ;(drive number)
mov     dx,[bx]         ;get drive into dl
xchg    dh,dl           ;COBOL COMP-0 bytes
                        ;reversed
mov     ah,SPACE        ;call MS-DOS function
INT     MSDOS           ;on return:

          ; bx = # of free clusters on drive
          ; dx = total # of clusters on drive
          ; cx = bytes per sector
          ; ax = sectors per cluster
```

```
            ;       or FFFF if error

cmp     ax, 0FFFFH      ; error?
jz      error
mov     savdx,dx        ; multiply wipes out dx,
                        ; so save it
mul     cx              ; bytes/cluster in ax
jc      error           ; should never overflow,
                        ; but ...
mov     savbc,ax        ; save for later use
mul     bx              ; low order result in ax
                        ; high order in dx
mov     bx,[bp].arg2
           ; get addr of COMP-4 free space
           ; store 4 byte value in this order:
           ; least significant word (in dx),
           ; then most significant word (in ax)
           ; with bytes of each word reversed
xchg    dh,dl           ; COBOL COMP-0 bytes
                        ; reversed
mov     [bx],dx
xchg    ah,al           ; COBOL COMP-0 bytes
                        ; reversed
mov     [bx+2],ax


mov     ax,savbc        ;restore ax and dx
mov     dx,savdx
mul     dx              ;low order result in ax,
                        ;high in dx
mov     bx,[bp].arg3    ;get addr of COMP-4 total
                        ;space store 4 byte
                        ;variable
xchg    dh,dl           ;COBOL COMP-0 bytes
                        ;reversed
mov     [bx],dx
xchg    ah,al           ;COBOL COMP-0 bytes
                        ;reversed
mov     [bx+2],ax

jmp     spret           ; done

error:
        xor     ax,ax           ; zero arg3 to
                                ; indicate error
        mov     bx,[bp].arg3
        mov     [bx],ax
        mov     [bx+2],ax
spret:
```

10.7 Chaining Non-COBOL Programs

Executable non-COBOL programs are chained in the same way as MS-COBOL programs (see Section 10.4, "Chaining MS-COBOL Programs"). The following additional information will be useful when you are writing non-COBOL programs that will be chained.

When the USING phrase is included in the CHAIN statement, the parameters passed between programs are stored in the command line area used by MS-DOS. See the *MS-DOS User's Guide* for information on accessing data stored in this area. Each parameter specified in the USING list of the chaining MS-COBOL program is copied into the command line area, and separated from the previous parameter by a single space. The amount of data that may be passed in this way is limited by MS-DOS to 128 bytes.

The chained program expects the same number and format of parameters as were passed. No checking will be done by the compiler or the runtime system.

Chapter 11
REBUILD Utility Version 2.0

The Rebuild Utility, (REBUILD) version 2.0 and later, is primarily a tool for rebuilding Indexed key files created by MS-COBOL, the Microsoft Multi-Key ISAM Facility, and Microsoft SORT. Once a damaged key file has been recovered, the associated data files may again be accessed by an MS-COBOL application. The Rebuild Utility can also modify a functional key file. See Section 11.5, "Adding and Deleting Indexes," for details.

Each Indexed file declared in an MS-COBOL program will generate two disk files: a key file and a data file.

The key file contains keys, pointers to keys, and pointers to data. The VALUE OF FILE-ID clause specifies an operating system file-name for a file containing data only. That file-name is given an extension (.KEY) to form the file specification of the key file.

The data file contains data records and usually a data dictionary which contains a description of the data record format (record description). The data dictionary resides at the beginning of the data file. See Section 11.7, "Data Dictionary," for details.

A record description, which describes the key fields within the record, is used to build a new key file. (The record description may also contain all fields in the record instead of only the key fields.)

REBUILD allows you to

1. Generate new key files from Indexed data files.

   This is probably the most common use of REBUILD for MS-COBOL users.

   REBUILD must be run to regenerate a corrupted key file. The data file is the input to REBUILD for this task. See Section 11.3.1 "Fixing a Corrupted Key File," for more information on this task.

2. Compress the data file.

   Free space is created during normal record processing using DELETE and REWRITE statements, especially when variable length records are used. Data file compression removes this free space from the data file.

3. View the data dictionary.

   Whenever MS-COBOL or REBUILD creates an Indexed data file, it also builds a data dictionary — a binary record description located at the beginning of the data file. This useful feature allows the Indexed file structure to be examined and, if desired, the data dictionary to be saved as an ASCII file by redirecting the output as shown in Section 11.6, "Creating and Using a dd (ASCII) Text File."

   REBUILD can use a data dictionary to rebuild key files.

Note

REBUILD also supports an interactive mode for entering new data dictionary information. You switch to this mode by starting REBUILD with the /I switch as explained in Section 11.5.2, "Updating a Key File: Interactive Mode."

11.1 Invoking REBUILD

Use the following syntax to provide arguments on the REBUILD command line:

```
REBUILD [command-line]
```

where [*command-line*] stands for

```
source-file , [ target-file ] , [ key  description ] , [ dd-file ] [ ; ] [ /switches ]
```

Blanks are not allowed in the command line between the arguments. A semicolon can be used to generate default values for the target-file, key description, or dd-file.

Note

REBUILD will prompt you for information that it needs that was not provided on the command line.

11.2 Definitions of the Command Line Arguments

REBUILD 2.0 supports the following command line arguments:

1. *source-file*

   The source-file is the name of the data file from which you want to generate a key file. Source-file is the only required element of the command line.

   An Indexed data file compatible with the Rebuild Utility can come from the following sources:

   a. as output from an MS-COBOL application which creates Indexed files using MS-COBOL version 2.0 or later

   b. as data files created by the Microsoft SORT Facility which can convert a file with RELATIVE organization to INDEXED organization

   c. as output from a non-MS-COBOL application which creates Indexed files using the Microsoft Multi-Key ISAM Facility

---

Note

If the Indexed file that you are supplying to REBUILD was created by MS-COBOL (version 2.0 or later), or a Microsoft ISAM application, or by REBUILD (which is a Microsoft ISAM application), the data dictionary for that data file already exists and is passed to REBUILD. In fact, this is the same record description that REBUILD will insert into the data file whenever an output data file is specified on the command line.

---

2. *target-file* (optional)

   If target-file is present, the data file will be copied and a key file (*target-file*.KEY) will be generated. All free space will be removed from the file. A target-file must be specified if a data dictionary is to be added to a data file.

3. *key description* (optional)

   Key description describes a single key of the format *integer-1:integer-2*. Integer-1 and integer-2 refer to the key field location and key field length.

   The key description can be used only when the Indexed file will contain a single key. The description contains the starting position and size of the key. The syntax matches the syntax used on the REBUILD 1.1 command line.

4. *dd-file* (optional)

   The dd-file is the ASCII file prepared with a text editor. dd-file contains an ASCII description of the data dictionary for a corresponding Indexed data file. The presence of dd-file tells REBUILD that a file in ASCII format must be processed. A description of the dd-file is given in Section 11.6, "Creating and Using a dd (ASCII) Text File."

The data dictionary is a binary record description located at the beginning of the data file. REBUILD generates this binary record automatically during a data file copy. It will also generate this binary data dictionary from the ASCII record description contained in dd-file if you give both the dd-file and target-file arguments.

If no target-file is specified, REBUILD will use the data dictionary in source-file, rather than the dd-file, to create the new key file.

See the FIELD statement syntax in Section 11.6.1, "Syntax Considerations," for details about defining the record fields with REBUILD 2.0.

5. *switches* (optional)

   a. /I(nteractive) switch

      The /I switch turns on REBUILD's interactive mode (see Section 11.5.2, "Updating a Key File: Interactive Mode," for details). All other arguments, except source-file and target-file, are ignored.

   b. /T(erse) switch

      The /T switch sets REBUILD in the terse mode. If there were no fatal errors, there will be no output to the screen.

   c. /P(rint) switch

      The /P switch brings an ASCII version of the data dictionary to the screen. The output can be redirected to a file or device. (See Section 11.6, "Creating and Using a dd (ASCII) Text File," for information on redirection.)

   d. /F(orce) switch

      The /F switch allows REBUILD to exit or perform other routines without waiting for your confirmation.

   e. /S(ingle key) switch

      The /S switch will copy the data file, remove the data dictionary and free space; no key file is built. /S generates a data file that is compatible with the single-key Rebuild Utility, version 1.1.

See the FIELD statement syntax in Section 11.6.1, "Syntax Considerations," for the details about defining the record fields with REBUILD 2.0.

11.3 Using REBUILD as a Tool

The following sections describe REBUILD's major uses as a tool.

See Section 11.1, "Invoking REBUILD," for a description of REBUILD's command line arguments.

11.3.1 Fixing a Corrupted Key File

Assume an Indexed file named EMPREC.DAT was open for writing when a power failure occurred. Since it was not closed by MS-COBOL, it was 'corrupted' and must have its key file rebuilt. Assume that the data file contains a data dictionary (this will be true in most cases). The command line to REBUILD would be

```
REBUILD EMPREC.DAT;
```

REBUILD will read the data file description from the data dictionary in data file EMPREC.DAT, and use that to build a key file. The key file will be called EMPREC.KEY.

If the data file EMPREC.DAT had not contained a data dictionary as assumed above, a description of the data would have to be provided by entering a key description on the REBUILD command line, by providing a dd file (see Section 11.6, "Creating and Using a dd (ASCII) Text File"), or by using REBUILD's interactive mode (see Section 11.5.2 "Updating a Key File: Interactive Mode").

If there is only a single key in the record, the command line itself can contain this data description. Use the command

```
REBUILD EMPREC.DAT,,1:10;
```

This will also build the key file called EMPREC.KEY. It will contain a single key that is a fixed-length string, ten characters in length, which starts on the first character of the record. Since no target-file was specified to initiate the data file copy, your data file still won't contain a data dictionary and never will until a data file copy using a target-file specification is initiated.

11.3.2 Compressing the Data File

When a target file is specified on the command line, a copy of the source data file is produced in addition to a new key file. This data file is compressed (all free space records are removed). Free space records are "empty" records resulting from deletions and rewrites, which have yet to be reused by the Indexed file system.

Whenever a data file copy is performed without the /S switch, REBUILD will put a data dictionary into the data file. If one did not exist before, it will after REBUILD is run. For example, using the indexed file EMPREC.DAT on the REBUILD command line

```
REBUILD EMPREC.DAT,EMPREC2.DAT;
```

will create the new file EMPREC2.DAT that will contain the data records and the data dictionary from EMPREC.DAT. It will contain no free space records. The key file will be called EMPREC2.KEY.

11.3.3 Converting Microsoft COBOL Indexed Files

The key file formats for Indexed files created by MS-COBOL or REBUILD versions prior to 2.0 (1.0 format files) and Indexed files created by MS-COBOL or REBUILD 2.0 or later, or the Microsoft ISAM Facility (2.0 format files) are not identical and are not compatible. The conversion process between these two formats follows:

1. 1.0 Format to 2.0 Format

   This is straightforward since REBUILD is able to support both formats. If the 1.0 format data file is called FILE1.DAT, a valid REBUILD command line could be

```
REBUILD FILE1.DAT,FILE2.DAT,1:10;
```

Note

Since Microsoft 1.0 format Indexed files do not produce a data dictionary, you must provide a key field description on the command line; in this case "1:10" is specified. REBUILD will build the new Indexed data file with an appropriate data dictionary. This new data file (FILE2.DAT) is now fully compatible with MS-COBOL version 2.0 and later.

---

2. 1.0 Format to Multi-Key 2.0 Format

   If you plan to simultaneously upgrade this single-keyed data file to include alternate record keys, you'll need to revise your source program and create a new key file. Use one of the following methods to make this conversion:

   New Data Dictionary Method

   a. Using MS-COBOL 2.0, create a data dictionary that reflects your new indexing structure (i.e., write, compile and execute a COBOL program that creates an empty data file by opening the file for output, then closing it). Use the FILE SECTION from your application, and a FILE-CONTROL entry with ALTERNATE RECORD KEY specifications.

   b. Extract a copy of the data dictionary from the created data (empty) file by invoking REBUILD 2.0 with the /P switch and redirecting the data dictionary to a file. Delete the empty data file.

```
REBUILD EMPTY.DAT;>MULTI.DD/P
```

c. Update the single-key file of SINGLE.DAT to a multi-key file and create your targeted multi-key data file by invoking REBUILD again with a target-file for the data file copy and the data dictionary filename.

```
REBUILD SINGLE.DAT,MULTI.DAT,,MULTI.DD
```

d. Make your application compatible with the new multi-key data file by adding the ALTERNATE RECORD KEY clause to the FILE-CONTROL entry and compiling using MS-COBOL 2.0.

REBUILD Key Manipulation Method

Use the key manipulating functions of REBUILD as described in Sections 11.5.1, "Updating a Key File: ASCII Input File" and 11.5.2, "Updating a Key File: Interactive Mode" to update the keyfile and data dictionary by adding the described new key fields.

COBOL File Rewriting Method

After converting the 1.0 format file to a single-key 2.0 format file as described in Method One, write a COBOL program to read the converted single key file and copy it by writing to an Indexed file defined with the desired alternate keys.

3. 2.0 Format to 1.0 Format

The data dictionary must be removed since this is an enhancement for 2.0 format Indexed files (MS-COBOL version 2.0 or later).

To effect the change, REBUILD version 2.0 and REBUILD version 1.10 must be used. We'll use the same file and single-key description used in Section 11.3.2, "Compressing the Data File."

```
(1) REBUILD EMPREC.DAT,EMPREC2.DAT;/S
(2) REBUILD EMPREC2.DAT,EMPREC1.DAT,1:10;
```

Line (1) shows the REBUILD 2.0 command line and line (2) shows the REBUILD 1.0 command line.

See version 1.0 of the *Microsoft COBOL Compiler User's Guide* for a full description of REBUILD versions prior to 2.0.

11.4 Data Loss After a System Crash

A data file can be damaged when electrical power to the computer system is interrupted or the operating system is rebooted while an Indexed file is open in I-O or OUTPUT mode.

A system failure may leave the data file with partially written data records because of the high degree of disk file buffering in memory. This may cause REBUILD to fail to completely recover an Indexed file for either or both of the following reasons:

1. If the system failure occurred during a file update job, when 512 bytes of the file are kept in memory, the file may contain records with both original and new information. REBUILD cannot determine which part of the data was written during the terminated job, and therefore cannot exclude the new, incomplete data from the rebuilt file. Adding a current date field to data records may help discriminate between original and new data.
2. If the system failure occurred while records were being added to the Indexed file, the last 512 bytes of data will not be written to disk. REBUILD will detect that information is missing from the end of the file but cannot add it to the recovered file.

A data file can also be damaged when space is exhausted during a WRITE operation to the disk on which the Indexed file resides.

You will know that space was exhausted during the WRITE operation when WRITE produces a boundary error (file status "24"), indicating that the disk is full. When this happens, you should perform a CLOSE in order to write as much information as possible to disk.

It is likely, however, that the CLOSE will also return with a boundary error. As in the case of a system failure during the addition of records, the last 512 bytes of information will not be present within the data file and will, therefore, be unrecoverable by REBUILD.

11.5 Adding and Deleting Indexes

When indexes need to be added, deleted, or otherwise modified, you can use two methods to update the key file: use an ASCII input file, or use REBUILD's interactive mode.

11.5.1 Updating a Key File: ASCII Input File

Use a command line of the form

```
REBUILD SAMPLE.DAT,SAMPLE2.DAT,,NEW.DD
```

This directs REBUILD to create SAMPLE2.DAT and SAMPLE2.KEY, using NEW.DD as the data dictionary (record description) for SAMPLE2.DAT. REBUILD will also use NEW.DD file to update the index information in the key file when SAMPLE2.KEY is generated.

Note

REBUILD will ignore the input file NEW.DD if you have not specified SAMPLE2.DAT. Without an indication that you want a data file copy, REBUILD will assume that the existing data dictionary in SAMPLE.DAT is acceptable.

If you use a dd-file that changes the record description for an Indexed file, you will need to add the same index and field information contained in the dd-file to the programs that access the data file.

For example, suppose that the dd-file, NEW.DD, contains the following record description, and is designed to update the key file for a multi-key data file called BUYER.DAT. NEW.DD contains this information (*n*:*n* represents beginning byte:length):

```
FIELD PROCESS-CODE IS 63:1 ALPHA
    KEYED BY 1;
FIELD NUMBER IS 1:5 ALPHA
    KEYED BY 2;
FIELD NAME IS 6:20  ALPHA
    KEYED BY 3 DUPLICATES ALLOWED;
FIELD CITY IS 26:20  ALPHA;
FIELD ZIP-CODE IS 46:5 ALPHA;
SPLIT KEYSET 5 IS PROCESS-CODE ZIP-CODE
    DUPLICATES ALLOWED;
```

All MS-COBOL programs that used the BUYER.DAT key file with its previous index structure must be changed to reflect the new index structure. In the following source fragment from an application, REPORT.INT, a split key containing the two fields italicized is to be added to the existing FD entry:

```
FD  TRANSACTION-FILE
    RECORD CONTAINS 63 CHARACTERS
    LABEL RECORDS ARE STANDARD
    VALUE OF FILE-ID IS "\DATA\BUYER.DAT"
    DATA RECORD IS BUYER-DATA.

