DataGeneral

Software Documentation

Interactive COBOL User's Guide (AOS, AOS/VS)

Interactive COBOL User's Guide (AOS, AOS/VS)

069-705015-02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Notice

Interactive COBOL User's Guide (AOS, AOS/VS)
Ordering Number 069-705015
Revision 02, August 1984

(Interactive COBOL, Rev. 1.30)

Original release: August 1982
First revision: June 1983

Changes to Interactive COBOL (revision 1.30)

Language Enhancements

Level 88 is implemented.

The COLUMN and LINE NUMBER clauses accept identifiers as well as literals. This allows field positions to be specified at execution time.

The ACCEPT and DISPLAY *screen-name* statements allow dynamic definition of an entire screen position.

Identifiers can be used with the FROM, TO, and USING clauses.

The display size has been enlarged to 255 lines and 255 columns.

Abbreviated combined relation conditions have been implemented.

The CALL, CANCEL, and EXIT PROGRAM statements have been added. The CALL statement calls an assembly language subroutine, the CLI, or any .PR file.

The ADVANCING clause in the WRITE statement accepts an identifier.

The AFTER clause in the PERFORM statement has been implemented.

Compiler Enhancements

The number of data, procedure, and file references has been increased from 764 to 2294.

Global /I switch allows ICOS cross-development.

Global /A switch implements ANSI 74 standard arithmetic for COMPUTATIONAL items.

Global /O switch suppresses copy files in the listing file.

Global /R switch does not round .PD file to a 2-KB boundary.

The code revision of the .PD and .DD files is 6.

Runtime Enhancements

#N system call renames a file.

CTRL-C CTRL-A terminates an ACCEPT statement.

Display size is determined at execution time with the CHARACTERISTICS command.

Internal computation registers now support 19 digits of accuracy.

On AOS/VS, the MINISAM lock server has been implemented in 32-bit code (in ring 6) of the runtime system process.

In Interactive COBOL revision 1.20, the runtime system was enhanced to handle support of 8-bit characters.

Table of Contents

Chapter 3
The Interactive COBOL Runtime System

Chapter 4
The Interactive COBOL Compiler

Chapter 5
The Interactive COBOL Debugger

Chapter 6 System Management

Chapter 7 Converting between RDOS and AOS or AOS/VS

Appendix A Error Messages

Related Documents

Index

Preface

Document Set

Interactive COBOL is documented by a set of manuals that describe the language, its utilities, and the system-dependent features that affect its use. The *Interactive COBOL Programmer's Reference* defines the Interactive COBOL programming language. It is the programmer's primary reference regardless of the operating system.

The system-dependent *User's Guides* explain the features of the user's particular operating system — RDOS, DG/RDOS, AOS, or AOS/VS — as they relate to Interactive COBOL. Each manual describes such factors as the file system and gives specific instructions for invoking the runtime system, compiler, and debugger.

The set of Interactive COBOL utilities is essentially the same for each system and provides similar functions on each system. However, variations do exist for invoking and using the general utilities on each of the operating systems. A separate *Utilities* manual for each system provides instructions for using the utilities.

In addition to the general utilities, Interactive COBOL includes two special COBOL source editors. *ICEDIT: Interactive COBOL Editor* describes an editor specifically designed for writing programs. *SCREEN: Screen Format Editor* describes the special-purpose editor for designing and automatically coding screen display formats.

The titles and order numbers of the Interactive COBOL documents are listed in "Related Documents" at the end of this manual.

Scope

This manual is written for the COBOL programmer who is familiar with the particular operating system being used. The programmer who is not familiar with AOS or AOS/VS should first consult the documentation related to those systems (see "Related Documents" at the end of this manual). This manual is a companion to the *Interactive COBOL Programmer's Reference*. It describes the relationships between Interactive COBOL and Data General's Advanced Operating System (AOS) or Advanced Operating System/Virtual Storage (AOS/VS).

Organization

The manual is divided into seven chapters and an appendix.

Chapter 1 presents the file system, including naming and managing files.

Chapter 2 discusses Data General's Interactive COBOL system calls, calls to the CLI, and calls to other programs.

Chapter 3 presents the Interactive COBOL runtime system, including its functions and program execution.

Chapter 4 provides instruction on how to operate the Interactive COBOL compiler, including the command line and its switches.

Chapter 5 discusses the Interactive COBOL debugger.

Chapter 6 discusses system management, including PREDITOR, the IPC file, the ISAM server, printer control, and batch processing capabilities.

Chapter 7 describes differences that may arise in converting a program and transporting files between RDOS and AOS or AOS/VS environments.

Appendix A lists File Status codes, Exception Status codes, and compiler, data validation, debugger, and runtime error messages.

Notational Conventions

The conventions described below are used in this manual and the *Programmer's Reference* to represent the various elements of COBOL language syntax. The following example contains most of the elements used in describing COBOL syntax:

DISPLAY { screen-name ... / id-lit } [; WITH NO ADVANCING]

UPPERCASE	Indicates a COBOL reserved word. Underlined uppercase words are required. Nonunderlined uppercase words are optional and are used to improve readability. In either case, all uppercase words must be spelled as shown: no abbreviations are permitted.
lowercase	Indicates a generic term representing words, literals, PICTURE character strings, comment entries, or a complete syntactical entry to be supplied by the programmer. For instance, where *screen-name* appears, the screen name that you have chosen should be used. Throughout this manual, the abbreviations *id, id-lit,* and *lit* are used in the syntax in place of the common COBOL constructs *identifier, identifier-literal,* and *literal.*
Hyphen	A hyphen appearing between uppercase words is required, as in PROGRAM-ID or SOURCE-COMPUTER. A hyphen between lowercase words indicates that the entry chosen by the programmer must not contain any spaces. In the example, the screen-name could be written as ACCTS-PAYABLE or ACCTSPAYABLE, but not ACCTS PAYABLE.
{ }	Braces enclosing part of a format mean that the programmer must select one of the options enclosed within the braces. Thus, the example indicates that either a screen-name or an id-lit must appear in the DISPLAY statement.
[ ]	Brackets enclose optional portions of a format. In the example, the phrase WITH NO ADVANCING is optional.
...	An ellipsis indicates that the item preceding it (defined by logically matching brackets or braces) may be repeated one or more times. In this example, you must enter a screen-name, an identifier, or a literal at least once; the ellipses indicate that you may repeat the entry.
|	Vertical bars in the margin highlight technical changes made since the last revision of this document.
Format Punctuation	The period (.) is required when it is present in a format. The comma (,) and semicolon (;) are optional and interchangeable. They may be used only in certain positions; these positions are indicated by a semicolon. In the example above, a comma or a semicolon may precede the WITH NO ADVANCING phrase. At least one space must follow a comma or semicolon used to separate statements.
Special Characters	When an arithmetic or logical operator (+, –, >, <, or =) appears in a format, it is required. These special characters are not underlined.

Chapter 1
The Interactive COBOL File System

This chapter discusses Interactive COBOL's interface to AOS and AOS/VS file structure and aspects of the file system that are important to an Interactive COBOL programmer.

In an Interactive COBOL program, the name of a file is used:

- To create or edit a COBOL source file. COBOL source text is written into a file using a text editor. This filename is the one used when compiling, debugging, and executing the program.
- To identify a data file. The SELECT entry identifies the data files to be used by the program.
- To identify a program. In the CALL or CALL PROGRAM statement, the filename of the program to be called is specified.

Files in Interactive COBOL are identified in two ways: COBOL (internal) filenames and operating system (external) filenames.

Internal (COBOL) Filenames

An *internal filename* is the name by which a file is referenced within a COBOL program, as in READ TEST-FILE. An internal filename is formed according to the standard COBOL rules for any programmer-created word.

External Filenames

The *external filename* is the name by which a file is known to the operating system. AOS and AOS/VS provide a flexible file-naming scheme. An external filename is formed according to the following rules:

- It may contain characters from the set A-Z (either upper- or lowercase; AOS and AOS/VS do not distinguish between the two), 0-9, dollar sign ($), question mark (?), period (.), and the underscore (_)
- It may contain up to 31 characters. Names to which extensions will be appended should not be longer than 28 characters. For example, an indexed filename can have only 28 characters and an .NX or .XD extension.
- Optionally, a pathname may precede the filename.

Examples of valid external filenames are:

```
TEST_FILE
:UDD:COBOL_PROGS:CUST_CODE2
4TH.QTR.82.INVENTORY?AMT
EMPL$FICA.QTR
```

Directory Structure

On the AOS and AOS/VS systems, files are grouped into directories. A *directory* is a file that is a catalog of other files, including other directories.

The operating system has a *root directory,* which is superior to all other directories. This root directory is represented by a colon (:). The root directory contains files and directories of utilities, peripherals, users, etc.

When you log on to the system, you are normally executing the command line interpreter (CLI) in your user directory. The CLI is your interface to the operating system. Through it you can create files, move them within the directory structure, compile, debug, and run programs, etc. The directory you are in is known as the *current* or *working* directory.

You may create a subordinate directory within any directory with the CREATE/DIRECTORY *directory-name* command. Enter the new directory with the command DIRECTORY *directory-name.* This directory then becomes your working directory. Another directory that is subordinate to the one you are in may then be created, and so forth. The result, as shown in Figure 1-1, is a hierarchical file system.