01  BUYER-DATA.
    05 NUMBER        PIC X(5).
    05 NAME          PIC X(20).
    05 CITY          PIC X(20).
    05 ZIP-CODE      PIC X(5).
    05 DESCRIPTION   PIC X(12).
    05 PROCESS-CODE  PIC X.
```

The italicized source lines in the following fragment indicate the corresponding fields which will make up the new split key:

```
SELECT TRANSACTION-FILE ASSIGN TO DISK

    LOCKING IS EXCLUSIVE
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS PROCESS-CODE
    ALTERNATE RECORD KEY IS NUMBER
    ALTERNATE RECORD KEY IS NAME  WITH
                              DUPLICATES
    ALTERNATE RECORD KEY IS SHIP-LOCATION =
      PROCESS-CODE, ZIP-CODE  WITH DUPLICATES.
```

When you invoke REBUILD with the command line

```
REBUILD BUYER.DAT,NEWBUYER.DAT,,NEW.DD
```

the data dictionary and the key file for NEWBUYER.DAT are updated, making NEWBUYER.DAT compatible with new versions of REPORT.INT.

Note

You may want to format the file description entry and the FILE-CONTROL entry codes on disk and embed them in the appropriate programs with the COPY statement. See Chapter 16, "COPY Statement," in the *Microsoft COBOL Reference Manual*, for more information.

11.5.2 Updating a Key File: Interactive Mode

REBUILD also has an interactive mode that supports your modification of the record description in an Indexed file's data dictionary.

To invoke REBUILD in this mode, enter a command line of the form

```
REBUILD SAMPLE.DAT,SAMPLE2.DAT;/I
```

These arguments place the table of commands on your terminal screen (see Figure 11.1, "REBUILD Interactive Mode Main Menu"), and place REBUILD into the interactive mode. The record description can then be examined or modified.

Target-file SAMPLE2.DAT is specified in the command line so that changes made to the record description will become part of the data dictionary once you complete the edit, and exit REBUILD.

See Section 11.6.2, "Statement Directory," for descriptions of the FIELD and RECORD statements.

List <field>:	List a field, or with no args, all fields.
Delete <field>:	Delete a field and/or split key.
Replace <field>:	Replace a field or split key.
Add <field>:	Add a new field or another field to a split key.
File options:	Change the file options.
Help:	List this display.
Exit:	Cancel REBUILD.
Quit:	Quit editing and perform REBUILD operations.

Figure 11.1 REBUILD Interactive Mode Main Menu

Menu Descriptions

The following commands may be typed in full or represented by their first letter. You will receive a prompt if additional information is required. If <field> is more than one word, enclose the name in quotation marks ("). Most of the commands take one or more arguments. You may enter all arguments at once or as prompted by REBUILD. Multiple arguments are separated by spaces.

List <field>:

Displays the current record description in ASCII file format. All field descriptions are printed if <field> is omitted.

Delete <field>:

Deletes a field description.

Add <field><statement>:

Adds a field description to the data dictionary. If <statement> is omitted, the Add statement issues the prompt

```
Specify a complete statement for Add
```

A FIELD/SPLIT statement, in the ASCII format described in Section 11.6.1, "Syntax Considerations," should then be entered.

File <options>:

Modifies the file options that have a global effect. If <options> is omitted, the prompt returned is

```
N)ame S)egmented M)inimum Allocation
```

Each option corresponds to one of the the options on the RECORD statement from the ASCII format. Enter the first letter for the desired option. The RECORD statement is described in Section 11.6.2, "Statement Directory." The Name and Minimum Allocation options will prompt for further input.

Replace <field><subcommand>:

Replaces all or part of a field description.

If the <subcommand> option is omitted, REBUILD issues the following prompts:

```
N)ame L)oc T)ype K)ey# DE)scending

D)uplicates I)nsensitive R)emove-Split

A)dd-Split DES)cending-Split
```

To select subcommands, enter the capitalized letter(s) to the left of the ")". The subcommands N)ame through I)nsensitive deal with fields and keys. The subcommands R)emove-Split through DES)cending-Split deal with split keys.

a. N)ame ["]*field name*["]

   Replaces the field name description. The quotation marks are needed only when the name has more then one word separated by spaces.

b. L)oc [*segment#*]*start position*[:*length*]

   Changes the field position. Segment # is not used in COBOL.

c. T)ype *field type*

Changes the data type. MS-COBOL only supports a type of ALPHA.

d. K)ey# *number*

Assigns a key number to the field. MS-COBOL assigns the prime key #1, the first alternate key #2, etc. . . You must select the same key number that MS-COBOL is going to use. Specifying a key number of 0 makes the field non-keyed.

e. DE)scending

This switches the internal "Descending" flag on or off each time it is excuted. MS-COBOL will read an Indexed file with this feature in reverse order. However, MS-COBOL can neither create a DESCENDING key file, nor inform a program that a file has this feature. DESCENDING key files should be used only with programs that expect this feature.

f. D)uplicates

This switches "duplicates allowed" on or off. MS-COBOL requires that key #1 not have duplicates allowed. This subcommand also works with split keys.

g. I)nsensitive

This switches the insensitive switch on or off. MS-COBOL will take advantage of the INSENSITIVE feature. However, it can neither create an INSENSITIVE FILE, nor inform a program that a file has this feature. INSENSITIVE files should be used only in programs that expect this feature.

h. R)emove-Split *field*

Allows you to delete the components which make up a split key.

i. A)dd-Split *field*

Allows you to add the components which make up a split key.

j. DES)cending-Split *field*

Allows individual components of a split key to have the descending attribute independent of the descending attributes in the referenced field. This switches "Descending" on or off each time it is excuted. MS-COBOL will read an Indexed file with this feature in reverse order. However, MS-COBOL can neither create a DESCENDING key file, nor inform a program that a file has this feature. DESCENDING key files should be used only with programs that expect this feature.

11.6 Creating and Using a dd (ASCII) Text File

To bring an ASCII version of SAMPLE.DAT's data dictionary to the screen, use a /P on the command line:

```
REBUILD SAMPLE.DAT;/P
```

If you want to redirect this output to another file (e.g., TEXT.DD), and then edit it to new specifications, enter the command line

```
REBUILD SAMPLE.DAT;/P>TEXT.DD
```

You could then edit TEXT.DD, and add the new version of the data dictionary to SAMPLE2.DAT, with the command line

```
REBUILD SAMPLE.DAT,SAMPLE2.DAT,,TEXT.DD
```

The ASCII files acceptable to REBUILD lend themselves to easy maintenance. The following sections describe the syntax and language recognized by REBUILD, and provide practical examples of record descriptions in data dictionary format.

11.6.1 Syntax Considerations

An Indexed file record may be described using the RECORD, FIELD, and SPLIT KEYSET statements. The RECORD statement is optional (only one RECORD statement may be given for any file). One or more FIELD statements must be used; SPLIT KEYSET statement(s) may be used. All statements must end with a semicolon.

Note that some entries in the diagrams will never be used with MS-COBOL Indexed files.

In the syntax diagrams that follow, optional material is indicated by square brackets ([ ]). Braces ({ }) are used with the vertical bar (|) to indicate a choice of two options, or with ellipses (...) to indicate a repeated group. The quoted square brackets (["["]) in the FIELD statement indicate that an actual bracket character may be entered. If the optional opening bracket ([) is chosen, the closing bracket (]) must also be used.

```
[RECORD  [record-name] [SEGMENTED]
     [MINIMUM ALLOCATION IS integer]] ;

FIELD [field-name] IS ["[" ] position [:size]
     datatype [INSENSITIVE] ["]"]

    [KEYED [BY] key-number [DUPLICATES ALLOWED]
    [DESCENDING]] ;

[SPLIT KEYSET key-number IS
    {{field-name | key-number} [DESCENDING]} ...
        [DUPLICATES ALLOWED]] ;
```

11.6.2 Statement Directory

REBUILD 2.0 supports the following keywords to create an ASCII file that describes data file records:

RECORD Statement

The optional RECORD statement sets global file options. These options can also be set with the F(ile) options command in the interactive mode. The arguments are

record-name	an optional identifying name.
SEGMENTED	the file will have a special segmented structure. This option is not used with MS-COBOL.
MINIMUM ALLOCATION	the minimum record size allocated from the data file. MINIMUM ALLOCATION can be used to minimize data file fragmentation when REWRITEing variable length records.

Data file fragmentation may occur when multiple level 01 entries of different sizes are given under an FD entry, or when the OCCURS DEPENDING ON clause is used in a record description. Choose the MINIMUM ALLOCATION size equal to the largest record size in the file to minimize fragmentation.

FIELD Statement

The FIELD statement describes each field in the record. We recommend that all fields in a record be defined even though REBUILD requires that only key fields be defined. This additional effort will allow future utilities to access desired data items. The interactive mode commands A(dd), L(ist), P(rint), R(eplace), and D(elete) use field names or entire FIELD statements as input.

The arguments of the FIELD statement are

field-name	Name of field; maximum 40 characters. If more than one word, enclose the name in double quotation marks (").
position	Position from start of the record (1 being the first position).

size	Size of data field in characters.
datatype	Data type is always ALPHANUMERIC for MS-COBOL files. ALPHA may be used as a synonym for ALPHA-NUMERIC.
INSENSITIVE	Characters which differ by case only are considered equal. MS-COBOL will take advantage of the INSENSITIVE feature. However, it can neither create an INSENSITIVE file, nor inform a program that a file has this feature. INSENSITIVE files should be used only in programs that expect this feature.

KEYED BY Clause

The optional KEYED BY clause makes a specific field value into a key to the record. During revisions of the key file using REBUILD, the KEYED BY clause is used to make a field into a record key. The arguments are

key-number	Identifies a key and is used in the Indexed file system. The number can range from 1 to (n), where (n) is the number of keys.
DUPLICATES ALLOWED	Allows two or more records to have the same key value. Without this parameter, an error will be returned when one record having the same key value as another is inserted.
DESCENDING	Reverses the ordering of keys. MS-COBOL will read an Indexed file with this feature in reverse order. However, MS-COBOL can neither create a DESCENDING key file, nor inform a program that a file has this feature. DESCENDING key files should be used only with programs that expect this feature.

SPLIT KEYSET Statement

The SPLIT KEYSET statement defines split keys. The parameters are

key-number	Serves the same purpose as the key number in the KEYED BY clause. Identifies a single key composed of several concatenated record fields.
field-name or *key-number*	*field-name* is the same field name used to identify the field in the FIELD statement. If no name was given in the FIELD statement, *key-number* can be substituted for *field-name*.
DESCENDING	Reverses the ordering of the preceding component of the split key. MS-COBOL reverses the ordering of keys, and will read an Indexed file with this feature in reverse order. However, MS-COBOL can neither create a DESCENDING key file, nor inform a program that a file has this feature. DESCENDING key files should be used only with programs that expect this feature.

11.7 Data Dictionary

The data file portion of a 2.0 format Indexed file contains a data dictionary.

If a data dictionary exists in the data file, it will always take precedence over any of the other ways of describing the record unless a data file copy and a dd-file have been supplied to REBUILD.

The two methods for modifying a data dictionary, specifying an ASCII record description file (dd-file), and using REBUILD's interactive facility, are described in Section 11.5, "Adding and Deleting Indexes."

11.8 ASCII Version of a Data Dictionary

In order for REBUILD to transfer data to an MS-COBOL Indexed file, RECORD, FIELD, KEYED BY, and SPLIT KEYSET information should be formatted as in the following example:

Example: *EMPLOYEE Identification Record*

```
RECORD EMPLOYEE;
 FIELD EMPLOYEE-NAME IS 1:30 ALPHA
   KEYED BY 1;
 FIELD EMPLOYEE-ADDRESS IS 31:40 ALPHA;
 FIELD SOCIAL-SECURITY IS 72:10 ALPHA;
 FIELD EMPLOYEE-SALARY IS 83:2 ALPHA;
 FIELD EMPLOYEE-NUMBER IS 85:2 ALPHA
   KEYED BY 2;
 FIELD DATE-OF-EMPLOYMENT IS 87:10 ALPHA
   KEYED BY 3;
 SPLIT KEYSET 4 IS EMPLOYEE-NAME
   SOCIAL-SECURITY EMPLOYEE-SALARY;
```

This example defines a record called EMPLOYEE. The name EMPLOYEE is optional and is present for readability. There are six fields and four keys for this record. The following discussion examines the function of each field.

Field 1: EMPLOYEE-NAME

The field type is ALPHA (ALPHANUMERIC), which means it is made up of characters. This is the only field type used when defining data records for MS-COBOL.

The field begins in position one — the first byte of the record. The field is 30 bytes in length. Finally, the field is also a key. The key number is a value ranging from 1 to (n) where (n) is the number of keys in the record. Key numbers should be assigned in the same order as the keys declared in the SELECT clause in the FILE-CONTROL entry. Key number 1 is always the prime key.

Field 2: EMPLOYEE-ADDRESS

The field is 40 bytes in length. It begins in position 31 in the record.

Field 3: SOCIAL-SECURITY

The field is 10 bytes in length. It begins in position 72 in the record.

Field 4: EMPLOYEE-SALARY

This field begins in position 83, and is two bytes in length. It allows duplicate values. A duplicate value occurs when two records have the same keyed field value. Identical non-keyed fields are not detected. The default for Field 1 is DUPLICATES NOT ALLOWED.

Field 5: EMPLOYEE-NUMBER

This field begins in position 85 and is two bytes in length. It is the second keyed field in the record and does not allow duplication of the value of this field by any record.

Field 6: DATE-OF-EMPLOYMENT

The field is 10 bytes long and starts in position 87. It is the fourth keyed field.

SPLIT KEY specification

The final item specified in Example 1, EMPLOYEE IDENTIFICATION RECORD, is not a field in the record, but a key made of up several keys. This is called a SPLIT KEY, and is used to gain quick access to records based upon more than one field. It is treated as a key in every respect except that it is made up of more than a single field. A field that is a member of a split key is called a component. The split key is constructed by concatenating the components in the order specified by the field names.

The most significant component is EMPLOYEE-NAME field, which has already been defined to be 30 bytes long starting at position one. A field may be both a key and a SPLIT KEY component. The second and third key components are DATE-OF-EMPLOYMENT and EMPLOYEE-SALARY and are defined in their own field statements. Note that EMPLOYEE-SALARY is not a key field and is defined only so that it can be named as a component in the split key.

Appendices

Appendix A
Differences: Microsoft COBOL 2.0 and Previous Versions

The significant differences between the current Microsoft COBOL system software and previous releases are:

- Microsoft COBOL 2.0 is validated by the Federal government as a HIGH-level implementation of COBOL. Previous versions were validated at LOW-INTERMEDIATE level.
- The user-directed linking process has been replaced by a built-in linking process in the compiler and runtime executor. As a result, the compiler now produces intermediate (.INT) files rather than object (.OBJ) files. Programs are now executed by entering "RUNCOB *filename*" rather than "*filename*".
- MS-DOS pathnames and the MS-DOS PATH environment are supported in both the compiler and the runtime system for system files and user-generated files. Compiler overlays are located by searching the current directory, then directories specified by the PATH environment.
- The MS-DOS exit code is set to 255 in case of fatal errors. This code may be tested with the MS-DOS "IF ERRORLEVEL" command.
- The runtime command line now supports a /S switch followed by a pathname for searching an explicit directory for compiled programs. The directory specified in this way must also contain the .INT files specified by the CALL and CHAIN statements. A /P switch that sends PRINTER files to a special disk file is also supported.

- Built-in subroutines, EXIST, RENAME, REMOVE, COMMAND, UPCASE, LOCASE, and EXCODE are included.
- Microsoft Rebuild Utility version 2.0 now accepts input files in ASCII format for updating a data file's control index or data dictionary.
- The Microsoft INSTALL Utility allows the runtime executor name to be specified, and allows it to be present in a directory on disk apart from the INSTALL files.
- Microsoft Multi-Key ISAM Facility version 2.0, included in the MS-COBOL runtime system, now supports Level 2 multi-key INDEXED file organization.
- A split-key syntax for Indexed files has been added as an extension to the ANSI 1974 COBOL Standard.
- FOREGROUND-COLOR and BACKGROUND-COLOR clauses have been added to the DATA DIVISION SCREEN SECTION.
- Full COPY REPLACING syntax is supported. COPY libraries are also supported.
- COMPUTATIONAL-4 (COMP-4) storage format has been added as a tool to reduce DATA DIVISION or data file storage requirements and to increase the execution speed of certain operations. This is an extension to the standard.
- File control for multi-user/multi-tasking systems has been implemented through file and record locking in the LOCKING IS {EXCLUSIVE | MANUAL | AUTOMATIC} clause. See Chapter 17 of the *Microsoft COBOL Reference Manual* for details.
- Tape-handling syntax and the RERUN clause as defined by ANSI 1974 COBOL are recognized by the compiler, but are not executed at runtime.
- OPTIONAL clause has been implemented for Sequential and Line Sequential files in the FILE-CONTROL Entry.
- MOVE, ADD, and SUBTRACT statements will now transfer CORRESPONDING data, and return data to multiple destinations.

- COMPUTE, DIVIDE, and MULTIPLY statements will return data to multiple destinations.
- REMAINDER clause has been implemented for the DIVIDE statement.
- OCCURS DEPENDING ON clause has been implemented for describing variable occurrences of table elements.
- Level 66 entries and the RENAMES clause for regrouping data fields are supported.
- SEGMENT-LIMIT clause for regrouping segment numbers is supported.
- CALL ON OVERFLOW is available for monitoring transfer of control for the overflow condition.
- CANCEL statement for releasing memory occupied by subprograms is implemented.
- ANSI 1974 COBOL alphabet-name entry in the SPECIAL-NAMES paragraph is present for collating files on a customized alphabet.
- ANSI 1974 COBOL SORT/MERGE statements with the COLLATING SEQUENCE clause are implemented.
- Arithmetic expressions and complex conditional expressions may be used in the IF statement, and in any other places where conditions may be used (PERFORM, SEARCH).
- Multiple-character figurative constants may be used.
- EJECT may be used as a mnemonic-name.
- The default source program tab settings are now set to every eight positions to conform with traditional tab stops. The /O compiler switch can be used to restore the tab settings used in previous MS-COBOL versions.

For more information about Microsoft COBOL extensions to ANSI 1974 COBOL, see the *Microsoft COBOL Reference Manual.*

Appendix B
Compiler Phases

The Microsoft COBOL Compiler creates an object code program from your source program. This is done by invoking five "phases," consisting of the root portion of the compiler, COBOL.EXE, and five overlays, COBOL0.OVR through COBOL4.OVR. These are the phases referenced by an error message such as "Compiler error in phase n."

Compilation is performed in two passes:

The first creates an intermediate version of the program, which is stored in a temporary file. The creation of the intermediate file is done in three steps:

Phase 0 (COBOL0.OVR) compiles the IDENTIFICATION and ENVIRONMENT DIVISIONS of the source program.

Phase 1 (COBOL1.OVR) compiles the DATA DIVISION of the source program.

Phase 2 (COBOL2.OVR) compiles the PROCEDURE DIVISION of the source program.

The second pass reads the intermediate file and creates the object code in two steps:

Phase 3 (COBOL3.OVR) reads the intermediate file and creates the object code.

Phase 4 (COBOL4.OVR) allocates file control blocks and finalizes the object code.

Appendix C
Tab Stop Customization

This appendix is intended for those who are proficient with a debugger and/or assembly language and would like to change the built-in source program tab stop parameters of Microsoft COBOL.

Default tab stops interpreted by COBOL are every 8 characters, through column 80:

8, 16, 24, 32, 40, 48, 56, 64, 72, 80

When a tab character (hex 09) is encountered in the source file, subsequent text is treated as if it began in the column following the next tab stop. Thus, in a line beginning with a tab and the letter "A", the "A" is interpreted as if in column 9.

Previous versions of the compiler assumed the following tab stops:

7, 11, 19, 27, 35, 43, 51, 59, 67, 72

The "/O" switch may be used to cause the compiler to interpret tabs using the old tab settings. (Note that old documentation refered to the stops as 8, 12, 20, etc. These are the columns following the tab stops, at which characters following the tabs are placed.)

The tab stops are maintained as a table. If you wish, you may modify this table to suit your needs by patching the compiler. The location of the default tab table may vary, based on the size of the .EXE file header. The following information may help you find this location.

The table is 10 bytes long, one byte per tab stop. Any values may be used, provided that:

1. The numbers are in ascending order.
2. No more than 10 stops are defined.
3. The last tab stop is 80.

Note that the "/O" switch, as mentioned above, takes precedence over the default tab stops, even if the defaults have been patched.

When using the MS-DOS DEBUG utility to change the tab settings, the following script may be used. A> is the MS-DOS prompt. - is the DEBUG prompt. You must rename COBOL.EXE to a name without the .EXE extension before using DEBUG.

Commands	**Comments**
A> COPY COBOL.EXE COBOL.SAV	Save original COBOL.EXE.
A> REN COBOL.EXE COBOL.NEW	Rename without .EXE.
A> DEBUG COBOL.NEW	Start DEBUG.
- d *nnnn*	Examine current settings. (must be 08 10 18 28 30 38 40 48 50). To determine nnnn, see "Finding the Tab Table."
- e *nnnnn*	Enter values now, in hexadecimal, followed by SPACE. The last entry is terminated by RETURN.
- d *nnnn*	Examine new settings.
- w	Write out file.
- q	End DEBUG.
A> REN COBOL.NEW COBOL.EXE	Rename back for use.

Finding the Tab Table

The tab table begins immediately after the .EXE file header. To find this location use DEBUG on COBOL.NEW, as described in the previous section.

The simplest method of finding the tab table is to search for the bytes that make up the table. The resulting location must be an address that ends in 00. The following search command will find the bytes 08 10 18 20 28 in the first 2000 (hexadecimal) bytes of the file:

```
-s 0 2000 08 10 18 20 28
```

The computer will respond with

```
????:nnnn
```

where

????	segment (ignored)
nnnn	the desired location

If nnnn does not end in 00, or no match is found, or more than one match is found, use the following method to find nnnn:

The size of the .EXE file header, in 16 byte paragraphs, is stored 8 bytes from the start of the .EXE file header. To find it, enter

```
-d 108
```

Take the first 2-digit hexadecimal number, and add hexadecimal 10 to it. Place a 0 on each end of the result, and you will have nnnn. For example, if the value at 108 was 40, adding 10 hex gives 50, and attaching zeroes gives 0500.

At the time this manual was printed, the value of nnnn was 0500.

Appendix D

INSTALL Terminal Database

This appendix contains a list of terminals included in the INSTALL data file (INSTALL.DAT) and shows the special key assignments for those systems. (See the disk file INSTALL.DOC on your distribution disk for the most current information on the terminals supported by the INSTALL program.)

Each list entry includes the functions, ASCII key names, and escape codes for a particular terminal. The description of the INSTALL general purpose default system is also included.

If your system is not one of those listed, check your system's technical manual for the appropriate values.

Table D.1 lists the escape codes which apply to all the terminals described in this section, including the default system.

Table D.2 lists the characteristics applied by INSTALL to the general purpose "default" system. These default values probably do not apply to your system. See your system's technical manual for the applicable values.

Table D.1.
Escape Codes

Function	Escape Code
Terminator Keys	
Backtab	99
Escape	01
Tab	00
Carriage Return	00
Line Feed	00
Function Keys	
Function 1	02
Function 2	03
Function 3	04
Function 4	05
Function 5	06
Function 6	07
Function 7	08
Function 8	09
Function 9	10
Function 10	11
Function 11	12
Function 12	13
Function 13	14
Function 14	15
Function 15	16

Table D.2.

Default System Interface

Function	ASCII Key Name
Editing Keys	
Delete Line	CTRL-U
Delete Character	DEL
Forward Space	CTRL-F
Back Space	CTRL-H
Plus Sign	+
Minus Sign	–
Terminator Keys	
Escape	ESCAPE
Back Tab	CTRL-B
Tab	CTRL-I or TAB
Carriage Return	RETURN
Line Feed	LINEFEED
Function Keys	
Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

The following output functions have no default interface in INSTALL:

Set Cursor Position

Backspace Cursor

Cursor On

Cursor Off

Erase to End of Screen

Erase to End of Line

Sound Bell

Start Highlight

End Highlight

Start Blink

End Blink

Start Reverse-Video

End Reverse-Video

Start Underline

End Underline

The following default system interface features have the default values indicated:

Screen Format	24 lines by 80 columns
System Name	Undefined
Terminal Initialization	Undefined
Reset COBOL	Undefined

AT&T® PC 6300

Editing Keys

Delete Line	CTRL-U or CTRL-End or CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key or BACK-SPACE key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or SHIFT TAB or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 0 or F10 or SHIFT F10
Function 11	CTRL-E A
Function 12	CTRL-E B
Function 13	CTRL-E C
Function 14	CTRL-E D
Function 15	CTRL-E E

Screen Attributes available:

HIGHLIGHT BLINK REVERSE-VIDEO.

UNDERLINE available under monochrome adapter only. UNDERLINE appears as HIGHLIGHT under graphics adapter.

Color available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	0	black
1	blue	1	blue
2	green	2	green
3	cyan	3	cyan
4	red	4	red
5	magenta	5	magenta
6	brown	6	brown
7	white	7	white
8	gray	8	blinking black
9	light blue	9	blinking blue
10	light green	10	blinking green
11	light cyan	11	blinking cyan
12	light red	12	blinking red
13	light magenta	13	blinking magenta
14	yellow	14	blinking brown
15	high-intensity white	15	blinking white

COMPAQ™

Editing Keys

Delete Line	CTRL-U or CTRL-End or CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key or BACK-SPACE key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or SHIFT TAB or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 0 or F10 or SHIFT F10
Function 11	CTRL-E A
Function 12	CTRL-E B
Function 13	CTRL-E C
Function 14	CTRL-E D
Function 15	CTRL-E E

Screen Attributes available:

HIGHLIGHT BLINK REVERSE-VIDEO.

UNDERLINE available under monochrome adapter only. UNDERLINE appears as HIGHLIGHT under graphics adapter.

Colors available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	0	black
1	blue	1	blue
2	green	2	green
3	cyan	3	cyan
4	red	4	red
5	magenta	5	magenta
6	brown	6	brown
7	white	7	white
8	gray	8	blinking black
9	light blue	9	blinking blue
10	light green	10	blinking green
11	light cyan	11	blinking cyan
12	light red	12	blinking red
13	light magenta	13	blinking magenta
14	yellow	14	blinking brown
15	high-intensity white	15	blinking white

DEC® Rainbow™ 100

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC (F11)
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINE FEED (F13)

Function Keys

Function 1	CTRL-E 1 or PF1
Function 2	CTRL-E 2 or PF2
Function 3	CTRL-E 3 or PF3
Function 4	CTRL-E 4 or PF4
Function 5	CTRL-E 5 or F14 (additional options)
Function 6	CTRL-E 6 or F16 (DO)
Function 7	CTRL-E 7 or F17
Function 8	CTRL-E 8 or F18
Function 9	CTRL-E 9 or F19
Function 10	CTRL-E 0 or F20

Note

Only the above noted Rainbow function keys are used by MS-COBOL. Keys PF1 through PF4 on the numeric keypad may be used as function keys 1 through 4.

DEC VT™-52

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

Screen Attributes available:

Vary by machine. Non-available attributes may be represented by bracketing in.

Color not available.

DEC VT-100 (ANSI mode)

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

Screen Attributes available:

UNDERLINE HIGHLIGHT BLINK REVERSE-VIDEO

Color not available.

Heath® / Zenith® 19

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 key
Function 2	CTRL-E 2 or F2 key
Function 3	CTRL-E 3 or F3 key
Function 4	CTRL-E 4 or F4 key
Function 5	CTRL-E 5 or F5 key
Function 6	CTRL-E 6 or ERASE key
Function 7	CTRL-E 7 or blue key
Function 8	CTRL-E 8 or red key
Function 9	CTRL-E 9 or white key
Function 10	CTRL-E 0 or SHIFT-ERASE key

Note

The key pad is enabled in unshifted mode. Use SHIFT-2 for down arrow, SHIFT-4 for left arrow, SHIFT-6 for right arrow, and SHIFT-8 for up arrow.

Screen Attributes available:

UNDERLINE, HIGHLIGHT, BLINK, and REVERSE-VIDEO all appear as REVERSE-VIDEO.

Color not available.

Hyperion™

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESCAPE
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

Screen Attributes available:

UNDERLINE, HIGHLIGHT, BLINK, and REVERSE-VIDEO all appear as REVERSE-VIDEO.

Color not available.

IBM® Displaywriter

Editing Keys

Delete Line	CODE-U or CODE-End or CODE-X
Delete Character	DEL key or BACKSPACE key
Forward Space	CODE-F or right arrow key
Back Space	CODE-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CODE-B or SHIFT TAB or up arrow key
Tab	CODE-I or TAB or down arrow key
Carriage Return	CODE-M or RETURN
Line Feed	CODE-J or LINEFEED

Function Keys

Function 1	CODE-E 1 or F1 or SHIFT F1
Function 2	CODE-E 2 or F2 or SHIFT F2
Function 3	CODE-E 3 or F3 or SHIFT F3
Function 4	CODE-E 4 or F4 or SHIFT F4
Function 5	CODE-E 5 or F5 or SHIFT F5
Function 6	CODE-E 6 or F6 or SHIFT F6
Function 7	CODE-E 7 or F7 or SHIFT F7
Function 8	CODE-E 8 or F8 or SHIFT F8
Function 9	CODE-E 9 or F9 or SHIFT F9
Function 10	CODE-E 0 or F10 or SHIFT F10

Screen Attributes available:

UNDERLINE HIGHLIGHT BLINK REVERSE-VIDEO

Color not available.

IBM PC

Editing Keys

Delete Line	CTRL-U or CTRL-End or CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key or BACK-SPACE key
Plus Sign	+
Minus Sign	−

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or SHIFT TAB or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 0 or F10 or SHIFT F10
Function 11	CTRL-E A
Function 12	CTRL-E B
Function 13	CTRL-E C
Function 14	CTRL-E D
Function 15	CTRL-E E

Screen Attributes available:

HIGHLIGHT BLINK REVERSE-VIDEO.

UNDERLINE available under monochrome adapter only. UNDERLINE appears as HIGHLIGHT under graphics adapter.

Color available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	0	black
1	blue	1	blue
2	green	2	green
3	cyan	3	cyan
4	red	4	red
5	magenta	5	magenta
6	brown	6	brown
7	white	7	white
8	gray	8	blinking black
9	light blue	9	blinking blue
10	light green	10	blinking green
11	light cyan	11	blinking cyan
12	light red	12	blinking red
13	light magenta	13	blinking magenta
14	yellow	14	blinking brown
15	high-intensity white	15	blinking white

Lear Siegler® ADM 42 Ergonomic Terminal™ Video Display

Editing Keys

Delete Line	CTRL-U or delete line key
Delete Character	DEL key or delete char key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 key
Function 2	CTRL-E 2 or F2 key
Function 3	CTRL-E 3 or F3 key
Function 4	CTRL-E 4 or F4 key
Function 5	CTRL-E 5 or F5 key
Function 6	CTRL-E 6 or F6 key
Function 7	CTRL-E 7 or F7 key
Function 8	CTRL-E 8 or F8 key
Function 9	CTRL-E 9 or F9 key
Function 10	CTRL-E 0 or F10 key

Screen Attributes not available.

Color not available.

Microsoft MS-DOS 2.xx ANSI Device Driver

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F
Back Space	CTRL-H
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B
Tab	CTRL-I or TAB
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

Screen Attributes available:

UNDERLINE HIGHLIGHT BLINK REVERSE-VIDEO

Color not available.

Victor® 9000 and Sirius™ 1

Editing Keys

Delete Line	ALT-U, ALT-X
Delete Character	DEL key
Forward Space	ALT-F or right arrow key
Back Space	ALT-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	CLR HOME
Back Tab	ALT-B or ALT-TAB or up arrow key
Tab	ALT-I or TAB or down arrow key
Carriage Return	ALT-M or RETURN
Line Feed	ALT-J or LINEFEED

Function Keys

Function 1	ALT-E 1 or F1
Function 2	ALT-E 2 or F2
Function 3	ALT-E 3 or F3
Function 4	ALT-E 4 or F4
Function 5	ALT-E 5 or F5
Function 6	ALT-E 6 or F6
Function 7	ALT-E 7 or F7
Function 8	ALT-E 8 or F8
Function 9	ALT-E 9 or F9
Function 10	ALT-E 0 or F10
Function 11	ALT-E A
Function 12	ALT-E B
Function 13	ALT-E C
Function 14	ALT-E D
Function 15	ALT-E E

Screen Attributes available:

UNDERLINE HIGHLIGHT REVERSE-VIDEO.
BLINK appears as UNDERLINE.

Color not available.

Soroc IQ® 120

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1
Function 2	CTRL-E 2
Function 3	CTRL-E 3
Function 4	CTRL-E 4
Function 5	CTRL-E 5
Function 6	CTRL-E 6
Function 7	CTRL-E 7
Function 8	CTRL-E 8
Function 9	CTRL-E 9
Function 10	CTRL-E 0

Screen Attributes not available.

Color not available.

TeleVideo® 925/950

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 key
Function 2	CTRL-E 2 or F2 key
Function 3	CTRL-E 3 or F3 key
Function 4	CTRL-E 4 or F4 key
Function 5	CTRL-E 5 or F5 key
Function 6	CTRL-E 6 or F6 key
Function 7	CTRL-E 7 or F7 key
Function 8	CTRL-E 8 or F8 key
Function 9	CTRL-E 9 or F9 key
Function 10	CTRL-E 0 or F10 key

Screen Attributes not available.

Color not available.

Texas Instruments Professional Computer

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key or backspace key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or BACKTAB or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 0 or F10 or SHIFT F10
Function 11	CTRL-E A or F11 or SHIFT F11
Function 12	CTRL-E B or F12 or SHIFT F12
Function 13	CTRL-E C
Function 14	CTRL-E D
Function 15	CTRL-E E

Screen Attributes available:

BLINK REVERSE-VIDEO UNDERLINE

Colors available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	not supported (no effect if	
1	blue	BACKGROUND-COLOR clause	
2	green	is used)	
3	cyan		
4	red		
5	magenta		
6	brown		
7	white		

(colors 8 – 15 are the same as 0 – 7)

Wang® Professional Computer

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DELETE
Forward Space	CTRL-F or RIGHT ARROW
Back Space	CTRL-H or LEFT ARROW or BACKSPACE
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC (HELP)
Back Tab	CTRL-B or BACK TAB or UP ARROW
Tab	CTRL-I or TAB or DOWN ARROW
Carriage Return	RETURN
Line Feed	LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 10 or F10 or SHIFT F10

Screen Attributes available:

UNDERLINE HIGHLIGHT BLINK REVERSE-VIDEO

Color not available.

Zenith Data Systems™ z-100™

Editing Keys

Delete Line	CTRL-U, CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key
Plus Sign	+
Minus Sign	−

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 key
Function 2	CTRL-E 2 or F2 key
Function 3	CTRL-E 3 or F3 key
Function 4	CTRL-E 4 or F4 key
Function 5	CTRL-E 5 or F5 key
Function 6	CTRL-E 6 or F6 key
Function 7	CTRL-E 7 or F7 key
Function 8	CTRL-E 8 or F8 key
Function 9	CTRL-E 9 or F9 key
Function 10	CTRL-E 0 or F10 key

Screen Attributes available:

UNDERLINE, HIGHLIGHT, BLINK, and REVERSE-VIDEO all appear as REVERSE-VIDEO.

Color available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	0	black
1	blue	1	blue
2	green	2	green
3	cyan	3	cyan
4	red	4	red
5	magenta	5	magenta
6	brown	6	brown
7	white	7	white

(colors 8 – 15 are the same as 0 – 7)

Zenith Data Systems z-100-PC™ (z-150™)

Editing Keys

Delete Line	CTRL-U or CTRL-End or CTRL-X
Delete Character	DEL key
Forward Space	CTRL-F or right arrow key
Back Space	CTRL-H or left arrow key or BACK-SPACE key
Plus Sign	+
Minus Sign	–

Terminator Keys

Escape	ESC
Back Tab	CTRL-B or SHIFT TAB or up arrow key
Tab	CTRL-I or TAB or down arrow key
Carriage Return	CTRL-M or RETURN
Line Feed	CTRL-J or LINEFEED

Function Keys

Function 1	CTRL-E 1 or F1 or SHIFT F1
Function 2	CTRL-E 2 or F2 or SHIFT F2
Function 3	CTRL-E 3 or F3 or SHIFT F3
Function 4	CTRL-E 4 or F4 or SHIFT F4
Function 5	CTRL-E 5 or F5 or SHIFT F5
Function 6	CTRL-E 6 or F6 or SHIFT F6
Function 7	CTRL-E 7 or F7 or SHIFT F7
Function 8	CTRL-E 8 or F8 or SHIFT F8
Function 9	CTRL-E 9 or F9 or SHIFT F9
Function 10	CTRL-E 0 or F10 or SHIFT F10
Function 11	CTRL-E A
Function 12	CTRL-E B
Function 13	CTRL-E C
Function 14	CTRL-E D
Function 15	CTRL-E E

Screen Attributes available:

HIGHLIGHT BLINK REVERSE-VIDEO.

UNDERLINE available under monochrome adapter only. UNDERLINE appears as HIGHLIGHT under graphics adapter.

Color available:

FOREGROUND-COLOR		BACKGROUND-COLOR	
Integer	Color	Integer	Color
0	black	0	black
1	blue	1	blue
2	green	2	green
3	cyan	3	cyan
4	red	4	red
5	magenta	5	magenta
6	brown	6	brown
7	white	7	white
8	gray	8	blinking black
9	light blue	9	blinking blue
10	light green	10	blinking green
11	light cyan	11	blinking cyan
12	light red	12	blinking red
13	light magenta	13	blinking magenta
14	yellow	14	blinking brown
15	high-intensity white	15	blinking white

Appendix E
Guide to the MS-COBOL Demonstration Programs

The following demonstration programs are included with the MS-COBOL Compiler:

E.1 CRTEST

CRTEST is a test program for the terminal interface, as modified by the INSTALL utility program. CRTEST must be compiled before it can be run. (Follow directions for compiling in Section 2.4, "Sample Session.") When you run the program, it will prompt you for input.

E.2 CENTER

CENTER is a program that centers a line of text or aligns it with the left or right margin. It is a simple MS-COBOL program that does not use sophisticated screen-handling features. CENTER.INT is provided on the demonstration disk. It does not need to be compiled.

E.3 MS-COBOL Demonstration System

The MS-COBOL demonstration system consists of three MS-COBOL programs:

DEMO.COB
BUILD.COB
UPDATE.COB

These programs must be compiled before they are run. You can run the batch file CLDEMO.BAT (described later in this section) to compile DEMO.COB, BUILD.COB, and UPDATE.COB.

DEMO is the executive program of the system. It asks if you would like a demonstration of the MS-COBOL SCREEN SECTION, or whether you would like to create or update an Indexed (ISAM) file of names, addresses, and phone numbers.

Use the following procedure to compile and run DEMO:

1. Run INSTALL (see Chapter 9, "INSTALL Program," for details).
2. Include drive A: on the MS-DOS PATH by entering

   ```
   PATH A:\
   ```

3. Load the Indexed file handler (see Appendix G, "Loading the Indexed File Handler") by putting a disk containing ISAM.EXE in the default drive and typing "ISAM".
4. Compile the system.
   a. Copy DEMO.COB, DEMO.CPY, BUILD.COB, UPDATE.COB, and CLDEMO.BAT onto a blank disk, and insert into drive B:.
   b. Place the disk containing COBOL.EXE and its overlays into drive A:.
   c. Type "B:" at the system prompt to make drive B: the default drive.
   d. Type "CLDEMO" to compile the programs and create files DEMO.INT, DEMO.050, BUILD.INT, and UPDATE.INT.
5. Remove the compiler disk from drive A: and replace it with a disk containing RUNCOB.EXE.
6. Now type "RUNCOB DEMO".

   When DEMO has been loaded, it will ask if INSTALL has been run. If INSTALL hasn't been run, cancel the job. (If you continue, you will have to restart the system to exit from DEMO.)

   If INSTALL has been run, DEMO will prompt you for input by providing menus and information screens to guide you through the demonstration.

Note that CLDEMO.BAT uses the /D switch to prevent the .DBG files used by DEBUGCOB from being created.

Appendix F
Error Messages

This appendix lists all the error messages you may encounter while compiling and executing a Microsoft COBOL program.

F.1 Compile Time Error Messages

Compile time error messages include messages for

1. command input and operating system I-O errors
2. program syntax errors
3. file usage errors
4. warnings

Command input and operating system input-output error messages will be displayed as errors occur during compilation.

Program syntax error messages (which include file usage and warning messages) are displayed as they occur during compilation, and are placed at the end of the listing file. They consist of

1. The source program line number, which is four digits followed by a colon (:).
2. An explanation of the error. If the explanation begins with an /F/ (inconsistent file usage) or a /W/ (warning), then the message is only a warning; if not, the error is severe enough to prevent you from executing the object file.

Program syntax error messages will always be listed at your terminal at the end of compilation. A message displaying the total number of errors or warnings is also displayed. This feature allows you to make a simple change to a program, recompile it without a listing, and still receive any error messages at your terminal.

Syntax error messages in this manual are listed in alphabetical order, with /F/ (file usage) and /W/ (warning) placed at the end of the list. The number included with an /F/ warning represents the order in which files are entered in the FILE SECTION of the MS-COBOL program, rather than a line number.

F.1.1 Command Input and Operating System I-O Errors

Cannot find overlay for loading.

One of the MS-COBOL Compiler overlay files (COBOLn.OVR) is not on the disk. It may have been written to another disk or destroyed. Recompiling and relinking may eliminate the problem. Be sure that the disk containing the overlays is either in the current directory of the current drive, or is in a directory in the search path specified by the MS-DOS PATH command.

Compiler error in phase n at address.

Usually caused by a damaged source program or damaged compiler or overlay file. In the latter case, try your backup copy. If this does not work, you can sometimes determine the cause of the error by compiling increasingly larger portions of the program, starting with only a few lines, until the error reoccurs. See Appendix B, "Compiler Phases," for more information.

Compiler table overflow.

One of the compiler's internal tables has used up all memory available to it. See the "Out of memory" error description.

Debug file cannot be opened.

The debug file (.DBG file) cannot be opened.

Disk input error.

An input-output operation on an input file has been unsuccessful.

Disk output error.

An input-output operation on an output file has been unsuccessful. This usually indicates that the disk is full.

Error while reading overlay.

The compiler's overlay was found but is damaged in some way. Restore a new copy of the compiler and its overlays from your backup disk.

Invalid switch.

You have entered a switch parameter that the compiler does not recognize.

List file cannot be opened.

The listing file (.LST file) cannot be opened.

Object file cannot be opened.

An output file cannot be opened. For example, the output disk is write-protected.

Out of memory.

Occurs when there is insufficient memory for all the symbols and other information obtained from the source program. It indicates that the program is too large and must be decreased in size, or split into modules and compiled separately.

The symbol table of data-names and procedure-names is usually the largest user of space during compilation. All names require as many bytes as there are characters in the name, with an overhead requirement of about 10 bytes per data-name and 2 bytes per procedure-name. On the average, each line in the DATA DIVISION uses about 14 bytes of memory during compilation, and each line in the PROCEDURE DIVISION uses about 3 1/4 bytes.

Source file cannot be found.

You have specified a filename for input that does not exist, or have specified an invalid filename.

F.1.2 Program Syntax Errors

A FILE-ID name is undefined.

A data-name specified in a VALUE OF FILE-ID clause is not defined.

A paragraph declaration is required here.

An EXIT statement is not followed by a section or paragraph header, or the PROCEDURE DIVISION header.

Area A not blank in continuation line.

A character was encountered in Area A of a continuation line.

Area A violation; resumption at next paragraph/section/division/verb.

The entry starting in one of columns 8-12 cannot be interpreted as a division header, section name, paragraph name, file description indicator, or 01 or 77 level number.

Bad SORT/RELEASE/RETURN usage.

A file described by an SD was OPENed or otherwise acted upon by something other than a SORT statement.

Clauses other than VALUE deleted.

The data-description of a level 88 item includes a descriptive clause other than VALUE IS.

COPY file not found.

The filename for the COPY file is invalid, or the COPY file cannot be found.

DATA DIVISION limitations exceeded.

One of the limitations on the contents of the DATA DIVISION has been exceeded. Check the *Microsoft COBOL Reference Manual*, Section 6.3, for DATA DIVISION limits.

Duplicate alphabet name.

Two or more alphabet definitions in the SPECIAL NAMES paragraph were given the same name.

Element length error.

The length of the quoted literal is over 120 characters; or the numeric literal is over 18 digits; or the identifier/name is over 30 characters.

Erroneous filename is ignored.

An entry which has not been declared as a filename appears where a filename is required.

Erroneous qualification; last declaration used.

The qualifiers used with a data-name are incorrect or are not unique.

Erroneous subscripting; statement deleted.

Too few or too many subscripts are provided for a data-name.

Excessive literal pool or display string length.

The total length of the literals contained within a single paragraph is greater than 4096 bytes.

Excessive OCCURS nesting is ignored.

OCCURS clauses are nested more than three deep.

Excessive segment number.

A section header contains a section number greater than 99.

Excessive segment number in DECLARATIVES.

A section header in the DECLARATIVES Region contains a section number greater than 49.

File not selected; entry bypassed.

An FD is given for a file-name which does not appear in any SELECT sentence.

File share syntax error.

LOCK, WAIT, LOCKING MANUAL, or LOCKING AUTOMATIC was specified for a Sequential or Line Sequential file.

File status declaration of this file is not correct.

The file status data-item is not alphanumeric.

Fill character conflict.

In a Format 3 ACCEPT statement, SPACE-FILL and ZERO-FILL are both specified.

First non-DECLARATIVE SECTION segment number cannot exceed 49.

The first section segment number following the optional DECLARATIVES Section must be less than 50.

Fractional exponent or negative scaled base (99p).

In a COMPUTE statement, an exponent is a numeric literal with a decimal point or a numeric data-item described with a digit to the right of an assumed decimal point; or the PICTURE of an exponentiation base (entry preceding) contains the character P as the rightmost digit.

Group item, therefore PIC/JUST/BLANK/SYNC is ignored.

A phrase which is allowed for only elementary data-items is used in the description of an item that is followed immediately by an item of a higher level number.

Illegal character.

An invalid character has been encountered. This can often be caused by older text editors.

Illegal MOVE or comparison is deleted.

The operands of a MOVE statement or relational condition are incompatible.

Imperative statement required. Statement deleted.

A conditional statement is contained within a conditional statement other than IF.

Improper item to control # of occurrences.

The DEPENDING ON data-name is not a proper identifier or does not refer to a numeric integer

Improper # of split key components.

More than 31 components have been specified or a psuedo-name has already been defined, or one of the names following the equal sign (=) has not been defined.

Improper character in column 7.

An invalid character in column 7 has been encountered.

Improper PICTURE. PIC X assumed.

An invalid PICTURE clause has been encountered.

Improper punctuation.

Incorrect punctuation has been encountered. For instance, a comma or period must be followed by a space.

Improper redefinition ignored.

The data-name specified in a REDEFINES clause is not at the same level as the current data-name, or it is separated from it by an item with a lower level number.

Improperly formed element.

Incorrect syntax for an item has been encountered. For instance, you may have ended a word with a hyphen or used multiple decimal points in a numeric literal.

Incomplete (or too long) statement deleted.

A verb immediately follows a partial statement form, or an otherwise acceptable statement is too large for the compiler to read.

Indexed/Relative requires disk assignment.

A file assigned to PRINTER is described as having INDEXED or RELATIVE organization.

Invalid key specification.

The key item for a Relative or Indexed file should not be subscripted, or it is inconsistent with the file organization in class or USAGE. This message is issued when the OPEN statement is processed.

Invalid quoted literal.

A literal of zero length, improper construction, or missing end quotes has occurred.

Invalid SELECT sentence.

The syntax of a SELECT sentence in the FILE-CONTROL paragraph is incorrect.

Invalid VALUE ignored.

The value specified in a VALUE IS phrase is not a properly formed literal.

Justification conflict.

In a Format 3 ACCEPT statement, LEFT-JUSTIFY and RIGHT-JUSTIFY are both specified.

Key declaration of this file is not correct.

The RELATIVE KEY clause is missing for a Relative file, or the RECORD KEY clause is missing for an Indexed file. A RECORD KEY or RELATIVE KEY clause was specified for a file with SEQUENTIAL or LINE SEQUENTIAL organization.

LINAGE-COUNTER may not be modified.

An illegal attempt was made to change the value of the LINAGE-COUNTER special register.

Literal truncated to size of item.

The literal specified in a VALUE IS phrase is larger than the data-item being declared.

Maximum number of SORT keys is 12.

A SORT or MERGE statement with more than 12 keys was encountered.

Misordered/redundant section processed as is.

A section in the IDENTIFICATION, ENVIRONMENT, or DATA DIVISION is out of order or repeated.

Name omitted; entry bypassed.

The data-name is missing in a data description entry.

No items 'CORRESPOND'.

In a MOVE, ADD, or SUBTRACT operation with the CORRESPONDING option, no items were found to correspond. The operation is ignored.

No msg for msg no.

This message indicates an internal compiler problem.

No PICTURE; elementary item assumed to be binary.

No PICTURE is given for an elementary data-item.

OCCURS disallowed at level 01/77, or count too high.

An OCCURS clause appears in a data-description entry at level 01 or 77; or the number of occurrences specified is greater than 32,767.

Omitted word SECTION is assumed here.

The required word SECTION is missing from the header of a section in the DATA DIVISION.

Over 255 values in 88 condition.

Level 88 condition names are limited to 255 values or ranges.

Procedure-name is unresolvable.

A reference to a section name or procedure-name is not sufficiently qualified or is not unique.

Procedure range not in current segment.

A PERFORM statement in a section with a number greater than 49 refers to a procedure in a section with a different number greater than 49.

Procedure range spans segments.

A procedure range (procedure-name-1 THRU procedure-name-2) mentioned in a PERFORM statement contains paragraphs in sections with different section numbers greater than 49, or in sections numbered both less than or equal to 49 and greater than 49.

Redundant FD processed as is.

The same filename appears in more than one file description.

REWRITE valid only for a disk file.

The file-name entry in a REWRITE statement is a file assigned to PRINTER.

Semantical error in screen description.

This message can be caused by five error conditions:

1. The SCREEN SECTION does not begin with a level 01 screen item description.
2. A level 01 screen item description does not include a screen name.
3. A group screen item is described with a clause which is allowed only for elementary items.
4. An elementary screen item description is missing in FROM, TO, USING, or VALUE clauses.
5. A screen item description contains inconsistent clauses (such as USING and VALUE).

SIGN clause ignored for unsigned item.

The PICTURE of a numeric item with USAGE IS DISPLAY describes it as unsigned, but a SIGN IS clause is present.

Single-spacing assumed due to improper ADVANCING count.

The operand of the BEFORE or AFTER phrase of a WRITE statement is not numeric, or it is outside the range 0-120.

Source bypassed until next FD/section.

An error in a file description prevents further analysis.

Statement deleted because integer item is required.

A numeric data-item whose PICTURE specifies digits to the right of the decimal point is used where an integer is required.

Statement deleted because operand is not a filename.

A name appearing where a filename is required has not been declared as a filename.

Statement deleted due to erroneous syntax.

A syntax error, to which no more specific message applies, is present.

Statement deleted due to non-numeric operand.

An alphanumeric or alphanumeric-edited item is used as an operand of an arithmetic statement; a numeric-edited item is used as an operand other than the result; or a number is longer than 18 digits.

Subscript 0 or over maximum number of occurrences; 1 used.

A literal used as a subscript is inconsistent with the range defined by the associated OCCURS clause.

Subscript or index-name is not unique.

A name which requires qualification is used as a subscript.

Syntax error in screen description.

A screen item description contains a clause which is unrecognizable, improperly constructed, or redundant.

Unrecognizable element is ignored.

A required keyword is missing, or a data-name or procedure name is unidentified.

Using-list item level must be 01/77.

A name used in the PROCEDURE DIVISION header USING list is not declared at level 01 or level 77.

USING/GIVING file's ACCESS MODE must be SEQUENTIAL.

In a SORT or MERGE statement, the files specified in a USING or GIVING clause must have been declared with ACCESS MODE IS SEQUENTIAL in their FILE-CONTROL entries.

VALUE disallowed--OCCURS/REDEFINES/type/size conflict.

The VALUE IS clause is specified for a data-item described with (or included within an item described with) an OCCURS or REDEFINES clause; or the literal given in a VALUE IS clause is not compatible with the PICTURE of the declared item.

VALUE OF FILE-ID required.

The VALUE OF FILE-ID clause is not specified in the file description of a file assigned to DISK.

Varying item may not be subscripted.

The data-item controlled by the VARYING phrase of a PERFORM statement is subscripted.

F.1.3 File Usage Errors

/F/ File never closed.

No CLOSE statement is present for the file.

/F/ File never opened.

No OPEN statement is present for the file.

/F/ Inconsistent READ usage.

An OPEN INPUT statement is present for a file, but no READ statement; or vice versa.

/F/ Inconsistent WRITE usage.

An OPEN OUTPUT statement is present for a file, but no WRITE statement; or vice versa.

F.1.4 Warnings

/W/ BLANK WHEN ZERO is disallowed.

The BLANK WHEN ZERO phrase appears in the description of an alphanumeric or alphanumeric-edited item.

/W/ 'COMP' ignored for decimal item.

An item declared with USAGE COMP-0 or COMP-4 had a PICTURE beyond the allowable size for the USAGE.

/W/ DATA DIVISION assumed here.

The DATA DIVISION header is missing.

/W/ DATA RECORDS clause was inaccurate.

The record-name(s) given in a DATA RECORDS clause is not consistent with the record descriptions following the file description.

/W/ Erroneous RERUN entry is ignored.

A RERUN clause of the I-O-CONTROL paragraph contains a syntax error.

/W/ FD-value ignored since LABELS ARE OMITTED.

The VALUE OF FILE-ID clause is used in the description of a file which is assigned to PRINTER.

/W/ FILE SECTION assumed here.

The FILE SECTION header is missing.

/W/ FIPS COBOL language extension.

When FIPS flagging has been selected, it indicates that the statement or clause identified is a Microsoft extension.

/W/ FIPS High-level feature.

When FIPS flagging has been selected, it indicates that the statement or clause identified is a High-level feature.

/W/ FIPS High-Intermediate level feature.

When FIPS flagging has been selected, it indicates that the statement or clause identified is a High-Intermediate level feature.

/W/ FIPS Low-Intermediate level feature.

When FIPS flagging has been selected, it indicates that the statement or clause identified is a Low-Intermediate level feature.

/W/ Invalid Blocking is ignored.

The BLOCK clause of an FD contains an error.

/W/ Invalid record size(s) ignored.

The RECORD clause of an FD contains an error.

/W/ LABEL RECORD STANDARD required.

The LABEL RECORD(S) STANDARD phrase is not present in the FD of a file assigned to DISK.

/W/ LABEL RECORDS OMITTED assumed for Printer file.

The LABEL RECORDS OMITTED clause is missing in the file description of a file assigned to PRINTER.

/W/ Level 01 assumed.

A record-description begins with a level number other than 01.

/W/ Period assumed after procedure-name definition.

A section or paragraph header does not end with a period.

/W/ PICTURE ignored for index item.

A data-item described with USAGE IS INDEX phrase also has a PICTURE phrase.

/W/ PROCEDURE DIVISION assumed here.

The PROCEDURE DIVISION header is missing.

/W/ Record size inconsistent with FD. RECORD CONTAINS clause ignored.

The record size specified in the RECORD CONTAINS clause of an FD is inconsistent with the sizes of the associated record-descriptions.

/W/ Redundant clause ignored.

The same clause is specified more than once in a file description.

/W/ Right parenthesis required after subscripts.

The closing parenthesis for a subscript is missing.

/W/ Terminal period assumed above.

A data-description entry or paragraph does not end with a period.

/W/ WORKING-STORAGE assumed here.

The WORKING-STORAGE header is missing.

F.2 Runtime Error Messages

Some programming errors cannot be detected by the compiler but cause the program to end prematurely during execution. These are program execution errors (runtime errors). They are displayed in the format