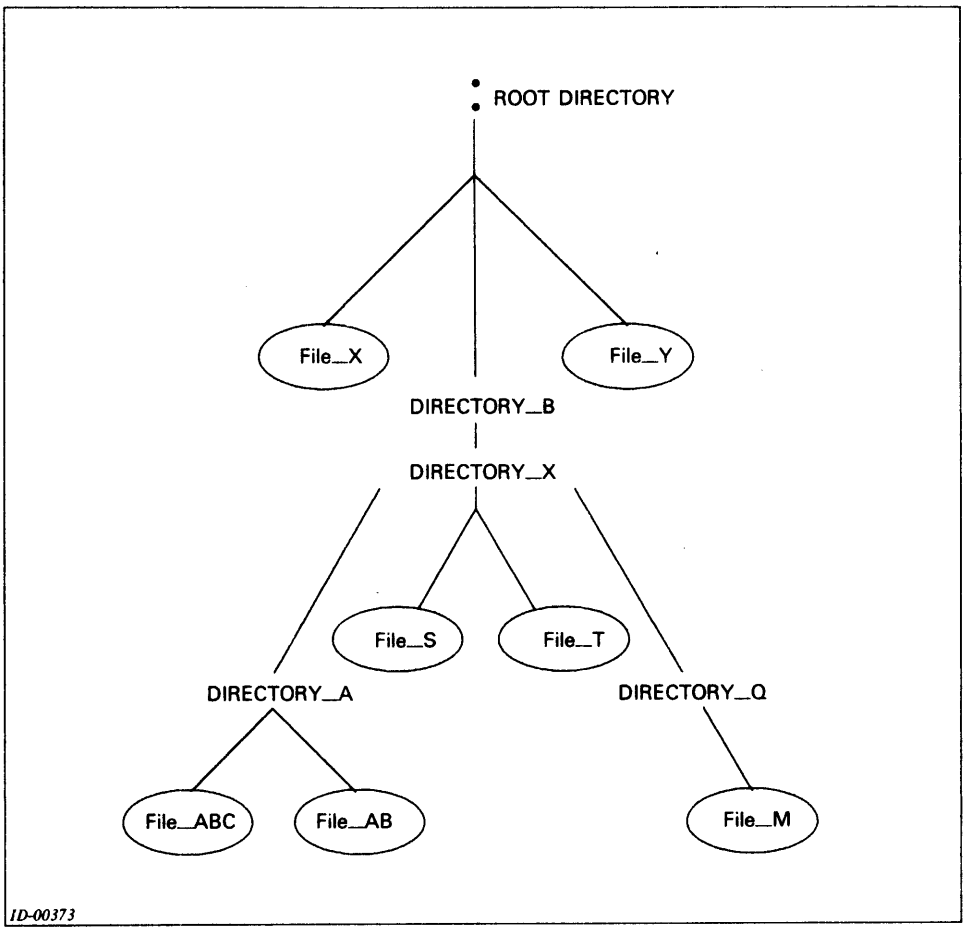


Figure 1-1 *Hierarchical File System*

Pathnames

A file in the working directory may be referenced simply by mentioning the desired filename. However, a file located in a directory other than the working directory must be referenced either by qualifying the name of the file with a pathname or by using a searchlist (see below). A *pathname* represents the unique path through the file system to a specific file. The pathname consists of one or more filenames separated by colons. All filenames except the last one must be the names of directories, and each directory named must be a subdirectory of the preceding one. The maximum number of characters permitted in a pathname is 127. Using the file system in Figure 1-1 as an example, the full pathname from the root directory to FILE_M is:

```
:DIRECTORY_B:DIRECTORY_X:DIRECTORY_Q:FILE_M
```

When a pathname begins with a subdirectory of the working directory, it is called a partial pathname. The following example shows a partial pathname to FILE_M from DIRECTORY_B:

```
DIRECTORY_X:DIRECTORY_Q:FILE_M
```

Pathnames can also start in other directories and be referenced in other ways. If the first character of a pathname is a carat (^), then the pathname begins in the next superior (higher) directory. For example, if your working directory is DIRECTORY_Q, then ^DIRECTORY_X:FILE_S is a path to FILE_S. Similarly, pathnames preceded by an at sign (@) start in the system directory :PER. Pathnames with an equal sign prefix (=) start in the working directory; FILE_M and =FILE_M refer to the same file.

Searchlists

Accessing files in the working directory is simple because the system automatically searches the working directory when it encounters a filename with no pathname. However, referencing several files in different directories by pathnames becomes tedious. Creating a searchlist relieves this problem. The *searchlist* is an ordered list of directories to be searched any time a file referenced with just a filename or partial pathname is not in the working directory. The searchlist can include from one to eight directories (but some programs, such as the compiler, append a directory to the searchlist, so you should not use more than seven). It typically names the directories that are accessed most frequently.

Set the searchlist with the CLI command SEARCHLIST. The format is:

```
SEARCHLIST pathname-1 ... [pathname-7]
```

As an example, using Figure 1-1, suppose that FILE_ABC is a COBOL program, but the program uses files in the root directory and also in DIRECTORY_X. The system automatically searches the working directory, DIRECTORY_A. The following command directs the system to search both the root directory and DIRECTORY_X when a filename is encountered:

```
SEARCHLIST : :DIRECTORY_B:DIRECTORY_X
```

The system does not follow the searchlist if you precede a pathname by an equal sign, an at sign, an up-arrow, or a colon.

File Access Privileges

The AOS and AOS/VS systems provide protection against improper use of files through the access control list (ACL). The ACL contains a list of users who can access a file or directory, plus the types of access each user is entitled to. The five access types and their abbreviations are:

O Owner access
W Write access
A Append access
R Read access
E Execute access

The COBOL statements CALL, CALL PROGRAM, COPY, DELETE, and OPEN require differing types of access. If you do not have the proper access, an error occurs. See the *Interactive COBOL Programmer's Reference* for more details on these verbs.

The range of options with ACLs is virtually unlimited. For example, ACLs could be assigned based upon the following rules:

- The ACL for each user's directory can be set so that the user has OWARE privileges, but no one else has any access. Thus no one can look at the user's files without permission.
- A directory named COMMON can be established with everyone having OWARE access to the directory. This directory would include the files utilized by all users. If a user wants to share a privileged file, he moves it into COMMON and changes the file's ACL.

  Note: Write access to any directory allows a user to add and delete files from the directory and to change the ACL to the directory.
- The ACL of the utilities directory (UTIL) can be set to OWARE for the system manager and RE for all other users. Everyone can then use the utilities, but only the system manager can change them.

For a detailed discussion of creating and working with files using the CLI, see the *Command Line Interpreter (CLI) User's Manual (AOS and AOS/VS)*, 093-000122.

Creating Files in a COBOL Program

Files are automatically created by AOS and AOS/VS when a file that does not exist is opened for OUTPUT or I-O by an Interactive COBOL program. No system commands are necessary, and no information specifying file length need be given. File space is allocated randomly by the system on an as-needed basis, limited only by the physical size of the device or maximum size of the control point directories. Thus you need not specify the size of a file. All blocking is handled by the system.

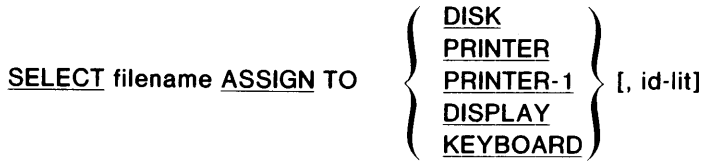
Instructions relating to file creation are given in the Environment and Procedure divisions of a COBOL program. The FILE-CONTROL entry in the Input-Output Section of the Environment Division contains a SELECT entry for each file to be processed by the program. This statement gives the file's internal and external names, organization, and access mode; SELECT also identifies the primary key and any alternate keys for indexed files, names the relative record key for relative files, and identifies an optional FILE STATUS item. In addition, it assigns the file to a device.

In the Procedure Division, the OPEN statement controls file creation according to the parameters specified in the SELECT entry. These instructions contain all the information the operating system needs to create files.

Interface to Data Files

Throughout an Interactive COBOL program, a data file is referenced by its internal (COBOL) name. In the file's SELECT entry, its internal name is assigned to an AOS or AOS/VS external filename. When the Interactive COBOL program is running, the runtime system makes the connection between the internal name and the external name each time an OPEN statement is executed.

In the SELECT entry, Interactive COBOL allows an optional identifier or literal to be supplied by the programmer to identify a specific external filename. The syntax for the SELECT entry is:



The literal must be a valid external filename. If an identifier is used, its full value must be a valid external filename, or the filename must be left-justified in the identifier.

Assigning Files to Disk

An entry of the following form allows a program to access a disk-resident data file:

```
SELECT filename ASSIGN TO DISK [, id-lit]
```

Using the optional identifier or literal allows an external filename to be specified at runtime; if it is omitted in the SELECT entry, the compiler creates an external name from the internal name by (1) truncating the internal name after the tenth character, if necessary, and (2) substituting dollar signs for hyphens. For example:

Internal Name	System External Name
ACCOUNTS-RECEIVABLE	ACCOUNTS$R
GENLEDGER	GENLEDGER
ACT-XY-07-1239	ACTXY07$

Storing the external filename as a data-item rather than coding it as a literal in the SELECT entry allows more flexibility in file naming. The operator or program can fill in the filename at runtime. The following example demonstrates how one internal file may be assigned to different external files at different times during program execution:

```
SELECT DATAFILE-INT ASSIGN TO DISK, EXT-FILENAME.

MOVE "DATAFILE1" TO EXT-FILENAME.
OPEN OUTPUT DATAFILE-INT.
...
CLOSE DATAFILE-INT.

MOVE "DATAFILE2" TO EXT-FILENAME.
OPEN OUTPUT DATAFILE-INT.
...
CLOSE DATAFILE-INT.

MOVE "DATAFILE3" TO EXT-FILENAME.
OPEN OUTPUT DATAFILE-INT.
...
CLOSE DATAFILE-INT.
```

Just as different literals can be moved into the data-item that defines the external filename, you can use the ACCEPT statement to ask the user to identify the file to process. The user's response can be stored in a second data-item and used in the same way as the literal filenames. In this way, the binding of filename to file is deferred until execution, for example:

```
SELECT DATAFILE-INT ASSIGN TO DISK, EXT-FILENAME.

ACCEPT OP-RESPONSE.
MOVE OP-RESPONSE TO EXT-FILENAME.
OPEN OUTPUT DATAFILE-INT.
...
CLOSE DATAFILE-INT.
```

Links may also be used to defer binding the filename to the file until execution. The SELECT entry would be as follows:

```
SELECT DATAFILE ASSIGN TO DISK, "FILE1".
```

Then use the CLI command CREATE with the /LINK switch to link FILE1 with the desired external file.

These features mean that a program is not confined to a particular data file or set of files. The same program code can process several files.

Interface to Devices

A program may perform I/O operations to a physical device instead of to a disk-resident data file. The three devices that you may designate as input or output files are a printer, the display screen, and the keyboard. AOS and AOS/VS consider every physical device a file, although certain physical devices cannot perform certain operations. For instance, you cannot ACCEPT from a printer or DISPLAY to a keyboard. The SELECT formats for assigning files to the physical devices are given below.

Assigning Files to a Printer

Files assigned to PRINTER must be sequential and opened only in OUTPUT or EXTEND modes. To assign files to a printer, use the following format:

SELECT filename ASSIGN TO { PRINTER / PRINTER-1 } [, "device or disk-filename"].

This statement can occur in four forms:

1. SELECT filename ASSIGN TO PRINTER. This sends the output to the default printer queue, LPT.
2. SELECT filename ASSIGN TO PRINTER-1. This sends the output to the queue that is normally used by AOS and AOS/VS as the second printer queue, LPT1. If no LPT1 queue exists, an error message will be displayed.
3. SELECT filename ASSIGN TO PRINTER, "disk-filename". This creates a line sequential disk file and sends the output to the disk-filename. The disk file may be printed later with the CLI command QPRINT. Example:

   ```
   SELECT PRINTOUT ASSIGN TO PRINTER, "EMP_RPT".
   ```

4. SELECT filename ASSIGN TO PRINTER, "device-name". This sends the output to the device or queue that you specify. (Obtain the names of the queues from your system manager.)

   Examples:

```
SELECT PRINTOUT ASSIGN TO PRINTER, "@LPT".
SELECT PRINTOUT ASSIGN TO PRINTER, "@LPB".
```

Assigning Files to the Display Screen

Use a statement in the following form to assign a file to DISPLAY:

```
SELECT internal-name ASSIGN TO DISPLAY [, id-lit]
```

Files assigned to DISPLAY must be sequential and must be opened only in OUTPUT or EXTEND modes. Assign a file to DISPLAY to bypass the Interactive COBOL Screen Section and send output directly to the terminal screen. The file produced is data-sensitive. In general, you would not use an external filename with this statement, although it is not illegal to do so. If you supply a disk-filename, the output will go to the disk file. If you supply an external name in the form @CON*nn*, and the console is enabled by EXEC, you get an error message. @CON*nn* cannot be accessed until it is disabled by EXEC.