```
**Run-time error: reason,
  Program: program-id, line: line-number*
```

The program execution errors detected by MS-COBOL are:

Can't find code file

The ".INT" file containing the program to be run or called could not be found.

Cursor position

Attempt was made to position the cursor beyond the line or column limits of the screen. A Format 3 or 4 ACCEPT statement or a DISPLAY statement with a position-spec or screen-name is the statement responsible for the error. If a screen has been displayed or accepted, one or more fields within the screen have starting positions outside the maximum screen line or column.

Data unavailable

Reference was made to a data record in a file that was not open or had reached the AT END condition.

DELETE; no READ

Attempt was made to DELETE a record of a SEQUENTIAL access mode file when the last operation was not a successful READ.

File locked

OPEN attempted after an earlier CLOSE WITH LOCK.

File not OPENed/previous error

An attempt was made to perform an I/O operation on a file that was not OPENed or was damaged in some way due to a previous error.

GO TO not set

A null GO TO statement, which has never been altered to refer to a destination, was executed.

Illegal ALTER

Attempt was made to alter a nonalterable GO TO.

Illegal CANCEL

Attempt was made to CANCEL an active subprogram or one which was never called.

Illegal DELETE

Relative or Indexed file not opened for I-O.

Illegal READ

Attempt was made to READ a file that was not open in the INPUT or I-O mode.

Illegal RELEASE

A RELEASE statement was encountered when no SORT statement was active.

Illegal RETURN

A RETURN statement was encountered when no SORT or MERGE statement was active.

Illegal REWRITE

REWRITE was executed for a record in a file not open in the I-O mode.

Illegal START

File not opened for INPUT or I-O.

Illegal WRITE

Attempt was made to WRITE to a file that was not open in the OUTPUT mode for SEQUENTIAL access files, or in the OUTPUT or I-O mode for RANDOM or DYNAMIC access files.

Incompatible runtime and compiler used

A program was compiled using one version of the MS-COBOL Compiler and run with a runtime executor that had a different version number.

Indexed file system not available

In an MS-COBOL implementation with a separate Indexed file handler program, the file handler has not been run before Indexed I/O was attempted.

Input/output

Unrecoverable I-O error, with no provision in the user's MS-COBOL program for acting upon the situation by way of an AT END clause, INVALID KEY clause, FILE STATUS item, or DECLARATIVES SECTION.

Insufficient memory for program

During a CALL or CHAIN statement, sufficient memory to load the called or chained program was unavailable.

MS-DOS version prior to 2.0 used

Versions 2.0 and later of MS-COBOL require MS-DOS with version 2.0 or later.

Need more memory

The indexed file manager or sorter has ended abnormally because of insufficient dynamically allocatable memory.

Non-numeric data

Whenever the content of a numeric item does not conform to the given PICTURE, this condition may arise. Always check input data, if it is subject to error (because input editing has not yet been done) by using the NUMERIC test.

Object code error

An undefined object program instruction has been encountered. This should occur only if the absolute version of the program has been damaged in memory or on the disk file.

PERFORM overlap

An illegal sequence of PERFORMs, as, for example, when paragraph A is performed and another PERFORM A is initiated prior to exiting from the first, has occurred.

PERFORMS too deeply nested

The number of currently active paragraphs being PERFORMed has exceeded the maximum of 40.

READ beyond EOF.

An attempt was made to read beyond the end of a file.

READ error on code file

The runtime found but was unable to successfully READ a given code (".INT") file.

Redundant CLOSE

A CLOSE operation was performed on an unopened file.

Redundant OPEN

An OPEN operation was executed on a file that had already been opened.

REWRITE; no READ

Attempt was made to REWRITE a record of a SEQUENTIAL access file when the last operation was not a successful READ.

Segment nn load error

An error occurred while attempting to load an overlay segment. nn is 31 hex (49 decimal) less than the overlay segment number.

SORT error nn

An error has occurred during a SORT operation. See Chapter 13, "SORT/MERGE Facility" in the *Microsoft COBOL Reference Manual* for a description of the errors.

Stopped by Interrupt key

The interrupt key was pressed, which caused the program to close files and stop.

Subroutine language not supported

A non-COBOL subroutine entered in the user-supplied subroutine table has an illegal or currently non-supported language type.

Subscript fault

A subscript has an illegal value. This error may be caused by an index reference whose value is less than 1.

Too many subroutines

The number of active called MS-COBOL subroutines has exceeded the maximum of 20.

Too many files OPENed

The number of open data files exceeds the allowable maximum number.

Unexpected error

This is an internal error.

Appendix G
Loading the Indexed File Handler

Warning

The MS-DOS ASSIGN command and Microsoft ISAM must not be used together. If ASSIGN has been used before ISAM was loaded, the system should be turned off and rebooted before loading ISAM. If ASSIGN is to be used after ISAM is loaded, the system should also be rebooted. Failure to "clear out" ISAM or ASSIGN will result in failures of the ASSIGN or ISAM functions or will cause the system to crash.

These problems occur whenever ISAM or ASSIGN have been loaded. Users of Microsoft COBOL and Microsoft ISAM applications should receive this warning.

This implementation of Microsoft COBOL uses the Microsoft Multi-Key ISAM Facility (ISAM) to perform its Indexed file input and output. This is the same ISAM facility used by other Microsoft languages and application products. This ensures compatibility with other Microsoft products and simplifies updating to new versions of ISAM. This will be especially important when networking versions of ISAM, with file and record locking, become available.

Microsoft ISAM belongs to a special class of programs known as "terminate and stay resident" programs. Once executed, ISAM remains in memory until it is removed, or the system is shut off, and it is always available to COBOL applications.

G.1 Loading ISAM

Microsoft ISAM consists of a separate program which must be loaded before any MS-COBOL applications using Indexed files are run. To load ISAM, place a disk containing ISAM.EXE into your default drive and enter a command of the form

```
ISAM [/switches]
```

Switches used by ISAM are

/S:*buffer size in bytes*	causes the specified amount of memory to be allocated for use as Indexed file buffers. Larger buffers generally result in faster file operations. If /S is not specified, 10,000 bytes will be allocated for buffers.
/F	frees memory previously occupied by ISAM, and "unloads" ISAM.

When this command is executed, ISAM is loaded and becomes available to applications, unless the /F switch is specified, in which case a previously loaded copy of ISAM is removed from the system, and is no longer available to applications.

G.2 Using ISAM With Batch Files

The command to load Microsoft ISAM may be made part of a batch file to ensure that ISAM will be available to any application that needs it. If the command is made part of AUTOEXEC.BAT, ISAM will be loaded at system startup, and will always be present. Alternatively, ISAM loading may be done as part of a batch file used to start a COBOL application. Note that ISAM occupies memory once it is loaded (approximately 35K if the /S switch is not specified), and can be removed only by rerunning ISAM with the /F switch, or rebooting MS-DOS.

G.3 Error Handling

If an MS-COBOL program is run when ISAM has not previously been loaded, all Indexed file operations will be unsuccessful, and normal MS-COBOL I-O error handling takes place. If INVALID KEY or AT END clauses are present, the imperative statements following these clauses will be executed. If these clauses are not present, any associated DECLARATIVES procedures would be executed.

If a FILE-STATUS item has been defined, Indexed file operations will return a status value of 95. If none of these error-handling mechanisms are invoked, Indexed file operations will cause a runtime error, and the message, "Indexed file system not available," will be displayed.

Index

Microsoft® COBOL

Reference Manual

Microsoft Corporation

If you have comments about the software or these manuals, please complete the Software Problem Report at the back of this manual and return it to Microsoft Corporation.

Document Number 8301b-200-02
Part Number 011-014-021

Contents

7 PROCEDURE DIVISION 171

Tables

Acknowledgment

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

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

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

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

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems, copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part in the COBOL specification, in programming manuals or similar publications."

from the ANSI
COBOL STANDARD
(X3.23-1974)

Chapter 1
Introduction

Microsoft® COBOL extends the power of the ANSI 1974 COBOL Standard with advanced verbs, a file-sharing (multi-user) construct for Relative and Indexed file processing, interactive and trace-style debugging, and formatted screen-handling with the ACCEPT and DISPLAY statements. These features assure you that Microsoft COBOL will perform to specification and will also provide you with the tools you need to create better business programs.

How to Use This Manual

Chapter 1 introduces you to Microsoft COBOL.

Chapters 2 and 3 discuss the MS™-COBOL language, including language conventions and concepts, definitions of terms, coding rules, and the kinds of statements recognized by the MS-COBOL Compiler.

Chapters 4 through 7 discuss the four divisions of an MS-COBOL program, including any statements or clauses normally placed in those divisions. The first page of each chapter gives the general format for the appropriate division, with the arrangement that would normally appear in an MS-COBOL source program. Individual portions of the general format are then discussed in alphabetical order.

Chapters 8 through 17 present advanced MS-COBOL topics, including interprogram communication, table handling, file organizations, the SORT/MERGE facility, declaratives, the USE statement, segmentation, the COPY statement, and file and record locking.

The appendices provide a table of permissible MOVE operands, a discussion of nested IF statements, a list of COBOL reserved words, and a list of ASCII characters.

1.1 Syntax Notation

Whenever the format for a language element appears in this manual, the following conventions will apply:

CAPS	All words shown in CAPS are Microsoft COBOL reserved words. Reserved words that are not underlined are optional. If included, they are used solely for improving the readability of the program. Microsoft COBOL reserved words may be entered in upper-case or lower-case.
CAPS (underlined)	All underlined reserved words are key words, and are required unless the portion of the format containing them is itself optional. A missing key word or incorrect spelling of a key word results in an error.
lower-case	Words printed in lower-case letters represent terms for which the user must substitute a valid entry.
lower-case (with suffix)	When more than one occurrence of a term is included in a format, a digit or letter may be used as a suffix. The suffix is for clarification only, and does not change the meaning of the term (e.g., data-name-1, data-name-2).
Relational operating signs (<, >, =)	The characters less than (<), greater than (>), and equals (=), although not underlined, are required when they appear in a general format.
[ ]	Any part of a statement or data description entry that is enclosed in brackets is optional.
( )	Optional elements may be indicated by parentheses instead of brackets, if no ambiguity results.

{ }	When braces enclose parts of a statement, one (and only one) of the options must be used. In cases where ambiguity might arise, braces may also delimit the portion of a statement that may be repeated and will be followed by an ellipsis.
|	Alternate options may also be separated by a vertical line (e.g., AREA | AREAS is equivalent to enclosure within braces).
. . .	The ellipsis (. . .) indicates that the immediately preceding unit may occur once or any number of times in succession. A unit is either a single lower-case word or a group of one or more words enclosed in brackets or braces. If repetition occurs, the entire unit must be repeated.
Special characters	Punctuation and special characters are required where shown in a general format. Additional punctuation can be inserted, according to the rules for punctuation specified in Section 2.3, "Punctuation."
Terminal periods	Terminal periods are shown in formats where they are required as separators. Semicolons and commas are sometimes shown as separators, but they are optional.
Blank following separators	To be considered separators, all commas, semicolons, and periods must be followed by a space or blank.

If necessary, the text accompanying the general format will contain comments, restrictions, or clarification on the use of the format.

Any clauses (e.g., BLOCK clause) or statements (e.g., PERFORM statement) mentioned in a format will be described elsewhere in the text.

1.2 Microsoft COBOL Conformance With the ANSI Standard

The following list compares the modules supported by Microsoft COBOL with the ANSI Standard, Level 2, to give you a sense of the full capabilities of the Microsoft COBOL Compiler.

Microsoft COBOL Module	Level of Implementation
Nucleus Table Handling Sequential I-O Relative I-O Indexed I-O Segmentation Library	All Level 2 features are implemented.
Interprogram-Communication	All Level 2 features are implemented. The CHAIN verb is an extension to the full language standard.
Sort/Merge	Full Level 2 implementation with an extension, SORT STATUS, for sort file status reporting.
Debug Report-Writer Communications	Not currently implemented. However, Microsoft COBOL does include the trace-style debug extensions to ANSI 74 Standard COBOL and an interactive debug facility.

1.3 Microsoft COBOL Extensions

Microsoft COBOL includes the following extensions to ANSI 74 Standard COBOL.

1. COMP-0, COMP-3, and COMP-4 data formats are available. COMP-3 format packs numeric data two digits per byte. COMP-0 and COMP-4 are two and four byte binary integers, respectively. All three data types can be used to reduce DATA DIVISION memory requirements, to reduce data file storage requirements, and to increase the execution speed of certain operations.

2. Microsoft COBOL is capable of defining screen attributes and having these attributes and other screen definitions displayed on your terminal's screen in an interactive mode.

   This capability comes from Microsoft extensions to the DATA DIVISION (SCREEN SECTION) and the PROCEDURE DIVISION (ACCEPT and DISPLAY statements). (See Section 6.4.4, "SCREEN SECTION," and Formats 1, 3, and 4 of the ACCEPT and Format 3 of the DISPLAY statement in Chapter 7, "PROCEDURE DIVISION.")

3. Lower-case characters are treated as if they were upper-case, unless made part of a non-numeric (quoted) literal.

4. The dynamic debugging statements, READY TRACE, RESET TRACE, and EXHIBIT, allow the display of procedure names or data-items during program execution.

5. The SELECT clause of the ENVIRONMENT DIVISION supports a split key option with both the RECORD KEY and the ALTERNATE RECORD KEY clauses. See Section 11.2, "Syntax Considerations for Indexed File I-O."

6. A CHAIN statement and CHAINING phrase extend the scope of interprogram communication and allow a program to be loaded into memory and executed.

7. A multi-tasking file-sharing construct exists to support file processing in multi-user/multi-tasking systems. The new syntax applies in the OPEN, READ, START, and UNLOCK statements, and the SELECT clause. Note that the UNLOCK statement is an extension to the full language standard.
8. Sort file status reporting has been implemented through the SORT STATUS clause in the FILE-CONTROL entry for a sort file.

1.4 Learning More About COBOL

If you are new to COBOL programming, you will probably want to learn more about the language before using this manual. The following texts are all COBOL tutorials, written for the novice programmer:

Abel, Peter. *COBOL Programming: A Structured Approach.* Reston, Virginia: Reston Publishing Co., 1980.

McCracken, Daniel D. *A Simplified Guide to Structured COBOL Programming.* New York, New York: John Wiley and Sons, Inc., 1976.

Parkin, Andrew. *COBOL for Students.* London, England: Edward Arnold, Ltd., 1978.

Seidel, Ken. *Microsoft COBOL.* Beaverton, Oregon: Dilithium Press, 1983.

Welburn, Tyler. *Structured COBOL — Fundamentals and Style.* Palo Alto, California: Mayfield Publishing Co., 1981.

Chapter 2
Language Elements

This chapter defines the terms that are used throughout this manual to refer to parts of a Microsoft COBOL program. It also gives the rules for coding a Microsoft COBOL source program and outlines the naming conventions recognized by Microsoft COBOL.

2.1 Source Coding Rules

Source programs may be written on standard coding sheets or on a terminal. The rules given below apply to both methods; though the actual column numbers may differ slightly for a particular terminal, the relative positions remain the same.

The Microsoft COBOL Compiler interprets source code in four areas of each source line: the sequence number area (defined by columns 1-6), the indicator area (column 7), Area A (columns 8-11), and Area B (columns 12-72). The compiler ignores the contents of the sequence number area, and the contents of columns beyond column 72 (see Figure 2.1).

Area A is reserved for the following elements of the source code: division, section, and paragraph headers, file description entry indicators (FD and SD), and level-numbers 01 and 77. Comment line characters are also accepted in Area A, provided that column 7 contains an asterisk (*) or a slash (/).

Area B is reserved for the filenames associated with the FD or SD indicators and the SELECT clause, level-numbers 66, 88, and 02-49, and the descriptive clauses and/or data descriptions that make up specific paragraph entries. Continuation lines are also accepted in Area B, provided that column 7 contains a hyphen (-).

The following diagram illustrates Area A, Area B, the indicator area, the ignored area, and the optional sequence number area in a COBOL source line.

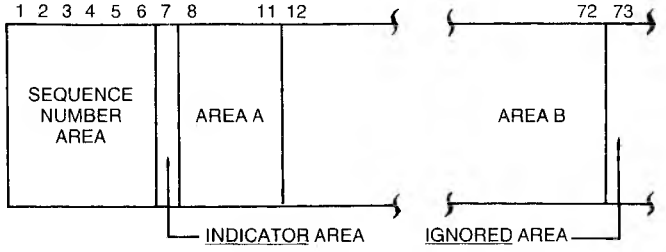


Figure 2.1. Format of MS-COBOL Source Line

The following rules apply to coding a Microsoft COBOL program:

1. Each line of code may have a six-digit line number in columns 1 through 6. These line numbers must be in ascending order. Blanks are also permitted in columns 1 through 6.
2. If an asterisk (*) is placed in column 7 of the line, the line will be treated as a comment. It will be shown on the source listing but will otherwise be ignored. If a slash (/) appears in column 7, the line will be treated as a comment, and the source listing will begin a new page before printing the line.
3. If the character "D" is placed in column 7 of the line, the line will be treated as a comment unless the WITH DEBUGGING MODE clause of the SOURCE-COMPUTER paragraph is used. See Section 5.2.2 for information concerning the SOURCE-COMPUTER paragraph.
4. If a hyphen (-) is placed in column 7, the line is treated as a continuation of the previous line. Except when non-numeric literals are continued, all trailing spaces on the preceding line and all leading spaces on the continuation line are ignored. (See Section 2.6.2, "Non-Numeric Literals," for special rules on continuing non-numeric literals.) Area A (columns 8 through 11) of the continuation line must be blank.
5. Reserved words for division, section, and paragraph headers must begin in Area A. Definitions of procedure-names must also begin in Area A, as must

level-numbers 01 and 77 and level-indicators FD and SD. Other level numbers must begin in Area B (columns 12 through 72).

6. All other program elements should be confined to Area B. Rules of statement punctuation must be observed.
7. Information entered beyond column 72 is ignored by the compiler.
8. Tab characters in a line are expanded as specified in the *Microsoft COBOL Compiler User's Guide*. Care should be taken that tab characters do not cause illegal placement of program elements.

2.2 Character Set

The Microsoft COBOL language character set consists of the following characters:

Letters A through Z, a through z
Blank or space
Digits 0 through 9
Special characters:

+	Plus sign
–	Minus sign
*	Asterisk
=	Equal sign
>	Relational sign (greater than)
<	Relational sign (less than)
$	Dollar sign
,	Comma
;	Semicolon
.	Period or decimal point
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
'	Apostrophe (alternate of quotation mark)
/	Slash

For non-numeric (quoted) literals and comments, the Microsoft COBOL character set is expanded to include the computer's entire character set.

2.3 Punctuation

The following characters are used for punctuation:

(	Left parenthesis
)	Right parenthesis
,	Comma
.	Period
;	Semicolon

The following general rules of punctuation apply in writing source programs:

1. When used as punctuation, a period, semicolon, or comma should not be preceded by a space, but must be followed by a space.
2. At least one space must appear between two successive words.
3. Relational characters should always be preceded by a space and followed by another space.
4. When the plus or minus characters, period, or comma are used in the PICTURE clause, they are governed solely by rules for numeric-edited items (see Section 6.5.21, "PICTURE Clause," for further discussion).
5. A comma may be used as a separator between successive operands of a statement, or between two subscripts. It must be followed by a space (e.g., 10, 20).
6. A semicolon or comma may be used to separate a series of statements or clauses. The punctuation must be followed by a space (e.g., SUBTRACT A FROM X; MOVE X TO Y).

2.4 Reserved Words

Reserved words are words with specific meanings within the COBOL language or within Microsoft COBOL. They appear in upper-case letters in general formats. They may contain the letters A through Z and a through z, the digits 0 through 9, or the hyphen (-). The maximum length is 30 characters. Many are verbs (e.g., ADD, SUBTRACT, MOVE) or descriptive phrases (e.g., PICTURE, VALUE IS). Reserved words may not be used for programmer-assigned names.

See Appendix C, "Reserved Words," for a complete list of COBOL reserved words.

2.5 Names

Any word that is not a Microsoft COBOL reserved word can be used as a programmer-assigned name, as long as it meets the naming conventions listed in the following section.

2.5.1 Naming Conventions

Names may be up to 30 characters long and must contain only the letters A through Z and a through z, the digits 0 through 9, or the hyphen (-). In addition:

1. All names except procedure-names must contain at least one letter or hyphen. Procedure-names may consist entirely of digits.
2. A name may not begin or end with a hyphen. However, a name may contain more than one hyphen, and consecutive hyphens are permitted.
3. A name is ended by a space or by appropriate punctuation.
4. If a programmer-supplied name is not unique, it must be used with qualifiers. Qualifiers are described in Section 2.5.2, "Qualification of Names."

2.5.1.1 Data-Names

A data-name is a word assigned by the user to identify a data-item referenced in a program.

Data-names are defined in the DATA DIVISION of the program. A data-name always refers to a region that contains data, rather than to a particular value, because an item often assumes a number of different values during the course of a program.

If some of the characters in a record are not referenced in the processing steps of a program, a data-name need not be assigned. Instead, the word FILLER is used to set aside the appropriate amount of space.

A data-name must begin with an alphabetic character. A data-name or the key word FILLER must be the first word following the level number in each data-name entry, as shown in the following general format:

```
level-number  { data-name-1 }
              { FILLER      }
```

See Chapter 3, "Structure of a COBOL Program," for discussion of level numbers. See also Chapter 6, "DATA DIVISION," for more information on assigning data-names.

2.5.1.2 File-Names

A file is a collection of data records, such as a printed listing or a region of a disk, containing individual records of a similar class or application. A file-name is defined by an FD entry in the FILE SECTION of the DATA DIVISION. The format is:

```
FD file-name
```

Rules for composition of the file-name are identical to those for data-names. References to file-names appear in the ENVIRONMENT DIVISION and in CLOSE, OPEN, and READ statements.

2.5.1.3 Condition-Names

A condition-name is a name assigned to a specific value, set, or range of values within the complete set of values that a data-item may assume. Condition-names are defined in level 88 entries within the DATA DIVISION. For example, a level 03 item "CLASS-NO" might be followed by the following subordinate level 88 entries:

```
88 VALID-NO      VALUE IS '1'.
88 INVALID-NO    VALUE IS '2'.
```

An IF statement could then reference either the literal value '1' or '2', or the condition-names VALID-NO or INVALID-NO.

Rules for forming condition-names are the same as those for data-names (see Section 2.5.1.1, "Data-Names"). Condition-names and their uses are explained more fully in Chapter 6, "DATA DIVISION," and Chapter 7, "PROCEDURE DIVISION."

2.5.1.4 Mnemonic-Names

A mnemonic-name such as PRINTER assigns a user-defined word to an implementation-specific name. The rules for naming are the same as those for data-names (see Section 2.5.1.1, "Data-Names"). Mnemonic-names are assigned in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION and are referenced by the ACCEPT and DISPLAY statements and the WRITE statement for Sequential files.

2.5.1.5 Procedure-Names

Procedure-names are assigned to paragraphs or sections that are executed with the PERFORM or GO TO statements or by falling through from another part of the program. Procedure-names are declared in the PROCEDURE DIVISION. They must conform to the rules for data-names (see Section 2.5.1.1, "Data-Names"), except that a procedure-name may consist entirely of digits.

2.5.1.6 Index-Names and Index-Data-Items

Index-names and index-data-items are used for table handling by the indexing method. An index-name is declared implicitly by its appearance in the "INDEXED BY index-name" appendage to an OCCURS clause. Index-data-items are defined by the USAGE IS INDEX phrase. See Chapter 9, "Table Handling by the Indexing Method," for further discussion.

2.5.2 Qualification of Names

When a data-name, condition-name, or paragraph-name (see Chapter 3, "Structure of a COBOL Program," for a description of paragraphs) is not unique, a specific instance of the name may be referenced by using qualifiers.

For example, if there were two or more items named YEAR, the qualified reference

```
YEAR OF HIRE
```

might differentiate between YEAR fields in HIRE and TERMINATION.

Qualifiers are data-names or condition-names preceded by "OF" or "IN". The qualifiers must designate broader-level groups that contain all the names in the reference. For example, HIRE must be a data-name, a section-name, or a library-name to which YEAR is subordinate or in some way related. (See Chapter 3, "Structure of a COBOL Program," for discussion of hierarchy.)

Paragraph-names may be qualified by a section-name. Text-names in COPY statements may be qualified by a library-name.

The maximum number of qualifiers allowed is: one for a paragraph-name or text-name; five for other qualified items. File-names and mnemonic-names must be unique.

A qualified name may be written in the ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. A reference to a paragraph-name that is defined more than once need not be qualified when the reference is made within the same section.

2.6 Literals

A literal is a constant. It is not assigned a data-name in a program, but is referred to only by its value, which does not change. Literals can be numeric, non-numeric (quoted), or figurative.

2.6.1 Numeric Literals

A numeric literal must contain at least one and not more than 18 digits. A numeric literal may consist of the characters 0 through 9, optionally preceded by a sign, and the decimal point. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the left-most character in the numeric literal. If a numeric literal is unsigned, it is assumed to be positive.

A decimal point may appear anywhere within the numeric literal except as the right-most character. If a numeric literal does not contain a decimal point, it is considered to be an integer.

The following are examples of numeric literals:

```
72      +1011      3.14159      -6      -.333      0.5
```

European notation (period and comma interchanged) can be specified by including the DECIMAL-POINT IS COMMA entry in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. In European notation, for example, the numeric literal pi would be written as 3,14159.

2.6.2 Non-Numeric Literals

A non-numeric literal is also called a "quoted" literal. It is delimited by quotation marks or apostrophes, and may consist of any combination of characters in the ASCII set. Generally, if the literal is delimited by apostrophes, quotation marks may be used within the literal, and vice versa. However, the delimiter character can be used as a character within the literal if two such characters are consecutive. In such a case, the two characters are considered as one representation of the delimiter within the literal. For instance,

```
"THE DATA-NAME ""VALID-NO"" IS ACCEPTED HERE"
```

would be interpreted as

```
THE DATA-NAME "VALID-NO" IS ACCEPTED HERE
```

All spaces enclosed by the delimiters are included as part of the literal and are counted when the length is checked. Delimiters are not included in the length, which must be in the range 1 through 120.

The following are examples of non-numeric literals:

```
"ILLEGAL CONTROL CARD"

'CHARACTER-STRING'

"DO'S & DON'T'S"
```

Non-numeric literals may be continued from one line to the next. The following rules apply to the continuation line:

1. Column 7 of the continuation line must contain a hyphen.
2. Area A of the continuation line must be blank.
3. A delimiter must be entered in Area B, followed by the continuation of the literal.
4. All spaces at the end of the previous line and any spaces from the delimiter to the end of the continuation line are considered to be part of the literal.

2.6.3 Figurative Constants

A figurative constant is a special type of literal. It represents a value or character to which a reserved data-name has been assigned by Microsoft COBOL. When the program is compiled, that value or character will be provided as needed. For example, the figurative constant SPACE clears its entire field to blanks; LOW-VALUE enters the computer's lowest value. A figurative constant is not bounded by quotation marks.

In Microsoft COBOL, the reserved words that follow are figurative constants. The plural forms of the words are accepted by the compiler but are equivalent to the singular forms.

ZERO	may be used in many places in a program as a numeric literal. It may also be used in alphanumeric fields.
SPACE	represents the blank character.
LOW-VALUE	represents the computer's lowest value.
HIGH-VALUE	represents the computer's highest value.
QUOTE	represents the double quotation mark (*"*).
ALL literal	indicates one or more instances of the literal, which may be a multiple-character non-numeric literal or a figurative constant. If the literal is a figurative constant, ALL is not necessary but is usually included for readability.

A figurative constant may be used anywhere a literal is called for in a general format, except where the literal is numeric only. In this case, the only figurative constant that can be used is ZERO.

2.7 Data Types (Categories)

Every elementary data-item except an index-data-item belongs to one of the following categories: alphabetic, alphanumeric, alphanumeric-edited, numeric, or numeric-edited.

Group data-items are treated as alphanumeric regardless of the data type(s) of their elementary items.

2.8 Statements

Statements specify actions to be taken by the compiler. They usually consist of a verb, such as ACCEPT or MOVE, followed by operands that are data-names or literals. The compiler recognizes three kinds of statements: imperative, conditional, and compiler directing.

2.8.1 Imperative Statements

An imperative statement specifies an unconditional action to be taken by the program. Imperative statements appear only in the PROCEDURE DIVISION of the program. The verbs that can be used in imperative statements are:

ACCEPT	EXIT	SEARCH
ADD*	GO	SET
CALL*	INSPECT	SORT
CANCEL	MOVE	START*
CHAIN	MULTIPLY	STOP
CLOSE	OPEN	STRING
COMPUTE*	PERFORM	SUBTRACT*
DELETE*	READ*	UNLOCK
DISPLAY	READY	UNSTRING
DIVIDE*	RESET	WRITE*
EXHIBIT	REWRITE*	

* If the statement contains an ON SIZE ERROR, INVALID KEY, AT END, or ON OVERFLOW clause, it becomes a conditional, rather than an imperative statement.

See Chapter 7, "PROCEDURE DIVISION," for discussion of individual statements.

2.8.2 Conditional Statements

Conditional statements test for conditions and provide alternate paths of program execution. Conditional statements occur only in the PROCEDURE DIVISION of a program. The following verbs can be used for conditional statements:

ADD*	MULTIPLY*	START*
CALL*	READ*	STRING
COMPUTE*	RETURN	SUBTRACT*
DELETE*	REWRITE*	UNSTRING
DIVIDE*	SEARCH	WRITE*
IF		

* These verbs form conditional statements when used with ON SIZE ERROR, INVALID KEY, AT END, or ON OVERFLOW phrases.

2.8.3 Compiler Directing Statements

A compiler directing statement is a command to the compiler itself, rather than a functional part of the program. Compiler directing statements can be used anywhere in the ENVIRONMENT, DATA, or PROCEDURE DIVISIONS. The two verbs used as compiler directing statements are COPY and USE. COPY reads source lines from another file and inserts them in the original program. USE specifies procedures to be executed when input-output errors occur. See Chapter 14, "DECLARATIVES Region and USE Statement," and Chapter 16, "COPY Statement," for a discussion of these two statements.

2.9 Arithmetic Statements

There are five arithmetic statements: ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE. Any arithmetic statement may be either imperative or conditional.

All Microsoft COBOL arithmetic statements may have multiple destinations. For example,

```
ADD 1 TO RECORD-COUNT, LINE-COUNT, SUBTOTAL.
```

will add one to the contents of each of the data-items in the list following the connector, TO.

Arithmetic statements are subject to the restriction of composite operands, and may also be modified by five optional phrases.

2.9.1 Composite of Operands

The composite of operands for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements is a hypothetical data-item whose size is the result of taking the sum, for all the applicable operands of the statement, of the largest number of digits to the left of the decimal point, and the largest number of digits to the right of the decimal point. An implied decimal point is assumed, if none is specified.

For the ADD and SUBTRACT operations, the composite of operands is determined by superimposing all operands in a given statement (except those following the GIVING option). If the CORRESPONDING option is used, the composite of operands is determined separately for each corresponding pair of data-items.

For the MULTIPLY operation, the composite of operands is determined by superimposing all receiving data-items.

For the DIVIDE operation, the composite of operands is also determined by superimposing all receiving data-items, but the data-item referred to by the REMAINDER option is not included.

Neither the composite of operands nor the receiving fields defined in the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements may exceed eighteen (18) decimal digits. This restriction does not apply to the COMPUTE statement. (See the example below.)

Example:

```
ADD A, B, C, D, E GIVING TOTAL.
```

where the data are defined as follows:

```
05  A          PIC 9(5)V9(3).
05  B          PIC 9(7)V9(2).
05  C          PIC 9(2)V9(5).
05  D          PIC 9(4)V9(3).
05  E          PIC 9(11)V9(4).
05  TOTAL      PIC 9(12)V9(5).
```

The composite of operands in this example would have a maximum left side number of 11, and a right side number of 5, effectively,

```
COMPOSITE-ITEM      PIC 9(11)V9(5).
```

which has 16 digits. However, if the B operand was defined as PIC 9V9(8), the right side number would become 8, and the composite of operands would contain 19 digits, which is not permissible. Note that the size of the GIVING data-item, TOTAL, is not included in the calculation.

2.9.2 Optional Phrases

Five optional phrases are available with arithmetic statements. They are: ON SIZE ERROR, ROUNDED, GIVING, CORRESPONDING, and REMAINDER. See Chapter 7, "PROCEDURE DIVISION," for a detailed discussion of individual arithmetic statements and optional phrases.

When an arithmetic statement includes an ON SIZE ERROR specification, the entire statement is termed conditional, because the size-error condition is data-dependent. For example, the following arithmetic statement is conditional:

```
ADD 1 TO RECORD-COUNT
    ON SIZE ERROR
        MOVE ZERO TO RECORD-COUNT
        DISPLAY "LIMIT 99 EXCEEDED".
```

If a size error occurs (in this case, it is assumed that RECORD-COUNT has PICTURE 99, and therefore cannot hold a value of 100), subsequent statements (i.e., MOVE and DISPLAY) are executed.

All data-names used in arithmetic statements must be elementary numeric data-items that are defined in the DATA DIVISION of the program, except that operands of the GIVING option may be numeric-edited items. Index-names and index-data-items are not allowed in these arithmetic statements. (See Chapter 6, "DATA DIVISION," for a definition of elementary items.)

Decimal point alignment is supplied automatically throughout arithmetic computations. Intermediate result fields are generated for the evaluation of arithmetic expressions. These intermediate fields assure the accuracy of the result, except where high-order truncation is necessary.

2.10 Arithmetic Expressions

An arithmetic expression is a combination of numeric literals, data-names, arithmetic operators, and parentheses. In general, the data-names in an arithmetic expression must designate numeric data. Consecutive data-names (or literals) must be separated by an arithmetic operator, and there must be one or more blanks on either side of the operator. The operators are:

+	for addition
–	for subtraction
*	for multiplication
/	for division
**	for exponentiation to an integer power

When more than one operation is to be executed using a given variable or term, the order in which the operations are performed is:

1. the unary operators plus and minus (involving one variable only)
2. exponentiation
3. multiplication and division
4. addition and subtraction

Parentheses may be used to change the standard order of evaluation. Expressions within parentheses are evaluated first, and parentheses may be nested to any level. When parentheses are used in an expression, the following punctuation rules should be observed:

1. A left parenthesis is preceded by one or more spaces.
2. A right parenthesis is followed by one or more spaces.

The following examples illustrate the evaluation process.

Example 1: The expression