Assigning Files to the Keyboard

Use a statement in the following form to assign a file to the keyboard:

```
SELECT filename ASSIGN TO KEYBOARD [, id-lit]
```

Files assigned to KEYBOARD must be sequential and must be opened only in INPUT mode. To read a line sequential disk file or an AOS or AOS/VS data-sensitive file (such as a file generated by a text editor), assign it to KEYBOARD. The READ statement for such files inputs characters up to the first NEW LINE (octal 012), CR (octal 015), NULL (octal 000), or FORM FEED (octal 014).

Default External Filenames

If you do not specify an external filename in the SELECT entry, the compiler generates default filenames. These default filenames are listed in Table 1-1.

Device	Default Filename
PRINTER	$LPT
PRINTER-1	$LPT1
DISPLAY	$TTO
KEYBOARD	$TTI
DISK	First 10 letters of file-name; $ replaces -.

Table 1-1 Compiler Default Filenames

The AOS and AOS/VS runtime systems convert these device filenames to the system filenames listed in Table 1-2. For more details, see the SELECT entry in the *Interactive COBOL Programmer's Reference.*

Device	Default Filename
$LPT	@LPT
$LPT1	@LPT1
$TTO	@OUTPUT
$TTI	@INPUT

Table 1-2 Runtime System Default Filenames

System Files

Interactive COBOL automatically creates a number of special files that are, with the exception of .SR, .CO, and .CLI, exclusively for system use. These files have filename *extensions:* a period and a suffix added to the name of each file being processed. The suffix identifies the file's function. System files are placed in the directory in which the parent file resides. Table 1-3 lists the system extensions. These system filename extensions (with the exception of .SR and .CO) should be avoided when naming files for use with Interactive COBOL.

Extension	System Use
.PD	Procedure Division of a COBOL object program
.DD	Data Division of a COBOL object program
.NX	Index portion of an indexed or relative file
.XD	Data portion of an indexed or relative file
.CLI	System macro file
.SR	COBOL source file (CRT format)
.CO	COBOL source file (card format)
.PR	Executable file
.OL	Overlay file
.ST	Symbol table file
.VM	Virtual memory file
.TX	Text file (ICEDIT)
.DL	Delete file (ICEDIT)
.CU	Cut file (ICEDIT)
.SS	SCREEN source file
.AX, .SX, .DX	SCREEN descriptor files

Table 1-3 System Filename Extensions

Links

Links are useful in reducing storage when more than one person wants to access the same file. For example, a programmer has the program ACCOUNT_REV.2 stored in the directory :UDD:JEFF:COBOL. Three other people want to use that program. They create in their directories the link ACCOUNT with the resolution :UDD:JEFF:COBOL:ACCOUNT_REV.2:

```
CREATE/LINK ACCOUNT :UDD:JEFF:COBOL:ACCOUNT_REV.2
```

Now each time they enter ICX ACCOUNT, the program ACCOUNT_REV.2 is executed.

This use of links has two main advantages: (1) it saves space, and (2) it removes the problems associated with duplication. If Jeff is in charge of maintaining the accounting program, and he changes it, the other people immediately have use of the new version.

Note that the ACLs on the resolution file and the directories must be properly set.

Open File Limits

Files opened by a COBOL program use operating system channels. A sequential file uses one channel. Indexed and relative (ISAM) files use two channels, one each for the .NX and .XD files. The following constraints are placed on the number of simultaneously open files per terminal:

1 <= P <= 16 (AOS and AOS/VS)
P + S + 2I <= 57 (AOS)
P + S + 2I <= 250 (AOS/VS)

where P is the number of programs in the run unit, S is the number of sequential files open, and I is the number of ISAM files open.

In addition, both AOS and AOS/VS have a system-wide maximum of 511 unique ISAM files open at one time.

Record Lock Limits

There is a maximum of 32 locked records per ISAM file at one time. The system-wide maximum is 32 * (number of ISAM files).

ISAM File Requirements

All ISAM files (both .NX and .XD portions) must have a file element size that is a multiple of four, or the OPEN statement fails with a File Status of 91.

In addition, ISAM files open for INPUT must have a file size that is on a 2 KB boundary, or the OPEN statement will fail.

ISAM files created on AOS or AOS/VS by the current revision of Interactive COBOL meet these requirements. However, contiguous files loaded from RDOS might not have a file element size that is a multiple of four, and ISAM files loaded from RDOS or produced by AOS Interactive COBOL revision 1.00 are unlikely to have a byte size that is a multiple of 2048.

The CLI macro MINISAM_CONVERT.CLI, which is included in the release media, converts a file so that its element size is a multiple of four and its byte size is a multiple of 2048. This macro creates a temporary copy of the file. Therefore, check remaining disk space before running the macro on a large file.

The REBUILD utility converts a file to a byte size that is a multiple of 2048. See *Interactive COBOL Utilities (AOS, AOS/VS)* for information on REBUILD.

Exclusive and Nonexclusive File Usage

The EXCLUSIVE option of the OPEN statement for an ISAM file prevents access to the file only by other Interactive COBOL programs and the REORG utility. However, some AOS and AOS/VS utilities can still access the file. Avoid using AOS or AOS/VS utilities on ISAM files when there is a possibility that they are in use. Be sure that files are closed when loading or dumping files in order to prevent possible corruption of the file.

ISAM Reliability Flags

To help provide file integrity, Interactive COBOL provides *ISAM reliability flags*. The reliability flags are set whenever a WRITE, REWRITE, DELETE, or UNDELETE statement changes the contents of the file. If the use count for an ISAM file is 0 at open time, the reliability flags are tested; if they are set, a File Status of 9F, indicating that the file may be corrupt, is returned. If the use count is not 0, the flags are not tested. Whenever a file is closed, the contents of the file are flushed to disk and the reliability flags are cleared. If the file is opened by other users, the flags are reset at the time of the next operation that changes the contents of the file.

If file corruption is detected while ISAM files are in use, all ISAM operations except CLOSE return a File Status code of 9B. CLOSE succeeds with a File Status of 0.

Chapter 2
Communication between Programs

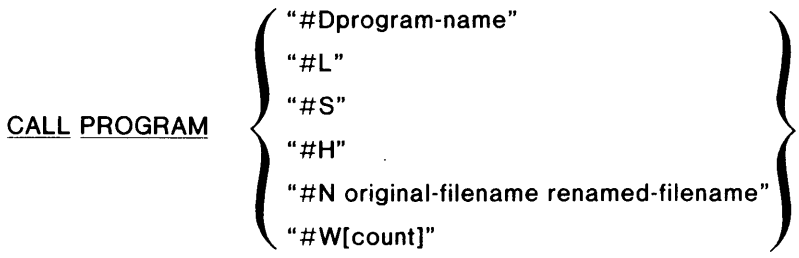
Interactive COBOL provides two statements, CALL and CALL PROGRAM, that allow interprogram communication. The CALL PROGRAM statement can invoke operating and runtime system utility functions or chain to another Interactive COBOL program. The CALL statement can call another Interactive COBOL program, the CLI, any .PR file, or an assembly language subroutine. The *Interactive COBOL Programmer's Reference* discusses calls to other Interactive COBOL programs; other calls are discussed in this chapter.

System Calls

The system call form of the CALL PROGRAM statement differs from the standard form in that the program name begins with the number-sign character (#). As with the standard form, the CALL PROGRAM argument may be either a quoted literal or a data-item identifier. The USING phrase is not valid when executing system functions.

Unlike other CALL PROGRAM statements, control returns to the calling program after execution of a system call. The runtime system updates the Exception Status data-item when it executes a system call. Exception Status codes are listed in appendix A.

The general syntax is:



The functions available on AOS and AOS/VS are:

#D — Enter the debugger. *Program-name* is the name of the program to be debugged. The debug option automatically brings up the debugger across CALL PROGRAMs; thus it is not necessary to recompile and use the #D chain. *Note:* The program to be debugged must have been compiled with the /D switch. For a complete description of the debugger, see chapter 5.

#L — Invoke Logon. Logon is called whether or not you used it to execute the program currently running. That is, if the current program was invoked with the command ICX *program-name* and you did not see the Logon menu initially, you will see it now.

#S, #H — Stop. Shuts down the runtime system and returns control to the program that invoked the runtime system (usually the CLI). If the runtime system was the initial program (i.e., it was specified using PREDITOR), the user is logged off when the runtime system terminates.

#N Rename a file. Separate *original-filename* and *renamed-filename* by at least one space. The original filename can be a pathname or a simple filename; the renamed filename must be a simple filename. Templates are not permitted; for example, to rename an ISAM file, which has an .NX and an .XD portion, you must rename both sections explicitly with two system calls.

#W Wait. This call creates a pause in the program. It might be used to allow a message to be read before the screen is blanked. *Count* specifies the pause in tenths of a second. The maximum value of the literal or data-name is 65,535; thus, the longest possible delay is 1 hour, 49 minutes, 13.5 seconds. If no count is specified, the pause is 3 seconds. After the pause, execution continues with the next statement.

For example, the following command results in a 15-second delay:

```
CALL PROGRAM "#W150".
```

The following technique can be used to produce a delay whose length is specified interactively. These statements appear in the Working-Storage Section:

```
01 WAIT.
    03 FILLER PIC X(2) VALUE "#W".
    03 TIMER PIC 9(5).
```

The Procedure Division contains a DISPLAY-ACCEPT sequence through which the operator enters a length of time for the delay. This value is moved to TIMER. The statement CALL PROGRAM WAIT produces the delay for the specified time.

System Call Errors

The following system calls supported by RDOS and DG/RDOS have no meaning under AOS and AOS/VS: #A, #I, #P, #M, #O, #R, #T, #F, and #C. If a system call is not supported, the system returns an Exception Status code of 203, indicating the program was not found. (See the *Interactive COBOL Programmer's Reference* for information about the Exception Status item.) If there is no ON EXCEPTION clause, the program continues normal execution at the next statement.

Program Calls

The precedence of calls is as follows:

1. Calls to assembly language subroutines
2. Calls to other Interactive COBOL subprograms
3. Calls to the CLI
4. Calls to a .PR file

Calling the CLI from a COBOL Program

An Interactive COBOL program can call the CLI with the CALL statement. The format for calling the CLI is basically the same as calling an Interactive COBOL subprogram.

To call the CLI without passing an argument:

```
CALL "CLI".
```

Alternatively, you can move "CLI" to an identifier and call the identifier. When the call is executed, you remain in the CLI, where you can enter any number of CLI commands. Enter BYE to return to the Interactive COBOL program.

You can specify a single argument with the CALL USING statement. This argument can contain one or more CLI commands, but it cannot be longer than 255 characters. You cannot pass information from the CLI back to your program.

The following program asks you for a directory, into which you will later move a file. Then the program calls the CLI and sorts an indexed file into a line sequential file. A copy of the line sequential file is moved to the directory that you have specified; the CLI deletes any existing file of that name. Then the Interactive COBOL program prints the line sequential file.

```
WORKING-STORAGE SECTION.
01  CLI-CALLER.
    05  SORTER      PIC X(29) VALUE "CSSORT FILE1/I FILE2/L 1:10/K".
    05  SEPARATOR-1 PIC X VALUE ";".
    05  MOVER       PIC X(9) VALUE "MOVE/V/D ".
    05  DEST-DIR    PIC X(127) VALUE SPACES.
    05  FILLER      PIC X   VALUE SPACE.
    05  SOURCEFILE  PIC X(5) VALUE "FILE2".
    05  SEPARATOR-2 PIC X VALUE ";".
    05  QPRINTER    PIC X(12)  VALUE "QPRINT FILE2".

PROCEDURE DIVISION.
    ...
   DISPLAY "Enter DESTINATION directory:  ".
   ACCEPT DEST-DIR.
   CALL "CLI" USING CLI-CALLER.
```

Errors in Calling the CLI

An Exception Status code is set whenever a call to the CLI is made from an Interactive COBOL program. The Exception Status code is the operating system error code returned from the AOS or AOS/VS ?PROC system call. The Exception Status is set to 0 if the call is successful. For a list of operating system codes, see the *Programmer's Manual* for your operating system.

Passing Arguments from the CLI to a COBOL Program

You can pass arguments from the AOS or AOS/VS CLI directly to an Interactive COBOL program. To do so, the Interactive COBOL program must be set up like a called program, with a Linkage Section and a Procedure Division Using header.

To pass arguments to the called program, type a command line in the following form:

ICX program-name {arguments}...

The arguments are transferred to the corresponding data-items in the Linkage Section.

If an argument is shorter in length than the corresponding data-item, the resulting value is left-justified and padded with spaces. For example, moving the argument YES into a data-item defined in the Linkage Section as PIC X(5) results in a value of YESbb.

If an argument is longer than the corresponding data-item, the runtime system returns the error `Argument Length > Item Length`.

If there are fewer arguments on the command line than there are data-items in the Procedure Division Using header, nothing is moved to the remaining arguments and no error message appears.

If there are more arguments in the command line than in the Procedure Division Using header, or if the program has no Procedure Division Using header, the runtime system returns the error **Wrong Number of Arguments.**

The following command line moves the values 12, 000, and NO to DATA-ONE, DATA-TWO, and DATA-THREE in the program ADSYS1.

```
ICX ADSYS1 12 000 NO

PROGRAM-ID. ADSYS1.
...
DATA DIVISION.
...
LINKAGE SECTION.
01   DATA-ONE PIC 99.
01   DATA-TWO PIC 999.
01   DATA-THREE PIC XX.
...
PROCEDURE DIVISION USING DATA-ONE, DATA-TWO, DATA-THREE.
...
EXIT-PARA.
   EXIT PROGRAM.
```

Calling a .PR File

You may call any .PR file from an Interactive COBOL program. The format is:

```
CALL "program-name[arguments]" [USING {id-lit}...]
```

If you specify the USING phrase, the identifiers or literals are concatenated into the initial IPC message sent to the process. Trailing spaces are stripped. The arguments cannot total more than 255 characters. The arguments must be in CLI format, which means that they cannot contain angle brackets, brackets, parentheses, semicolons, or the NEW LINE, CR, formfeed, or null characters.

For example, the program FILCOM.PR compares two files; the command line entry is XEQ FILCOM/L=output-filename file_1 file_2. To call FILCOM from your Interactive COBOL program, have it compare FILE_X with FILE_Y, and put the output in COM_OUT.LS, you would code the following:

```
WORKING-STORAGE SECTION.
...
01  FILE1   PIC X(6)    VALUE "FILE_X".
01  FILE2   PIC X(6)    VALUE "FILE_Y".
PROCEDURE DIVISION.
...
   CALL "FILCOM/L=COM_OUT.LS" USING FILE1 FILE2.
```

Errors in Calling a .PR File

An Exception Status code is set whenever a call to a .PR file is made from an Interactive COBOL program. The Exception Status code is the operating system error code returned from the AOS or AOS/VS ?PROC system call. The Exception Status code is set to 0 if the call is successful. For a list of operating system codes, see the *Programmer's Manual* for your operating system.

Calling Assembly Language Subroutines

An Interactive COBOL program can use the CALL statement to call an assembly language subroutine. For the call to complete successfully, however, you must follow these rules:

- If your assembly language subroutine uses any physical I/O instructions, precede these instructions with the ?LEFD system call, which changes the mode to I/O. After I/O is finished, return to LEF mode with the ?LEFE system call.
- A name table must be defined that gives the names and addresses of all assembly language subroutines called by the Interactive COBOL program. The name table must be defined by the label NAMTB. It must be an external, in order for the runtime system to find it.
- Each assembly language subroutine must have a two-word entry in the name table. The first word is a byte pointer to a null-terminated text string to be used as the name of the routine. The characters in the text string must match the characters of the identifier or literal used in the CALL statement. The second word is the address of the actual subroutine.
- The called subroutine must begin with a SAVE instruction and end with a RTN instruction.
- The parameters must be accessed by indexing off the frame pointer in AC3.
- Any user-defined external names should have names beginning with UU. Names beginning with UU are reserved for user symbols.
- The maximum number of parameters that can be passed to an assembly language subroutine is 32.
- The assembly language subroutines must be linked to the runtime system. To do this, use the ICXLINK macro, which is provided with your system.

  ICXLINK[/O=root-filename] module-name...

  Root-filename is the name of the runtime system you are building. Do not enter the filename with a .PR extension; the runtime system extensions .ST, .PR, etc. are appended automatically. *Module-name* is the name of a module that contains one or more of the subroutines that you are linking to the runtime system. If you omit /O=*root-filename* the runtime system will take the name of the first module you specify.

Linking an assembly language subroutine may reduce the program size that is available. The macro ICXSIZE gives you the maximum program size that can be run by the runtime system.

The following example shows an Interactive COBOL program (ASMCALL) calling two assembly language subroutines. The COBOL program performs the following:

- Allows you to enter a console name and a message
- Calls the assembler subroutine that sends the message you entered to the console name you provided
- Receives a code from the first subroutine indicating whether the call was successful
- Calls the second subroutine if the call was not.

The two assembly language subroutines are ASMX1 and ASMX2. ASMX1 sets up the name table and gets the arguments from the Interactive COBOL program (console name, the message, and the message length). ASMX2 gets the text of the error.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ASMCALL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ICOBOL.
OBJECT-COMPUTER. ICOBOL.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  MESSAGE-LENGTH PIC 99 COMP VALUE 50.
01  ERROR-CODE     PIC 9(4) COMP.
01  CONNAME-STRUCTURE.
    03  CONSOLE-NAME   PIC X(10).
    03  TERMINATOR     PIC X VALUE LOW-VALUE.
01  MESS           PIC X(50).

PROCEDURE DIVISION.

START-PROGRAM.
    DISPLAY "Enter console name: " NO ADVANCING.
    ACCEPT CONSOLE-NAME; ON ESCAPE GO TO FINISH-PROGRAM.
    INSPECT CONSOLE-NAME REPLACING FIRST SPACE BY LOW-VALUE.
    DISPLAY "Enter message: " NO ADVANCING.
    ACCEPT MESS; ON ESCAPE GO TO FINISH-PROGRAM.
    CALL "SEND" USING
        ERROR-CODE, CONNAME-STRUCTURE, MESS, MESSAGE-LENGTH;
        ON EXCEPTION GO TO CALL-FAILED.
    DISPLAY " ".
    IF ERROR-CODE = 0
        DISPLAY "Message sent."
    ELSE
        DISPLAY "SEND Failed:"
        MOVE SPACES TO MESS
        CALL "GET-ERROR-MESSAGE"
             USING ERROR-CODE, MESS, MESSAGE-LENGTH
        DISPLAY MESS.
    DISPLAY " ".
    GO TO START-PROGRAM.

FINISH-PROGRAM.
    STOP RUN.

CALL-FAILED.
    DISPLAY "CALL to SEND failed.".
    STOP RUN.
```

Figure 2-1 *COBOL Program to Call Assembler Subroutines*

```
; ASMX1 - Example program to demonstrate ICOBOL's ability to call
;         assembler subroutines (first module)

        .TITL   ASMX1
        .NREL   1

; Set up the name table
        .EXTN   UUERM

        .ENT    NAMTB
NAMTB:
        SENAM*2
        UUSND
        UUGEM*2
        UUERM
        -1

UUGEM:  .TXT    "GET-ERROR-MESSAGE"

; Define the argument offsets
.DUSR   ARGC=   -5
.DUSR   ARG1=   -6
.DUSR   ARG2=   -7
.DUSR   ARG3=   -10
.DUSR   ARG4=   -11


; SEND - Send a message

; Expects:
;       ARG1 - Error code (returned) (two bytes)
;                -1 if argument count is invalid
;                 0 if call to ?SEND succeeds
;                AOS error code if ?SEND fails
;       ARG2 - Console name (null terminated)
;       ARG3 - Message
;       ARG4 - Message length (one byte)

SENAM:  .TXT    "SEND"

        .ENT    UUSND

UUSND:  SAVE    0

        LDA     0,ARGC,3        ; Get the argument count
        LEF     1,4             ; Form a 4 in AC1
        SUB#    0,1,SNR         ; Is the arg. count 4?
        JMP     LARGS           ;   Yes, arg. count is valid
        MOV#    0,0,SNR         ; Is the arg. count 0?
        RTN                     ;   Yes, no error code arg.
        ADC     0,0             ;   No, return -1 as the
        JMP     SNDER           ;       error code
```

Figure 2-2 *Assembler Subprogram 1—ASMX1 (continues)*

```
LARGS:  LDA     0,ARG2,3        ; Load console name byte ptr.
        LDA     1,ARG3,3        ; Load message byte ptr.
        LDA     2,ARG4,3        ; Load message len. byte ptr.
        LDB     2,2             ; Get message length
        IORI    2B7,2           ; Indicate to ?SEND that a
                                ;   console name is in AC0
        ?SEND                   ; Send the message
        JMP     SNDER           ; Error return

        SUB     0,0             ; Zero indicates success

SNDER:  LDA     2,ARG1,3        ; Get byte pointer to error code
        MOVS    0,1             ; Get high order error in AC1
        STB     2,1             ; Stuff high byte of the error
        INC     2,2             ; Point to the low byte
        STB     2,0             ; Stuff low byte of the error
        RTN

        .END
```

Figure 2-2 *Assembler Subprogram 1—ASMX1 (concluded)*

```
; ASMX2 - Example program to demonstrate ICOBOL's ability to call
;         assembler subroutines (second module)

        .TITL   ASMX2
        .NREL   1

.DUSR   ARGC=   -5
.DUSR   ARG1=   -6
.DUSR   ARG2=   -7
.DUSR   ARG3=   -10


  ; GET-ERROR-MESSAGE

; Expects:
;       ARG1 - Error code (two bytes)
;       ARG2 - Message buffer (filled with error text on return)
;       ARG3 - Message buffer length (one byte)

        .ENT    UUERM
UUERM:  SAVE    0

        LDA     0,ARGC,3        ; Get the argument count
        LEF     1,3             ; Form a 3 in AC1
        SUB#    0,1,SZR         ; Is the arg. count 3?
        RTN                     ;   No, return

        LDA     2,ARG1,3        ; Get byte pointer to error code
        LDB     2,1             ; Get high byte of the error
        MOVS    1,1             ;   in AC1
        INC     2,2             ; Point to the low byte
        LDB     2,0             ; Get low byte of error in AC0
        IOR     1,0             ; Combine the bytes in AC0

        LDA     1,ARG3,3        ; Load message len. byte ptr.
        LDB     1,1             ; Get message length
        MOVS    1,1             ;   in high byte of AC1
        IORI    377,1           ; Use system channel #

        LDA     2,ARG2,3        ; Message buffer byte ptr.