```
A + B / (C - D * E)
```

is evaluated in the following sequence:

1. Compute the product D times E, considered as intermediate result R1.
2. Compute intermediate result R2 as the difference C – R1.
3. Divide B by R2, providing intermediate result R3.
4. The final result is computed by addition of A to R3.

Example 2: The same expression, without parentheses,

```
A + B / C - D * E
```

is evaluated as follows:

1. R1 = B / C
2. R2 = D * E
3. R3 = A + R1
4. The final result is R3 − R2.

Example 3: The expression

```
A - B - C
```

is evaluated as:

```
(A - B) - C
```

Chapter 3
Structure of a COBOL Program

Every COBOL source program is divided into four divisions:

1. IDENTIFICATION DIVISION, which names and documents the program
2. ENVIRONMENT DIVISION, which indicates the computer equipment and features to be used in the program
3. DATA DIVISION, which defines the names and characteristics of data to be processed
4. PROCEDURE DIVISION, which consists of statements that direct the processing of data and program execution

Each division must begin with a division header, and divisions must appear in the program in the order shown above.

Each program division consists of a particular arrangement of "grammatical" parts. In hierarchical order, these parts fit together as follows, with "division" as the highest level part:

Division
Region
Section
Paragraph
Sentence | Entry
Statement | Clause
Phrase | Option

When one of the levels shown in the preceding list contains multiple terms separated by a vertical bar (e.g., Sentence | Entry), the term that is used depends on which part of the program it occurs in. For example, the progression in the ENVIRONMENT DIVISION is: Division, Section, Paragraph; while the progression in the DATA DIVISION is: Division, Section, Entry, Clause. See Chapters 4 through 7 for general formats for each division.

3.1 Terms

The following list defines the preceding terms and other Microsoft COBOL terms associated with them. The list starts with the lowest level in the hierarchy. For a more comprehensive glossary of COBOL terms, see the 1974 ANSI Standards document, *American National Standard Programming Language COBOL*, ANSI X3.23-1974, ISO 1989-1978, corrected edition July 1978.

1. Phrase

   A group of words that performs part of a procedural statement or clause. For example, the WRITE statement contains an optional INVALID KEY phrase which specifies a procedure that will be performed if an INVALID KEY condition exists.

2. Option

   Because most phrases are optional (as denoted by brackets in the general format), they are often referred to as options.

3. Statement

   An action that is to be performed. It includes a verb, one or more operands (data-names or literals) that are to be acted on, and any necessary phrases. The three kinds of statements — imperative, conditional, and compiler directing — are defined in Chapter 2, "Language Elements."

   All statements except COPY appear only in the PROCEDURE DIVISION. The COPY statement may appear anywhere except the IDENTIFICATION DIVISION.

4. Clause

   A group of words that specify an attribute, or characteristic, of an entry in the DATA DIVISION. An entry may have multiple clauses.

5. Sentence

   A group of one or more statements. The last statement in the sentence is followed by a period (.) and a space.

Like statements, sentences appear only in the PROCEDURE DIVISION.

6. Entry

   Entry is often used as a general term for anything that is "entered" in a particular place in a program. However, it does have a specific meaning in COBOL: an entry is a descriptive set of clauses, ending with a period. Unless stated otherwise, the specific meaning will be used in this manual. Entries may occur in the IDENTIFICATION, ENVIRONMENT, and DATA DIVISIONs.

7. Paragraph

   A group of related sentences (in the PROCEDURE DIVISION) or entries (in the IDENTIFICATION, ENVIRONMENT, or DATA DIVISIONs). A paragraph always starts with a paragraph-name or paragraph header.

   In some cases, a group of entries will constitute a section, rather than a paragraph. This happens, for example, in the FILE SECTION of the DATA DIVISION.

8. Section

   A set of related paragraphs or entries. A section always starts with a section header.

9. Region

   A set of PROCEDURE DIVISION sections. The only region in Microsoft COBOL is the DECLARATIVES Region in the PROCEDURE DIVISION. It starts with the DECLARATIVES header and ends with END DECLARATIVES.

10. Division

    One of the four major functional parts of a COBOL program. A division always starts with a division header.

3.2 Level Numbers and Data-Items

In the DATA DIVISION of a Microsoft COBOL program, all entries except the FD and SD entries (for file and sort file descriptions) are names and descriptions of data-items used in the program. These data-items can be group items, elementary items, or conditions. A group item has subordinate items within it; an elementary item does not. Level numbers are a form of outline that shows how the data-items are related to each other. Group items can be at any level from 01 to 48. An item's status as a group or elementary item is determined solely by whether another item is subordinate to it. An item is a group item if a higher numbered level exists within the program before a lower or equal level is found. Subordinate items may themselves be either group items or elementary items.

Subordinate levels need not be consecutive, and numbers can be skipped to allow for later insertions.

The following example shows two levels of subordination:

```
01  TIME-CARD.
    02  NAME.
        03 LAST-NAME     PICTURE X(18).
        03 FIRST-INIT    PICTURE X.
        03 MIDDLE-INIT   PICTURE X.
    02  EMPLOYEE-NUM  PICTURE 99999.
    02  WEEKS-END-DATE.
        05  MONTH              PIC 99.
        05  DAY-NUMBER         PIC 99.
        05  YEAR               PIC 99.
    02  HOURS-WORKED  PICTURE 99V9.
```

In this example, all level 03 items are subordinate to the group item NAME (02), and all level 05 items are subordinate to the group item WEEKS-END-DATE (02). The level 02 items are, in turn, subordinate to level 01. Therefore, level 01 is a group item, all items in levels 03 and 05 are elementary items, level 02 items NAME and WEEKS-END-DATE are group items, and level 02 items EMPLOYEE-NUM and HOURS-WORKED are elementary items.

The level numbers assigned to various types of data-items are:

01-49	group and elementary items
66	items that are objects of the RENAMES clause
77	items that are noncontiguous, are not subordinates of other items, and do not have subordinates
88	condition-names and conditions

The following rules apply to level numbers and data-items:

1. When a PROCEDURE DIVISION statement refers to a group item, the reference applies to the area reserved for the entire group, including all subordinate items.
2. In the FILE SECTION, consecutive 01 level numbers subordinate to any given file represent implicit redefinitions of the same area. In the WORKING-STORAGE SECTION, however, each 01 level number defines its own memory area, unless the REDEFINES clause is used.
3. When data-items are coded, level numbers 01 and 77 are placed in Area A. The level numbers 66, 88, and 02-49, the data-names for these items, and data-names for all subordinate items, begin in Area B.
4. All elementary items must be described with a PICTURE or USAGE IS INDEX clause.

See Chapter 6, "DATA DIVISION," for descriptions of the various types of data-items.

3.3 Compiler Directing Statements

Two nonexecutable statements, USE and COPY, issue direct instructions to the compiler.

The USE statement defines procedures that are to be executed under certain file I-O error conditions. The USE statement is confined to use in the DECLARATIVES Region of a PROCEDURE DIVISION.

The COPY statement logically imbeds the contents of a file into a source program, and may be used anywhere in the ENVIRONMENT, DATA, or PROCEDURE DIVISIONs.

For more information on the USE and COPY statements, see Chapter 14, "DECLARATIVES Region and USE Statement," and Chapter 16, "COPY Statement," respectively.

Chapter 4
IDENTIFICATION DIVISION

Every Microsoft COBOL program begins with the IDENTIFICATION DIVISION, which names the program and its author, and describes other characteristics of the program.

4.1 IDENTIFICATION DIVISION Header and General Format

Purpose

To state the program name, author, and other characteristics, and to indicate the beginning of a program.

Format

The IDENTIFICATION DIVISION is divided into a header and accompanying paragraphs.

The general format is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
[ AUTHOR. [ comment-entry ] ... ]
[ INSTALLATION. [ comment-entry ] ... ]
[ DATE-WRITTEN. [ comment-entry ] ... ]
[ DATE-COMPILED. [ comment-entry ] ... ]
[ SECURITY. [ comment-entry ] ... ]
```

Remarks

The IDENTIFICATION DIVISION header must be the first line of any MS-COBOL program.

Only the division header and the PROGRAM-ID paragraph are required. The other paragraphs are included only for documentation.

A period (.) is required at the end of the division header and at the end of each paragraph header.

Example

The following example shows a typical IDENTIFICATION DIVISION, with the paragraphs in the order in which they are usually entered. Comment entries may contain any character allowable on the computer. Coding of IDENTIFICATION DIVISION paragraphs must begin in Area A (columns 8-11). Coding of comment entries is restricted to Area B (columns 12-72).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INVENTORY.
AUTHOR. M A HOWELL.
INSTALLATION. SHIPPING AND RECEIVING DIV.
DATE-WRITTEN. 1-15-83.
DATE-COMPILED. 1-20-83.
SECURITY. DEPT USE ONLY.
```

The paragraphs in the IDENTIFICATION DIVISION are discussed, in alphabetical order, in the remainder of this chapter.

4.2 AUTHOR Paragraph

Purpose

The AUTHOR paragraph tells who wrote the program.

Format

The general format is:

```
[ AUTHOR. [ comment-entry ] ... ]
```

Remarks

This paragraph is optional, and is used for documentation only.

The name may not contain embedded periods (.).

Example

```
AUTHOR. M A HOWELL.
```

4.3 DATE-COMPILED Paragraph

Purpose

Tells when the program was first compiled.

Format

The general format is:

```
[ DATE-COMPILED. [ comment-entry ] ... ]
```

Remarks

This paragraph is optional. When used, the comment entry, excluding lines with comment indicators in column 7, is replaced by the current date and time.

Example

```
DATE-COMPILED. 1-20-83.
```

4.4 DATE-WRITTEN Paragraph

Purpose

Tells when the program was written.

Format

The general format is:

```
[ DATE-WRITTEN. [ comment-entry ] ... ]
```

Remarks

This paragraph is optional, and is used for documentation only.

Example

```
DATE-WRITTEN. 1-15-80.
```

4.5 INSTALLATION Paragraph

Purpose

Tells how the program is used.

Format

The general format is:

```
[ INSTALLATION. [ comment-entry ] ... ]
```

Remarks

This paragraph is optional, and is used for documentation only.

Example

```
INSTALLATION. SHIPPING AND RECEIVING DIV.
```

4.6 PROGRAM-ID Paragraph

Purpose

Tells the name of the object program created by the compiler.

Format

The general format is:

```
PROGRAM-ID. program-name.
```

where program-name can be any alphanumeric string of characters, except that the first character must be a letter. Embedded periods (.) are not allowed. If the name contains more than one word, the words must be separated by hyphens, rather than by spaces. Only the first six characters of the program-name are retained by the compiler.

Remarks

This paragraph is required. It must be the first paragraph in the IDENTIFICATION DIVISION.

Example

```
PROGRAM-ID. INVENTORY.
```

4.7 SECURITY Paragraph

Purpose

Tells the security level of the program.

Format

The general format is:

```
[ SECURITY. [ comment-entry ] ... ]
```

Remarks

This paragraph is optional, and is used for documentation only.

Example

```
SECURITY. DEPT USE ONLY.
```

Chapter 5
ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION specifies the aspects of a Microsoft COBOL program that depend on the physical characteristics of the computer. This division is required in every program and follows the IDENTIFICATION DIVISION.

5.1 ENVIRONMENT DIVISION Header and General Format

Purpose

To specify aspects of the program that depend on the physical characteristics of the computer.

Format

The ENVIRONMENT DIVISION always begins with a division header. The division may contain two sections: an optional CONFIGURATION SECTION, and an INPUT-OUTPUT SECTION that is required unless the program has no data files. Each of these sections is further divided into paragraphs, which may, in turn, be divided into clauses.

The general format is:

```
ENVIRONMENT DIVISION.
[ CONFIGURATION SECTION.
[ SOURCE-COMPUTER. source-computer-entry ]
[ OBJECT-COMPUTER. object-computer-entry ]
[ SPECIAL-NAMES. special-names-entry ] ]
[ INPUT-OUTPUT SECTION.
[ FILE-CONTROL. { file-control entry } ... ]
[ I-O-CONTROL. input-output-control-entry ] ]
```

Remarks

The remainder of this chapter presents the division and section headers, each of the paragraphs in the ENVIRONMENT DIVISION, and the related clauses.

Example

The following example shows a typical ENVIRONMENT DIVISION, with sections and paragraphs given in their order of appearance in the program.

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.
SPECIAL-NAMES. PRINTER IS LPRINTER.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT INVENTORY-MASTER-FILE
       ASSIGN TO DISK
       FILE STATUS IS MASTER-STATUS.
    SELECT INVENTORY-REPORT-FILE
       ASSIGN TO PRINTER.
I-O-CONTROL.
    SAME RECORD AREA FOR
       INVENTORY-MASTER-FILE,
       INVENTORY-REPORT-FILE.
```

5.2 CONFIGURATION SECTION Header

Purpose

Indicates the beginning of the CONFIGURATION SECTION. The type of computer being used and any special characteristics or names are specified in the CONFIGURATION SECTION.

Format

```
CONFIGURATION SECTION.
```

Remarks

The CONFIGURATION SECTION is optional.

The header must be entered as shown above, including the period (.). The header must begin in Area A.

The CONFIGURATION SECTION may contain three paragraphs:

SOURCE-COMPUTER
OBJECT-COMPUTER
SPECIAL-NAMES

The contents of the SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs are treated as comments, except for the WITH DEBUGGING MODE clause of the SOURCE-COMPUTER paragraph. The SPECIAL-NAMES paragraph assigns user-defined names to system names such as EJECT and PRINTER, and changes default editing characters. If any of these paragraphs are included in the program, the CONFIGURATION SECTION header must be entered.

For more information on these paragraphs, see the individual descriptions which follow in this chapter.

Example

```
ENVIRONMENT DIVISION.
  .
  .
  .
CONFIGURATION SECTION.
  .
  .
  .
```

5.2.1 OBJECT-COMPUTER Paragraph

Purpose

Identifies the computer on which the program is to be executed and, optionally, the collating sequence to be used with SORT and MERGE functions on non-numeric fields.

Format

```
OBJECT-COMPUTER. computer-name

[ , MEMORY SIZE integer { WORDS      } ]
                        { CHARACTERS }
                        { MODULES    }

[ , PROGRAM COLLATING SEQUENCE IS alphabet-name ]

[ , SEGMENT-LIMIT IS segment-number ].
```

Remarks

This paragraph is primarily used for documentation. The header must be entered exactly as shown above, including the period (.). The period must be followed by at least one space.

The MEMORY SIZE and PROGRAM COLLATING SEQUENCE clauses are optional. The MEMORY SIZE clause is treated as commentary. If the PROGRAM COLLATING SEQUENCE clause is used, the collating sequence associated with "alphabet-name" is used to determine the truth of non-numeric comparisons that are:

1. explicitly specified in relation conditions
2. explicitly specified in condition-name conditions
3. implicitly specified by the presence of a CONTROL clause in a report description

If the PROGRAM COLLATING SEQUENCE clause is not specified, the native collating sequence is used (the native sequence for MS-COBOL is ASCII).

If the COLLATING SEQUENCE phrase is used in either the SORT or MERGE statement, the collating sequence specified there will override all others during execution of these statements.

The SEGMENT-LIMIT clause delimits the segment numbers that can be considered as permanent segments, and is used to provide more memory resources for the largest overlayable segment of the object program.

If the SEGMENT-LIMIT clause has been used, the segment-numbers from the newly defined limit to 49 become fixed overlayable segments and may be overlaid by an independent segment at runtime.

Example

```
OBJECT-COMPUTER.  IBM-PC,
     MEMORY SIZE 65535 CHARACTERS,
     PROGRAM COLLATING SEQUENCE IS ASCII.
```

5.2.2 SOURCE-COMPUTER Paragraph

Purpose

Specifies the computer on which the program is to be compiled.

Format

SOURCE-COMPUTER. computer-name [WITH DEBUGGING MODE].

Remarks

Except for the WITH DEBUGGING MODE clause, the contents of this paragraph are used for documentation only.

If the WITH DEBUGGING MODE clause is included, source program lines with "D" in column 7 (indicating a debug statement) are compiled. If the WITH DEBUGGING MODE clause is not included, these lines are ignored. See Section 7.4, "Dynamic Debugging Statements," for more information on debug statements.

Example

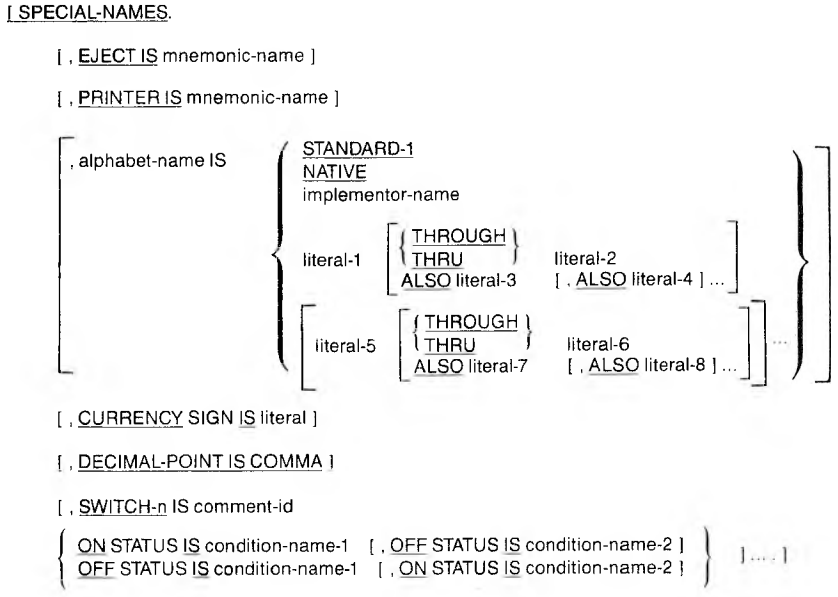
```
SOURCE-COMPUTER.   IBM-PC
    WITH DEBUGGING MODE.
```

5.2.3 SPECIAL-NAMES Paragraph

Purpose

Assigns user-defined names to standard implementor names, such as PRINTER. This paragraph can also be used to change editing characters.