        ?ERMSG                  ; Get the error message
        NOP                     ; Error return
        RTN

        .END
```

***Figure 2-3** Assembler Subprogram 2--ASMX2*

Chapter 3
The Interactive COBOL Runtime System

The Interactive COBOL runtime system is a program that acts as a monitor or executive for the execution of Interactive COBOL object programs. The runtime system performs the following functions:

- Uses a COBOL program named Logon as the default for program execution
- Enables you to interrupt program execution
- Monitors the size of Interactive COBOL programs being loaded
- Interprets input-editing characters entered at the terminal keyboard
- Accepts data and validates data entries in screen fields
- Informs operators of runtime errors
- Maintains system information
- Provides utility functions that support system operations
- Handles the passing of data-items for interprogram communication
- Maintains data-items that contain codes to indicate the status of program execution: identity of the last terminator key pressed by the operator, result of the most recent I/O operation, and result of the most recent CALL or CALL PROGRAM operation
- Controls the Interactive COBOL debugger

Runtime System and Compiler Compatibility

The runtime system and the COBOL compiler are both system programs with revision levels. The runtime system cannot execute a program unless the code revision level of the compiled program is within an acceptable range. The range of acceptable code revisions is kept as wide as possible to provide upward compatibility for code files generated on older systems. The compiler writes its revision level in the program's object code. Use the CREV utility to inspect the code revision level of a program (see *Interactive COBOL Utilities (AOS, AOS/VS)*).

Execution of Interactive COBOL Programs

The runtime system may be called manually from the CLI or automatically by the operating system when a user signs on. Both of these methods are described below.

Starting the Runtime System from the CLI

At the CLI, bring up the runtime system with the following command line:

```
ICX [program-name[/switch]...[arguments]...]
```

The command line allows two possibilities. If you enter ICX, the runtime system looks for a program named Logon and runs it. This program typically displays a menu like that in Figure 3-1. If you enter ICX with a program name, the runtime system executes that program; the Logon menu is not displayed. The optional arguments are passed from the CLI to data-items in the Linkage Section of *program-name*. (See chapter 2 for details on passing data from the CLI to an Interactive COBOL program.)

The runtime system uses the standard AOS and AOS/VS methods for accessing the file; you can use either a partial or full pathname, and the runtime system uses the searchlist to find the file.

Logon

Logon is a program that displays a menu through which you invoke Interactive COBOL programs. When executing Interactive COBOL programs from Logon, the following files must be in the current directory or in the searchlist: ICX.PR, ICX.OL, LOGON.PD, and LOGON.DD. The standard Logon program displays the screen in Figure 3-1.

```
AOS Interactive COBOL - Monitor Revision 1.30
      Standard Development Logon

         DATE: 12/11/83
         TIME: 12:34:56
         Terminal Number: 02



         Option:(?)

       (R)un a COBOL program
       (D)ebug a COBOL program
       (S)top
```

***Figure 3-1** Logon Screen*

Logon provides a 64-character input field for specifying a program to run or debug. The runtime system then locates the object program files, known to the operating system as *program-name*.DD and *program-name*.PD, and execution begins.

Note: Logon uses an ACCEPT statement to get the name of a program from the user. Although underscores are permitted in AOS and AOS/VS filenames, the underscore is a field terminator in an Interactive COBOL ACCEPT statement. If you enter a program name with underscores, all characters to the right of the underscore are deleted.

If you decide not to run or debug a program, select the Stop option to return to the CLI.

Logon is a COBOL program and is supplied in both object and source formats. You can modify it to suit the needs of the application. If Logon is modified, it must be recompiled.

Starting the Runtime System Automatically

The system manager can also create environments in which the runtime system is brought up automatically. Using the program PREDITOR, the system manager specifies the program that is executing when the user signs onto AOS or AOS/VS. In most cases this is the CLI, since it gives the user maximum flexibility. However, the system manager may specify the runtime system as a user's sign-on program. As mentioned above, the runtime system normally executes the Logon program when it starts. This technique makes the system easy to use for end-users who only run Interactive COBOL programs and do not need to know how to explicitly call the runtime system. This also restricts such users from the CLI.

The second method of starting a user directly at the runtime system is similar to the first. In this method the user enters the CLI after signing on, but the initial IPC file specifies what actions the CLI takes when it starts. A macro file lists the commands the CLI executes initially, such as calling the runtime system. This method increases the system developer's flexibility, since macros can be written to perform functions other than calling the runtime system.

For more details about PREDITOR, see chapter 6 of this manual and *Managing AOS* or *Managing AOS/VS*. Macros are described in the *CLI User's Manual* and *Learning to Use Your AOS System* or *Learning to Use Your AOS/VS System.*

Program Switches

When running a program, the program name can be followed by logical switches, which indicate an ON or OFF status for program-defined switches. The switches are defined in the SPECIAL-NAMES paragraph of the Environment Division of the COBOL program. A logical switch can be any uppercase alphabetic character. For example, typing PAYROLL/D in response to the RUN PROGRAM prompt would indicate ON status for the /D switch.

The CALL statement cannot pass switches to a called program. Switches are inherited from the calling program automatically.

Program Size

The size of an Interactive COBOL program that will run on AOS or AOS/VS is determined by the number of pages rather than the number of bytes. One page is equal to 2048 bytes. Both the .PD file and the .DD file are rounded up to the next higher page size, and this sum must be equal to or less than 16 pages in order for the program to run.

Note: Program size for Interactive COBOL on RDOS is calculated using 512-byte blocks. Because of the rounding used to calculate page size on AOS and AOS/VS, it is possible to create a program that fits within the 31 KB size limitation on RDOS but does not fit within the 32 KB limitation on AOS and AOS/VS. Transporting programs between RDOS and AOS or AOS/VS could thus produce problems. In this situation, examine the file size statistics that are automatically provided during compilation, then refer to chapter 4, "The Interactive COBOL Compiler," in this manual and in the *Interactive COBOL User's Guide (RDOS, DG/RDOS)* for details on solving this problem.

Interactive Data Entry

During execution of an ACCEPT statement, you type characters into input fields and press field terminator keys to move among the several fields processed by the statement. At the last input field, pressing a field terminator completes the ACCEPT statement.

During this process, the runtime system performs several functions:

- It moves the terminal cursor among the input fields and supports several input-field editing features.
- It generates codes that indicate the field terminator key (including function keys) most recently pressed by the operator. These codes are stored in a data-item named ESCAPE KEY.
- It validates the contents of input fields against corresponding PICTURE statements.

Note: In this chapter, no distinction is made between TO fields, USING fields, and fields that contain both a FROM and a TO, since they are all handled similarly. The general term *input field* refers to all of these fields. For screen fields specified with the AUTO descriptor, you need not press a terminator key. The runtime system acts as if a terminator had been pressed when you fill an AUTO field with characters.

Chapter 3 of the *Programmer's Reference* gives detailed information on data movement during execution of the DISPLAY *screen-name* and ACCEPT *screen-name* statements.

Input Field Editing

At each input field, you can type as many characters as the PICTURE of the field specifies. If you attempt to type past the end of the field, the terminal's tone sounds. While entering characters in an input field, you may use the editing keys listed in Table 3-1.

Field Terminator Keys

The runtime system recognizes the following field terminator keys: NEW LINE, CR, TAB, down-arrow, ESC, function keys used alone (see exceptions below), and function keys used in combination with CTRL (or CMD), SHIFT, or both.

CR. When CR is used as a terminator, it erases the character at the cursor and all characters to the right of the cursor.

Single-Field Screens. If the screen-name has a single input field, the terminator keys treat it as both the first and final input field.

The ESC Key. Unlike the other field terminators, ESC does not move the contents of the current input field to the receiving data-item. This data-item remains unchanged, the ACCEPT statement terminates immediately, and the ON ESCAPE statement, if any, is invoked. This "cancel data entry" effect applies only to the current field; entries made in previous screen-name fields have already been moved and are accepted.

Implementing the Function Keys

The function keys other than ESC do not cancel the current input field. They move the contents of the current field to the receiving data-item before terminating the ACCEPT statement and invoking the ON ESCAPE statement.

When you press a function key during an ACCEPT statement, the runtime system records the key as a two-digit code in the data-item ESCAPE KEY. Use of non-function-key terminators is also recorded.

Key	Action
DEL	Replaces the character preceding the cursor with a space and moves the cursor back to that space. Ignored if the cursor is at the beginning of the field.
Left-arrow	Moves the cursor to the left one character. Ignored if the cursor is at the beginning of the field.
Right-arrow	Moves the cursor to the right one character. Ignored if the cursor is at the last character in the field.
Up-arrow	Moves the cursor to the previous input field. When pressed at the first input field, returns the cursor to the beginning of the field. The contents of the field are still moved to screen storage, and the ACCEPT statement remains in effect.
Underscore	Indicates a logical end-of-field. When you press a terminator key to enter the field, the runtime system moves only the characters preceding the first underscore in the field to the receiving data-item.
ERASE EOL	Erases from the cursor position to the end of the line. Does not move the cursor.
CTRL-A	Moves to the end of the characters that have been entered on the current line.
CTRL-B	Moves to the end of the previous word.
CTRL-E	Turns insert mode on or off. CTRL-E works only if the input field is not full. Note that the runtime system may fill the field with underscores, and a COBOL program fills the field with blanks.
CTRL-F	Moves to the beginning of the next word.
CTRL-H	Moves to the beginning of the input area without erasing any characters.
CTRL-I	Terminates the field.
CTRL-K	Erases everything to the right of the cursor. (Same as ERASE EOL.) Does not move the cursor.
CTRL-U	Moves to the beginning of the input field and erases the entire field.
CTRL-X	Moves the cursor to the right one character (same as right-arrow).
CTRL-Y	Moves the cursor to the left one character (same as left-arrow).

Table 3-1 Editing Keys

An Interactive COBOL program can branch on function keys by using the value of ESCAPE KEY. To implement function keys, perform the following:

1. In the ACCEPT statement that handles input, code an ON ESCAPE *imperative-statement* clause. This clause is executed only if you press a function key (including ESC) to terminate the ACCEPT.
2. Code *imperative-statement* to execute a routine that retrieves the value of ESCAPE KEY, then branches on this value. The value of ESCAPE KEY cannot be used directly. It must first be loaded into an identifier using an ACCEPT id FROM ESCAPE KEY statement:

```
77 FN-KEY-CODE PIC 99.

...
ACCEPT FN-KEY-CODE FROM ESCAPE KEY.
IF FN-KEY-CODE = 01 PERFORM ESC-ROUTINE
  ELSE IF FN-KEY-CODE = 02 PERFORM FN1-ROUTINE
    ELSE ...
```

Note: An ON ESCAPE clause is not absolutely necessary. The statement following the original ACCEPT can perform the ESCAPE KEY processing.

Interactive COBOL implements up to 15 function keys. However, different terminals have differing numbers of function keys, and their positions and markings also differ. For details, refer to your terminal operations manual.

Table 3-2 lists the two-digit codes loaded into ESCAPE KEY during execution of ACCEPT statements.

	Key Alone	Key+SHIFT	Key+CTRL	Key+CTRL+SHIFT
NEW LINE	00	00	00	00
CR	00	00	00	00
TAB	00	00	00	00
Down-arrow	00	00	00	00
ESC	01	01	01	01
F1	02	10	18	26
F2	03	11	19	27
F3	04	12	20	28
F4	05	13	21	29
F5	06	14	22	30
F6	07	15	23	31
F7	08	16	24	32
F8	09	17	25	33
F9*	-	41	48	55
F10*	00	42	49	56
F11*	00	43	50	57
F12	37	44	51	58
F13	38	45	52	59
F14	39	46	53	60
F15	40	47	54	61

Table 3-2 Terminator ESCAPE KEY Codes

*Function key 9 is equivalent to a hyphen (i.e., minus). It is not a terminator. Function keys 10 and 11 are equivalent to the keys labeled ENTER and ENTER - on some terminals.

Input Field Validation

After a display field is terminated, the runtime system validates its contents:

- For an ACCEPT *screen-name* statement, the field is validated against the PICTURE of the associated Screen data-item.
- For an ACCEPT *id* statement, the field is validated against the PICTURE of *id* in the File, Working-Storage, or Linkage Section.

If the field does not match the PICTURE, an error message appears at the bottom of the screen and the cursor moves to the first invalid character of the field. The error message describes the error and shows the correct PICTURE. Appendix A lists the data validation error messages.

File Status

Every time the runtime system performs an I/O operation, it generates a two-byte File Status code indicating the outcome of the operation. If a FILE STATUS IS *data-name* clause appears in the SELECT clause and the data-name is defined as PIC XX in the Working-Storage Section, a value indicating the status of the I/O operation is placed into the specified two-character data-item. This occurs before any applicable USE procedure, AT END phrase, or INVALID KEY phrase is executed. You can test this item to determine the condition that terminated the I/O operation. File Status codes are most commonly examined in the USE procedures of the Declaratives portion of a program, but they may be examined anywhere within the Procedure Division. Appendix A contains a table of File Status codes and their meanings.

Data-Items Maintained by the Runtime System

The runtime system maintains a number of data-items that can be passed to your program: DATE, DAY, TIME, LINE NUMBER, USER NAME, ESCAPE KEY, and EXCEPTION STATUS. Your COBOL program must not define these items explicitly. Instead, set up a data-item in Working-Storage, pass the value of the system data-item to the Working-Storage data-item, and examine the Working-Storage data-item. Table 3-3 describes these data-items and lists the PICTUREs required for the Working-Storage items that receive their values.

Use an ACCEPT statement to pass the system data-item to the Working-Storage data-item:

```
ACCEPT id FROM { DATE
               | DAY
               | TIME
               | LINE NUMBER
               | USER NAME
               | ESCAPE KEY
               | EXCEPTION STATUS }
```

Data-Item	Required PIC	Description
DATE	9(6) or X(6)	Current date: yymmdd
DAY	9(5) or X(5)	Current day of the year: yyddd (ddd = 1,2,3,...,366)
TIME	9(8) or X(8)	Current time: hhmmss00 (The last two digits are always zeros.)
LINE NUMBER	999 or XXX	Terminal on which the program is running.
USER NAME	X(15)	Allows the program to retrieve the user name of the current process.
ESCAPE KEY	99	Contains the two-digit code generated by a termination key.
EXCEPTION STATUS	999	Status of most recent CALL or CALL PROGRAM, and I/O system errors.

Table 3-3 Data-Items Maintained by the Runtime System

Exception Status

Every time the runtime system executes a CALL or CALL PROGRAM statement, it generates the three-character data-item named EXCEPTION STATUS that records the operating system exceptional condition code. An Interactive COBOL program can recover from a failing CALL or CALL PROGRAM statement and take appropriate action by using the value of EXCEPTION STATUS. To implement this facility, perform the following steps:

1. In the CALL or CALL PROGRAM statement, include an ON EXCEPTION *imperative-statement* clause. This clause is executed only if the call is unsuccessful and processing of the calling program continues.
2. Code the imperative-statement to execute a routine that retrieves the value of EXCEPTION STATUS, and then displays the value to the user or branches on the value. The value of EXCEPTION STATUS cannot be used directly but must first be loaded into a data-item using an ACCEPT data-item FROM EXCEPTION STATUS statement. This ACCEPT statement must immediately follow the CALL or CALL PROGRAM statement, or the code found in the EXCEPTION STATUS item will be undefined.

The following example illustrates these steps.

```
77 EXC-CODE PIC 999.
...
CALL PROGRAM "PROG1".
ACCEPT EXC-CODE FROM EXCEPTION STATUS.
IF EXC-CODE = 203 PERFORM NOT-FOUND
ELSE IF EXC-CODE = 201 PERFORM BAD-REVISION
  ELSE ...
```

Note: If there is no ON EXCEPTION clause, execution of the calling program continues at the next statement. That statement can be coded to perform the EXCEPTION STATUS processing. It is good programming practice to use one of these methods, since there is no system-generated error message that informs the operator that the CALL or CALL PROGRAM statement has failed.

Appendix A contains a list of Exception Status codes and their meanings.

Normal Exit from a Program

If the program is executed through the Logon menu, a normal termination of the program causes the message STOP RUN to be displayed. Pressing NEW LINE, CR, or ESC returns the Logon display. If the program was executed directly from the CLI (by entering ICX *program-name*), normal termination returns you to the CLI. No message is displayed.

Terminating Program Execution

The runtime system supports two ways of interrupting program execution: terminating execution of an Interactive COBOL program, and terminating execution of the runtime system itself.

Program Termination

To stop an Interactive COBOL program and return control to Logon, you may do any of the following:

- Include a CALL PROGRAM "#L" system call in your program. When this is executed, the Logon screen is displayed and the runtime system waits for the operator's input. This method can be used whether the program is called from Logon or is started directly from the CLI.
- Include a STOP RUN statement in your program and execute your program from Logon. When the STOP RUN is executed, the runtime system displays the message `STOP RUN` and pauses. Press NEW LINE or CR to redisplay the Logon menu. If the program is called from the CLI, STOP RUN returns to the CLI; no message is displayed.
- Press CTRL-C followed by CTRL-A if the program is called from Logon. This has the same effect as a STOP RUN statement, except that the message `Program Stopped by Console Interrupt` is displayed.

Runtime System Termination

To stop the runtime system and return to the CLI, do any of the following:

- Include a CALL PROGRAM "#S" or "#H" system call in your program. This shuts down the runtime system and returns you to the CLI regardless of whether the program was called from Logon or from the CLI.
- Select the STOP command from the Logon menu.
- Include a STOP RUN statement in your program and execute the program with the ICX command line.
- Press CTRL-C CTRL-A, CTRL-C CTRL-B, or CTRL-C CTRL-E if the program has been invoked from the CLI. The CTRL-C CTRL-E sequence creates a breakfile. All three sequences allow for the current operation to be completed and a cleanup to be executed. (A cleanup releases system resources in a logical way; i.e., locked records are unlocked, files are updated, etc.) If you issue a CTRL-C CTRL-B or CTRL-C CTRL-E sequence and the current operation has not been completed after a five-second delay, the runtime system terminates without cleanup. The message `Program Stopped by Console Interrupt` is displayed.
- If you cannot stop the runtime system with any of the methods listed above, issue a TERMINATE command from another CLI. If a record is being written or rewritten to an ISAM file when you issue this command, or if the runtime system terminates without cleanup after a CTRL-C sequence as described above, the file can become corrupted. If this occurs, the ISAM reliability flags (see chapter 1) are not cleared when the file is closed. Furthermore, any other users who attempt to access the file (except for the CLOSE statement) fail with a File Status of 9B. If you TERMINATE the runtime system, or if the runtime system terminates without cleanup, run ISAMVERIFY on your ISAM files. The *Utilities* manual contains instructions on running ISAMVERIFY.

Runtime System Failure

Runtime system failure can be caused by an error in the command line that brings up the runtime system, a system call failure, or an attempt to run an Interactive COBOL program that was compiled with errors. When these errors occur, the runtime system returns a severity level to the parent process. The severity levels are IGNORE, WARNING, ERROR, and ABORT. The severity level at which the CLI or EXEC is processing warnings, errors, and aborts is listed in the *CLI User's Manual*.

If you have entered an incorrect command line or if a system call has failed, the runtime system returns an error code and a severity level to the AOS or AOS/VS CLI. The severity level for a command line error is ERROR, and the CLI displays the associated error message. These error messages are listed in the *Programmer's Reference* for your system. If, for example, you specify a nonexistent file-name for the runtime system, the AOS or AOS/VS CLI sends you the error message PROGRAM NOT FOUND and you return to the CLI. If your program was not called from the CLI, the error messages displayed and actions taken reflect the parent program.

If you have attempted to run an Interactive COBOL program that was compiled with errors, the results are unpredictable. If a fatal COBOL program error occurs, the runtime system returns a severity level of ABORT, halts program execution, and displays an error message on the screen. Copy the error message and the data displayed. If the program was executed from Logon, pressing NEW LINE returns you to Logon. If the program was executed from the CLI, the system automatically returns to the CLI.

The error messages have the form:

```
ERROR: error text          [COBOL PC = relative-procedure-address]
```

Relative-procedure-address is a five-digit number that the Interactive COBOL compiler inserts in an output listing in place of a procedure name's line number. It gives the approximate location of the statement that caused the error. To obtain more precise locations, examine a decompilation listing of the program, which is obtained with the /U compiler switch.

If the runtime system fails catastrophically, the severity level is ABORT. Control returns to the CLI, and a message in the following form is displayed:

```
Runtime System Panic
[Error message text]
PC=nnnnnn AC0=nnnnnn AC1=nnnnnn AC2=nnnnnn AC3=nnnnnn
```

PC is the runtime system's program counter and the ACs are the accumulators. A breakfile under AOS, or a memory image dump under AOS/VS, is created in the current directory. On AOS, the file is named *?pid.hh_mm_ss*.BRK. On AOS/VS, the file is *?pid.hh_mm_ss*.7.MDM. *Pid* is the process ID of the runtime system, and *hh_mm_ss* is the time of the failure. The values of the PC and the accumulators are saved in the file. The breakfile and memory image dump are temporary files that are deleted whenever the operating system is rebooted. Rename the files so that they do not begin with a question mark to prevent their automatic deletion.

Appendix A lists the runtime system error messages and gives an explanation of each.

Chapter 4
The Interactive COBOL Compiler

The Interactive COBOL compiler transforms a COBOL source program into object code that can be processed by an AOS, AOS/VS, RDOS, or DG/RDOS runtime system. This compiler can process source files in CRT or card format (with or without line numbers) and with either indexed or sequential file organization.

The compiler produces up to four output files:

- Two object files constituting the executable program:
  source-filename.DD, containing the Data Division
  source-filename.PD, containing the Procedure Division
- An optional listing file
- An optional error file

The Command Line

The compiler is a program named ICOBOL.PR. It is invoked with a CLI command. As with any CLI command, a compilation command can be included in a CLI macro (see the *CLI User's Manual*).

The compilation command line has the following format:

```
ICOBOL[/global-switches]...source-filename[/l]
```

Source-filename is the file containing the Interactive COBOL source code. The compiler looks for the source file using the following algorithm: First it looks for the filename you typed. Then it looks for the filename with .SR appended to it. Last, it looks for the filename with .CO appended.

The name of the source file cannot exceed 28 characters in length. If the source-filename contains periods, the object files are named by dropping all characters after the final period and appending the .PD and .DD extensions. For example, if a source file is named TEST.ONE.EXTRA, the object files are TEST.ONE.PD and TEST.ONE.DD.

Standard AOS and AOS/VS procedures are followed to find, read, and create files when you operate the compiler. The source file must be found in your searchlist. You must have read access to the source file and any copy files, as well as read and execute access to the directory containing the source and copy files. You must have write access to the current directory and the directory you specify for your listing and error files. The compiler opens temporary files in the current directory. At the end of compilation, any existing .PD and .DD files are deleted and the temporary files are renamed. See chapter 1 and the *CLI User's Manual* for a discussion of these features.

Global Switches

Modify the compiler command line with the following switches; they can be used in any order:

/A Produces ANSI '74 standard arithmetic for COMPUTATIONAL items. SIZE ERROR and truncation occur for values that exceed the number of digits specified in the PICTURE. With relative files having more than 9999 records, the data-name for the relative key must be declared PIC 9(5) COMP to PIC 9(18) COMP. The /A switch cannot be used with the /I switch.

/C The source is in card format. The compiler ignores all characters beyond column 72 of a card-format source line. Absence of this switch indicates CRT format. Maximum line length for source code in CRT format is 132 characters; in card format, 80 characters.

/E[=error-file]
If you specify an error file, errors are listed to the specified file but do not appear on the screen. The file is created if it does not exist. The error messages are appended to the file if it already exists.

If you do not specify an error file or a listing file, no message is displayed if the program compiled without errors. If the program compiled with errors, the message `COMPILED WITH ERRORS` is displayed.

If you do not specify an error file but you do specify a listing file, error messages are sent to the listing file.

/I Allows ICOS cross-development. With this switch, the .PD and .DD files are code revision 4. The current code revision is 6. You cannot use the /A switch with the /I switch. Relative keys must be described as PIC 9(4) COMP. In addition, your program can use only the following new features of Interactive COBOL: level 88, PERFORM...AFTER, abbreviated combined relation conditions, and the compiler /O and /R switches. Other new features, listed in the "Changes to Interactive COBOL" page at the front of this manual, cannot be used.

/L Use current listing file as the compilation listing file.

/L=listing file
Write the compilation listing to the specified file. Create the file if it does not exist. Append the listing to the contents of the file if it already exists.

/N Do not produce an object program. In the absence of this switch, the compiler produces two object files: *source-filename*.DD and *source-filename*.PD. This switch cannot be used with the /U switch. If it is, a compiler error message is displayed.

/X Append a cross-reference table to the listing file.

/O Suppress the listing of COPY files in the listing file.

/R Do not round the .PD up to a 2-KB boundary. Use this switch if you will transport the program to RDOS (see chapter 7 for more information).

/S Append compiler statistics to the listing file.

/U Append a listing of the intermediate code to the listing file. This switch cannot be used with the /N switch. If it is, a compiler error message is displayed.

/D Add a symbol table to the .DD object program file for runtime debugging. This switch is ignored if the object program is suppressed with the global /N switch.

/W Do not print warning messages in the listing file.

Local Switch

/I Indicates that the source code is stored as an indexed data file, under the names *source-filename*.NX and *source-filename*.XD. Interactive COBOL's source-program editor, ICEDIT, automatically produces indexed files.

Example

The following example shows the command line to compile the source file INVENTORY. The global switches request a symbol table and a cross reference table, prepare the program for execution under the Interactive COBOL debugger, and request the creation of a listing file with the name INVEN.LIST.

```
ICOBOL/S/X/D/L=INVEN.LIST INVENTORY
```

Source Listing

If a listing file is requested, the top line of each listing page includes a page number and the program name from the PROGRAM-ID entry. A new page occurs at a form-feed character, at a slash character in the indicator field, or after 60 lines. Line numbers within each page appear at the left margin of the listing file. COPY file lines are listed with their COPY file line numbers, followed by a C.

Warning Messages

These messages are printed only in the listing file. If no listing file is produced, or if the program is compiled with the /W switch, they are not seen.

```
WARNING: "n" ITEMS CORRESPOND
```

In an ADD CORRESPONDING, SUBTRACT CORRESPONDING, or MOVE CORRESPONDING statement, it is not always clear how many items will be affected. This message reports the number of items that do correspond.

```
WARNING: NO "WITH DUPLICATES" CLAUSE. DUPLICATES ARE ALWAYS PERMITTED
```

In Interactive COBOL, the WITH DUPLICATES phrase is not necessary in a SELECT clause, but duplicate key values are still allowed. This warning reminds the programmer of this fact.

Statistics

The following statistics always appear on the console at the end of compilation unless the /E switch is used, and they also are appended automatically to the listing file when one is requested.

- Source Lines: the number of source lines compiled
- Compile Time: the time it took for the computer to compile the source
- Errors: the number of compilation errors detected
- Data Size: the size of the .DD file produced
- Procedure Size: the size of the .PD file produced
- Total Size: the size of the total program

If a listing file is requested and the /S switch is used, the following statistics are also appended to the listing file:

- Initial Free Space: the amount of space available to the compiler at the start of compilation
- Total Memory Used: the amount of memory used by the compiler during compilation
- Memory % Used: (memory used / initial free space) x 100
- Symbol Space % Used: percentage of the available symbol space used for this compilation
- Symbol Count: the number of symbols used in the program
- Data References: the number of times a statement refers to a data-item. The maximum number of data references is 2294.
- Procedure Names: the number of procedure names used by the program
- The symbol name, byte address of .DD storage, length, and data reference table index number for each symbol

Error Messages

When the compiler detects an error in a program, it writes an error message to the error file. If the error file is the same as the listing file, the error messages appear below the line containing the error. In some cases, the messages may appear two lines below the line of the error; in other cases, an error message does not appear until the end of the division. The source line count, error count, compilation time statistics, and object program size statistics are included in the error file at the end of compilation.

The compiler error messages are listed in appendix A.

Chapter 5
The Interactive COBOL Debugger

The Interactive COBOL debugger is a feature of the runtime system that allows you to control the execution of a COBOL program interactively. The debugger allows you to:

- Execute the program one procedure (paragraph or section) at a time
- Set trap points that suspend program execution at the beginning or end of a procedure
- Set or examine the value of any data-item, except those defined as COMPUTATIONAL or INDEX, while program execution is suspended

Using these functions, you can isolate logical errors and problems in program design and relate them to specific areas of a program's Procedure Division.

Starting the Debugger

Using the Interactive COBOL debugger involves three steps:

1. Before compiling the program for debugging, be sure that procedure names have been inserted near the COBOL statements where execution is to be halted. The debugger can suspend program execution only at the start or end of a paragraph or section.
2. Compile the program with the global /D switch:
   ICOBOL /D [additional switches] program-name [arguments]
   The /D instructs the compiler to add a symbol table to the data portion of the object program, *program-name*.DD. This table allows the debugger to locate the beginning and end of each procedure. It also provides data-names and their attributes so that their values can be set and displayed. If you invoke the debugger without first compiling the program with the global /D switch, the program executes as though you have entered ICX *program-name*.
3. Debug the program. To debug the program you can:
   a. Use the debug program option on the Logon menu (the *D* choice).
   b. Code a debugging system call in either of the following ways:
      CALL PROGRAM "#Dprogram-name".
      CALL PROGRAM id.
      where *id* has the value of #D*program-name*. *Program-name* is the name of the program to be debugged.
   c. Call the debugger directly from the CLI: ICDEBUG *program-name*. *Program-name* is the name of the program to be debugged.

The runtime system loads the program and its symbol table, then passes control to the debugger, which is implemented as part of the runtime system itself. The debugger displays its command prompt, an exclamation point (!).

Using the Debugger

Following the debugger's prompt (!) is a 79-character input field in which you enter debugger commands. This field is a standard Interactive COBOL input field (FROM SPACE TO data-item). Thus you can use the standard input-editing keys listed in Table 3-1.

The field-terminator keys that cause the debugger to act on a command are also the same as the field terminators for an ACCEPT. They are listed in Table 3-2.

As it executes commands, the debugger informs you of its Procedure Division location. Whenever program execution is suspended, the debugger displays its paragraph and section.

If you enter an illegal command, the debugger responds with an error message and redisplays its prompt. Debugger error messages are listed in appendix A.

Terminating the Debugger

During a debugging session, you enter commands to start or suspend execution and set or examine data-item values. The debugging session ends if the program commits a fatal error, if it comes to a STOP RUN statement, or if you issue the debugger command STOP.

If the debugger has been called from Logon, control returns to Logon (when you press NEW LINE after a STOP RUN). If the debugger has been called from the CLI, control returns to the CLI.

Once a program is debugged, recompile it without the /D switch. There is no longer any need for the symbol table, which adds substantially to the object program size. In addition, delete any extra paragraph names that you have inserted.

Program Calls under Debugger Control

In many applications, COBOL programs pass control to one another using CALL or CALL PROGRAM statements. With both statements, if the called program has been compiled for debugging, it is loaded into memory, execution is suspended, and the debugger command prompt is displayed.

Subprograms executed with CALL statements remain in the debugger across the CALL.

The debugger treats CALL PROGRAM statements as if they have the #D form. If the called program has not been compiled with the /D switch, it runs as if a standard CALL PROGRAM were issued.

For example, two programs exist, PROGA and PROGB, and PROGA issues a CALL PROGRAM to PROGB. PROGB, but not PROGA, has been compiled with the global /D switch. The command `ICDEBUG PROGA` is equivalent to