Format



Remarks

The SPECIAL-NAMES paragraph is optional, as is each individual clause within it.

The clauses in this paragraph are discussed on the following pages.

ALPHABET-NAME

The "alphabet-name" clause specifies the language conventions that are used. In MS-COBOL, the default is ASCII IS NATIVE. In MS-COBOL, STANDARD-1 and NATIVE are equivalent.

Implementor-name refers to a name specified by the manufacturer of the computer.

By using the explicit form of the "alphabet-name IS" clause, a unique alphabet can be constructed that can be used to specify a different collating sequence for SORT and MERGE statements, or a different collating sequence for the entire program, when used in the PROGRAM COLLATING SEQUENCE clause.

For example:

```
OTHER-ALPHA IS "A" THRU "Z", "0" THRU "9".
```

defines an alphabet named OTHER-ALPHA, consisting of only letters and digits, where the letters precede the digits in the collating sequence. In the standard set, digits precede letters, and other characters are also included.

The ASCII character set is given in Appendix D.

CONSOLE

The CONSOLE IS clause allows a user-defined name to be used in the DISPLAY or ACCEPT statement with the UPON phrase.

CURRENCY SIGN

In MS-COBOL, the default currency sign is the dollar sign ($). The user may change this sign by specifying a single-character, non-numeric literal in the CURRENCY SIGN clause. The designated character may not be a quotation mark, a digit (0-9), or any of the characters defined for PICTURE representations.

DECIMAL-POINT

The DECIMAL-POINT IS COMMA clause may be included to specify European notation. In European notation, the decimal point and comma are interchanged, so that the representation for pi, for example, is 3,14159.

EJECT

The EJECT IS clause allows a user-defined name to be used in the WRITE statement for Sequential files and causes page ejection.

PRINTER

The PRINTER IS clause allows a user-defined name to be used in the DISPLAY statement with the UPON phrase.

SWITCH-n

The SWITCH-n clause allows switches to be set at runtime. The maximum number of switches that can be set is 8. The user is prompted at runtime to enter the switch settings; the condition-name may then be used in a condition statement in the PROCEDURE DIVISION. The default setting is OFF.

Example

```
SPECIAL NAMES.
    PRINTER IS LPRINTER
    ASCII IS NATIVE
    CURRENCY SIGN IS "L"
    DECIMAL-POINT IS COMMA
    EJECT IS NEWPAGE
    SWITCH-2 IS TEST2
    ON IS ON2
    OFF IS OFF2.
```

5.3 INPUT-OUTPUT SECTION Header

Purpose

Indicates the beginning of the INPUT-OUTPUT SECTION.

Format

This section header must be the first entry in the INPUT-OUTPUT SECTION, and must be entered exactly as shown below, including the period (.). The header must start in Area A.

```
[ INPUT-OUTPUT SECTION.
  FILE CONTROL. (file-control-entry) ...
[ I-O-CONTROL. input-output-control-entry ] ]
```

Remarks

The INPUT-OUTPUT SECTION is required unless the program has no data files. The section begins with the section header and contains two paragraphs, the FILE-CONTROL paragraph and the I-O-CONTROL paragraph. These two paragraphs define the file assignment parameters, including buffering. For more information on these paragraphs, see the individual listings in this chapter.

Example

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
  .
  .
  .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  .
  .
  .
I-O-CONTROL.
  .
  .
  .
```

5.3.1 FILE-CONTROL Paragraph

Purpose

Names the files that are to be processed and associates them with specific input or output devices.

Format

The FILE-CONTROL paragraph of the ENVIRONMENT DIVISION describes each file whose records are subsequently described in the FILE SECTION of the DATA DIVISION.

The general format for a FILE-CONTROL paragraph with SEQUENTIAL or LINE SEQUENTIAL file organization is:

```
FILE-CONTROL.
SELECT [ OPTIONAL ] file-name
    ASSIGN TO    { DISK    }
                 { PRINTER }
    [ ; [ LOCKING IS ] EXCLUSIVE                ]
    [ ; RESERVE integer  [ AREA  ]              ]
                         [ AREAS ]
    [ ; ORGANIZATION IS [ LINE ] SEQUENTIAL ]
    [ ; ACCESS MODE IS SEQUENTIAL ]
    [ ; File STATUS IS data-name-1 ].
```

The following format applies to sort files:

```
FILE-CONTROL.
SELECT file-name
    ASSIGN TO DISK
[ SORT STATUS IS data-name-1 ].
```

Two other formats are available for INDEXED and RELATIVE file organizations. See Chapters 11 and 12 for these formats.

Remarks

For the two formats given here, the SELECT and ASSIGN clauses are required; all other clauses are optional.

Example

```
FILE-CONTROL.
    SELECT INVENTORY-RECORDS
        ASSIGN TO DISK.
    SELECT INVENTORY-REPORT
        ASSIGN TO PRINTER.
```

5.3.1.1 ACCESS MODE Clause

Purpose

Specifies the method of access to records.

Format

The three formats of the ACCESS MODE clause are dependent on file organization. The formats that follow apply to Sequential, Indexed, and Relative files, respectively.

```
[ ; ACCESS MODE IS SEQUENTIAL ]

[ : ACCESS MODE IS { SEQUENTIAL } ]
                   { RANDOM     }
                   { DYNAMIC    }

[ : ACCESS MODE IS { SEQUENTIAL   [ , RELATIVE KEY IS data-name-1 ] } ]
                   {                                                }
                   { { RANDOM  }                                    }
                   { { DYNAMIC }     , RELATIVE KEY IS data-name-1  }
```

Remarks

Depending on the structure of the data file, the access mode can be defined as SEQUENTIAL (the default), RANDOM, or DYNAMIC (alternately SEQUENTIAL or RANDOM).

If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is implied.

The access mode must be specified or implied before any input-output instructions are executed.

If you have specified SEQUENTIAL or LINE SEQUENTIAL in the ORGANIZATION clause, the access mode must be SEQUENTIAL. If you have specified INDEXED or RELATIVE organization, the access mode may be RANDOM, SEQUENTIAL, or DYNAMIC.

The ACCESS MODE clause is optional. If used, this clause must begin in Area B, and is generally indented from the SELECT clause for readability.

5.3.1.2 ASSIGN Clause

Purpose

Specifies that a file is to be used with a particular input or output device.

Format

The ASSIGN clause always appears as part of the SELECT clause. It may be entered on the same line as the SELECT clause, but is generally entered on the following line and indented for readability.

The general format for SEQUENTIAL or LINE SEQUENTIAL file organization is:

```
ASSIGN TO { DISK    }
          { PRINTER }
```

The period (.) appears at the end of the sentence that comprises the entire SELECT clause. This means that if the ASSIGN clause is followed by optional clauses, the period will appear at the end of the entire sequence, rather than after ASSIGN.

The general format for Relative and Indexed files and sort files is:

```
ASSIGN TO DISK
```

5.3.1.3 FILE STATUS Clause

Purpose

Specifies the data-item that will receive file status codes from file I-O operations.

Format

```
[ ; FILE STATUS IS data-name-1 ].
```

Remarks

In the FILE STATUS clause, data-name refers to a two-character, alphanumeric item in the WORKING-STORAGE or LINKAGE SECTIONS of the DATA DIVISION. File status information will be placed in this item after an I-O statement. The left-hand character of data-name assumes the following values:

0	for successful completion
1	for end-of-file condition
2	for INVALID KEY (only for Indexed and Relative files)
3	for a nonrecoverable (I-0) error
9	for special cases

The right-hand character of data-name is set to zero if no further status information exists for the previous I-O operation.

The following combinations of values are possible:

Left Character	Right Character	Meaning
0	0	OK
1	0	EOF
3	0	Permanent error
3	4	Disk space full

For values of status-right when status-left has a value of 2 or 9, see Chapters 10, 11, and 12, "Sequential Files," "Indexed Files," and "Relative Files," respectively.

In an OPEN INPUT or OPEN I-O statement, a file status of "30" means "File Not Found."

5.3.1.4 LOCKING Clause

The LOCKING clause specifies the file locking mode (EXCLUSIVE, AUTOMATIC, or MANUAL) that a process will exercise at runtime. See Chapter 17, "File and Record LOCKING," for details about the LOCKING clause.

5.3.1.5 ORGANIZATION Clause

Purpose

Specifies whether the structure of a file is SEQUENTIAL, LINE SEQUENTIAL, INDEXED, or RELATIVE.

Format

[; ORGANIZATION IS [LINE] SEQUENTIAL]

[; ORGANIZATION IS INDEXED]

[; ORGANIZATION IS RELATIVE]

Remarks

With SEQUENTIAL organization, a two-byte count of the record length is followed by the actual record, for as many records as exist in the file.

With LINE SEQUENTIAL organization, each record is followed by delimiters, usually a linefeed or a carriage return/line feed pair, for as many records as exist in the file. See the *Microsoft COBOL Compiler User's Guide* for the delimiters used in your implementation.

Both forms assume the records in the file are variable-length. For information about INDEXED and RELATIVE file structures, see the introductory material in Chapters 11 and 12.

No COMP-0, COMP-3, or COMP-4 information should be written into a Line Sequential file because these data-items may contain the same binary codes used for the record delimiters. This duplication would cause problems when the file was subsequently read.

5.3.1.6 SELECT Clause

Purpose

Identifies the files that will be used in file I-O operations.

Format

Four general formats are available for the FILE-CONTROL entry. The formats used with files that have SEQUENTIAL or LINE SEQUENTIAL organization and for sort files are given here. See Chapters 11 and 12 for the general formats for INDEXED and RELATIVE file organization.

The format that applies for Sequential and Line Sequential files is:

```
SELECT [ OPTIONAL ] file-name

        ASSIGN TO  {DISK   }
                   {PRINTER}

        [ ; [ LOCKING IS ] EXCLUSIVE ]

        [ ; RESERVE integer [AREA ] ]
                           [AREAS] ]

        [ ; ORGANIZATION IS [ LINE ] SEQUENTIAL ]

        [ ; ACCESS MODE IS SEQUENTIAL ]

        [ ; FILE STATUS IS data-name-1 ].
```

The general format for sort files is:

```
SELECT file-name

        ASSIGN TO DISK

        [ SORT STATUS IS data-name-1 ].
```

The SELECT clause must begin in Area B. After the SELECT clause is entered, the other clauses may be entered in any order. The preceding format shows them in the usual arrangement.

Four of the clauses: ASSIGN, FILE STATUS, OPTIONAL, and ORGANIZATION are described below.

Most of the optional clauses in MS-COBOL default to SEQUENTIAL organization. The ORGANIZATION and ACCESS MODE clauses, if not specified, default to ORGANIZATION IS SEQUENTIAL and ACCESS MODE IS SEQUENTIAL, respectively. For Line Sequential files, ORGANIZATON IS LINE SEQUENTIAL must be specified, and ACCESS MODE IS SEQUENTIAL is optional.

Remarks

The following discussion applies to Sequential and Line Sequential files. For discussion of the use of the SELECT clause for Indexed or Relative files, see Chapters 11 and 12, respectively.

For details about the optional file locking syntax for the SELECT clause which supports processing in a multi-tasking environment, see Chapter 17, "File and Record LOCKING."

ASSIGN Clause

The ASSIGN clause is required. It specifies the type of device that is going to receive the file. The possibilities are DISK or PRINTER. This clause may appear on the same line with the SELECT clause, but is generally indented on a separate line for readability.

The form of the LABEL RECORD(S) clause and the use of the VALUE OF FILE-ID clause in the File Description entry of the DATA DIVISION will depend on the destination of your data file (DISK or PRINTER).

For details about the LABEL RECORD(S) and VALUE OF FILE-ID clauses, see Sections 6.5.17 and 6.5.33, respectively.

FILE STATUS Clause

The FILE STATUS clause is an option of the SELECT clause in the FILE-CONTROL paragraph. FILE STATUS specifies a data-name that refers to a two-character, alphanumeric item in the WORKING-STORAGE or LINKAGE SECTIONS of the DATA DIVISION. File status information will be placed in this item after an I-O statement.

OPTIONAL Phrase

The OPTIONAL phrase may only be specified for input files. The OPTIONAL phrase must be specified if input files are being selected that are not necessarily present each time the object program is executed.

If the file specified with the OPTIONAL phrase is not present at runtime, it may be OPENed for INPUT and CLOSEd, but the first READ of the file causes the AT END condition to occur, and the execution of the READ statement is unsuccessful. The AT END phrase, if present in the READ statement, will be executed, and the file-status item, if declared, will be given the value "10", indicating end-of-file.

ORGANIZATION Clause

In the context of Sequential file processing, the ORGANIZATION clause may specify SEQUENTIAL. In the context of Line Sequential file processing, ORGANIZATION IS LINE SEQUENTIAL must be specified. Both forms assume the records in the file are variable-length.

No COMP-0, COMP-3, or COMP-4 information should be written into a Line Sequential file because these data-items may contain the same binary codes used for carriage return and line feed. This duplication would cause problems when the file was subsequently read.

RESERVE Clause

The RESERVE clause is not functional in MS-COBOL, but is scanned for correct syntax. One physical block buffer is always allocated to the logical record area assigned to the RESERVE clause. This allows logical records to span physical block boundaries. For files assigned to PRINTER, the logical record area is used as the physical buffer as well.

Disk filenames that correspond to the source program filenames provided by the SELECT clause are provided by the user in respective File Description entries using the VALUE OF FILE-ID clause in the DATA DIVISION.

Example

```
SELECT INVENTORY-MASTER-FILE
   ASSIGN TO DISK
   ORGANIZATION IS LINE SEQUENTIAL
   FILE STATUS IS MASTER-STATUS.
```

5.3.2 I-O-CONTROL Paragraph

Purpose

Specifies the points at which the RERUN operation is to be established, the files that will be sharing the same physical buffer space, and the location of files on a multiple file reel.

Format

```
[ I-O-CONTROL.

[ ; RERUN [ ON { file-name-1       } ] EVERY { { [ END OF ] { REEL } }                  OF file-name-2 } ... ]
               { implementor-name }         { { UNIT   }   }                                             }
                                            {   integer-1 RECORDS                                        }
                                            {   integer-2 CLOCK-UNITS                                    }
                                            {   condition-name                                           }

  [ ; SAME [ RECORD ] AREA FOR file-name-3 { , file-name-4 } ... ] ...

  [ ; MULTIPLE FILE TAPE CONTAINS file-name-5 [ POSITION integer-3 ]

        { , file-name-6 [ POSITION integer-4 ] } ... ] .... ]
```

Remarks

The I-O-CONTROL paragraph is optional.

In general, the RERUN clause specifies when and where the rerun information is recorded, the SAME AREA clause specifies that two or more files are to use the same memory area for processing so that memory space can be conserved, and the MULTIPLE FILE clause specifies that more than one file shares the same reel of tape.

The I-O-CONTROL paragraph header must be entered exactly as shown, including the period (.). The header must begin in Area A. The clauses usually begin in Area B for readability.

Further details about the MULTIPLE FILE, RERUN, and SAME AREA clauses are given in Sections 5.3.2.1, 5.3.2.2, and 5.3.2.3, respectively.

Note

While the Microsoft COBOL Compiler recognizes and checks the full language tape-handling and RERUN syntax, it does not support tape-handling or RERUN commands during program execution.

Example

```
I-O-CONTROL.
    SAME RECORD AREA FOR
         INVENTORY-MASTER-FILE,
         INVENTORY-REPORT-FILE.
```

5.3.2.1 MULTIPLE FILE Clause

Purpose

Identifies files and file positions on a single reel of tape that is being shared by more than one file.

Format

```
[ ; MULTIPLE FILE TAPE CONTAINS file-name-5 [ POSITION integer-3 ]
        [ , file-name-6 [ POSITION integer-4 ] ] ... ] ... .
```

Remarks

If all the files have been listed in consecutive order, the POSITION option is not required.

Note

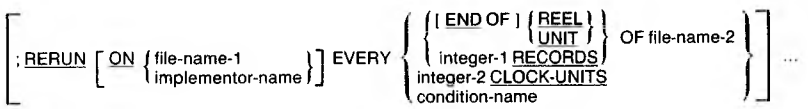
While the Microsoft COBOL Compiler recognizes and checks the full language tape-handling syntax, it does not support tape-handling commands during program execution.

5.3.2.2 RERUN Clause

Purpose

Specifies when and where the rerun information is recorded, and, if necessary, the destination of the updated file.

Format



Remarks

The RERUN clause may take several forms depending on the context under which the RERUN points are to be established.

Note

While the Microsoft COBOL Compiler recognizes and checks the full language RERUN syntax, it does not support RERUN commands during program execution.

5.3.2.3 SAME AREA Clause

Purpose

Specifies that two or more files are to use the same memory area during processing. This clause is generally used to save memory space.

Format

Remarks

This clause is optional. The files named in a SAME AREA clause need not have the same organization or access, but no file-name may appear in more than one SAME AREA clause, and the files cannot be opened concurrently.

If the RECORD option is used, however, all the files may be opened at the same time, provided that none of the files named by the SAME RECORD AREA clause has also been named in a SAME AREA clause.

More than one SAME AREA clause may be included in an I-O-CONTROL paragraph.

Example

```
I-O-CONTROL.
    SAME RECORD AREA FOR
        INVENTORY-MASTER-FILE,
        INVENTORY-REPORT-FILE.
```