```
ICX PROGA
ICDEBUG PROGB
```

Debugger Commands

This section identifies and describes the debugger's commands and arguments.

Procedure-name can be a section name, paragraph name, or paragraph name qualified by a section name. *Id* is any data-item identifier defined in the Data Division, except screen-names and data-names for COMPUTATIONAL or INDEX items. A screen-name does not identify actual program storage, it is a temporary "window" through which data passes. Group items, qualified data-names, and subscripted data-names are all allowed.

Start or End Program Execution

RUN
: Runs the program, starting where execution was last suspended. If no RUN-type command has been issued earlier, execution starts from the beginning. The debugger executes statements until it reaches a trap, a STOP RUN statement, or a fatal error.

RUN *procedure-name*
: Runs the program starting at *procedure-name*. The debugger clears the PERFORM stack before beginning execution. Thus, this command can cause unanticipated results if used in the middle of a PERFORM sequence.

RUN START
: Runs the program, starting where execution was last suspended, and continues until it reaches the beginning of a procedure.

RUN END
: Runs the program, starting where execution was last suspended, and continues until it reaches the end of a procedure.

RUN *procedure-name START*
: Runs the program, starting where execution was last suspended, and continues until it reaches the beginning of *procedure-name*.

RUN *procedure-name* END
: Runs the program, starting where execution was last suspended and continuing until it reaches the end of *procedure-name*.

STOP
: Terminates debugger execution. Control returns to the program that called the debugger.

Suspend Debugger Execution

The trap commands differ from the run and stop commands in that the program halts at a trap point every time that procedure is reached. The trap point remains until explicitly removed by a CLEAR command.

TRAP *procedure-name*
: Sets a trap to suspend execution at the beginning of the given procedure.

TRAP *procedure-name* END
: Sets a trap at the end of the given procedure.

CLEAR
: Removes all traps.

UNCLEAR
: Cancels a CLEAR command, causing the traps to be retained. This command must be issued immediately following a CLEAR command.

CLEAR *procedure-name*

Removes traps at the beginning and end of the specified procedure. UNCLEAR does not undo this command.

LIST

Displays all procedure-names that have trap points set at their beginning.

LIST END

Displays all procedure-names that have trap points set at their end.

Display Values

DISPLAY *[id]*

Displays the value of the data-item *id*. If the id argument is omitted, the value of the most recently referenced data-item (whether named in a SET or DISPLAY command) is displayed.

(blank)

Equivalent to DISPLAY. Press NEW LINE alone. The debugger displays the value of the most recently referenced data-item.

SET *[id]* TO *lit*

Moves the value indicated by *lit* to the data-item *id*. If the id argument is omitted, the most recently referenced data-item is used. If *lit* is a nonnumeric literal, it must be enclosed in quotation marks.

Chapter 6
System Management

Certain issues pertaining to running Interactive COBOL on AOS and AOS/VS are not covered in the various manuals that explain the generation, management, and operation of the two systems. Many of the topics discussed below should be handled by your system manager or operator.

PREDITOR

The program PREDITOR lets the system manager add new users to the system and define what they may do when they sign on. Using Interactive COBOL on AOS and AOS/VS has implications for the initial program and IPC file.

Initial Program

When a user signs on, and the AOS or AOS/VS system has verified the username and password, the process is started by running an initial program. Usually this is the CLI, which allows users to run other programs, inspect files, print data, etc. Some end users, however, may find the CLI complicated. Also, giving all users access to the CLI can make security more difficult.

A developer of end-user systems may want the runtime system, ICX.PR, as the initial program for some users. In this case, the users start the runtime system as soon as their usernames and passwords are accepted. The runtime system, ICX.PR, looks for a program named Logon and runs it as its first program.

PREDITOR does not allow you to further specify an Interactive COBOL program for the runtime system to run. That is, you cannot tell PREDITOR that the initial program is, for example, :UTIL:ICOBOL:ICX.PR ACCOUNTS_PAYABLE. However, you can link Logon to ACCOUNTS_PAYABLE:

```
CREATE/LINK LOGON.(PD,DD) ACCOUNTS_PAYABLE.(PD,DD).
```

If a user is to run a variety of programs, he may use the standard Logon program or substitute his own. The runtime system then displays the Logon menu and the user can enter the name of an Interactive COBOL program. If a user only runs one Interactive COBOL program, you may rename it Logon (in the user's directory); that program then runs automatically when the user signs on.

Initial IPC File

The initial IPC file is one step removed from the initial program. It is a set of commands given to the initial program as soon as the initial program starts to run. The IPC file is typically used when the CLI is the initial program; if the initial program is ICX.PR, the contents of the initial IPC file are never used, although the file must exist. The following steps illustrate one use of an IPC file:

1. While in PREDITOR specify that the user's initial IPC file is :UDD:username:filename. (For example, filename could be RISE.CLI.)
2. Create the initial IPC file (e.g. RISE.CLI in the user's home directory).
3. Use a text editor to insert CLI commands in the initial IPC file. List them on separate lines; for example:

```
WRITE Good morning John
TIME
DATE
PAUSE 3
ICX ACCOUNTS_PAYABLE
BYE
```

When the CLI starts and this user's IPC file is run, the screen displays the message `Good morning John`, shows the time and date, pauses to allow the user to read the messages, and calls the Interactive COBOL runtime system, which runs the ACCOUNTS_PAYABLE program. After the user exits ACCOUNTS_PAYABLE, he is logged off the system.

MINISAM Lock Server

The MINISAM lock server is a separate process that the runtime system communicates with in order to synchronize access to files being shared by different users. The only interaction a programmer or user has with the MINISAM lock server is to start it and shut it down.

To start the MINISAM lock server, use the UP.ICOBOL macro in :UTIL:ICOBOL or type the following command line from the operator's console:

```
PROCESS/DEFAULT/DIR=@/NAME=MLS :UTIL:ICOBOL:MLS.PR          (AOS)
PROCESS/DEFAULT/DIR=@/NAME=MLS32 :UTIL:ICOBOL:MLS32.PR      (AOS/VS)
```

To stop the MINISAM lock server, use the DOWN.ICOBOL macro in :UTIL:ICOBOL or type the following from the operator's console:

```
CONTROL @MLS STOP                                           (AOS)
CONTROL @MLS32 STOP                                         (AOS/VS)
```

MLS will not shut down if ISAM files are being used. If the MLS is not active when the runtime system is executed, the message `ERROR: MLS IS NOT ACTIVE` is issued.

If MLS has an internal problem that causes it to abort, the following message is displayed on the operator's console:

```
MLS REV. 1.30 PANIC
[Error message text]
PC=nnnnnn AC0=nnnnnn AC1=nnnnnn AC2=nnnnnn AC3=nnnnnn
```

PC is the runtime system's program counter and the ACs are the accumulators. Under AOS a breakfile, or under AOS/VS a memory image dump, is created in the current directory. On AOS, the file is named *?pid.hh_mm_ss*.BRK. On AOS/VS, the name of the file is *?pid.hh_mm_ss*.7.MDM. *Pid* is the process ID of the runtime system, and *hh_mm_ss* is the time of the failure. The values of the PC and the accumulators are saved in the file.

These are temporary files that are deleted whenever the operating system is rebooted. Rename them so that they do not begin with a question mark to prevent their automatic deletion.

Printer Control

Requests by an Interactive COBOL program for output to a printer are handled by the printer queues. In most cases you want the printer(s) to handle the jobs of all users on a first-come-first-served basis.

All Interactive COBOL programs that need print jobs handled by the normal system print queue should use the following statement in the Environment Division:

```
SELECT filename ASSIGN TO PRINTER.
```

This statement does not mention a particular queue name, so all print requests are sent to the default queue, LPT.

Obtaining Exclusive Control

You may want a particular Interactive COBOL program to have exclusive use of a printer. For example, in printing payroll checks it is unacceptable to have other users' output printed while the printer is loaded with check forms. The printer should continue to accept print requests from all users, but delay requests until the user with exclusive control is finished.

Note: When obtaining direct access to the printer, perform forms control from your Interactive COBOL program. Do not use the Forms Control Utility (FCU) described in the AOS and AOS/VS *CLI User's Manual*. Any format specifications created by FCU are ignored when you have direct access to the printer.

To gain direct access to the printer from your program, perform the following steps:

1. Include the following statement in your Interactive COBOL program:
   ```
   SELECT filename ASSIGN TO PRINTER, "@printer-name".
   ```
   where *printer-name* is in the form LP*x* or LP*xn* (for example, LPB, or LPA1).
2. Before you run the program, execute the following command from the master console:
   ```
   CONTROL @EXEC PAUSE @printer-name
   ```
   If any file is currently printing, let it finish before issuing the following commands:
   ```
   CONTROL @EXEC STOP @printer-name
   ```
   Files to be printed can still be placed in the print queue but they are not printed until the queue is started up again.
3. Change the form in the printer to the form required.
4. Run the Interactive COBOL program to print the form.
5. Change the form in the printer back to the standard form.

6. Start the printer again by issuing the following commands from the master console:

```
CONTROL @EXEC START queue-name @printer-name
CONTROL @EXEC CONTINUE @printer-name
```

You must restart all queues assigned to the printer name. For example, you might need to issue the following commands:

```
CONTROL @EXEC START LPT @LPB
CONTROL @EXEC START BATCH_OUTPUT @LPB
CONTROL @EXEC START BATCH_LIST @LPB
```

To use the Forms Control Utility and spooling, perform the following steps:

1. Include the following statement in your Interactive COBOL program:

```
SELECT filename ASSIGN TO PRINTER, "disk-filename".
```

2. Execute FCU from the CLI to define the form. (Instructions for executing FCU are found in the AOS and AOS/VS *CLI User's Manual.*)
3. Execute the Interactive COBOL program to create the disk file.
4. Place the disk file into the print queue and designate it as a form:

```
QPRINT/FORMS=fcu-form-name disk-filename
```

5. Halt normal printing by issuing the following command from the master console:

```
CONTROL @EXEC PAUSE @printer-name
```

If a file is currently being printed, it will finish before the printer is stopped.

6. Change the form on the printer.
7. From the master console, load the FCU form file into the printer's memory, set headers and trailers off, and print the form:

```
CONTROL @EXEC FORMS @printer-name fcu-form-name
CONTROL @EXEC HEADERS @printer-name 0
CONTROL @EXEC TRAILERS @printer-name 0
CONTROL @EXEC CONTINUE @printer-name
```

8. Put the standard form back in the printer.
9. Reset the forms control, headers and trailers to the standard settings, and restart the printer by issuing the following commands from the master console:

```
CONTROL @EXEC PAUSE @printer-name
CONTROL @EXEC FORMS @printer-name
CONTROL @EXEC HEADERS @printer-name 1
CONTROL @EXEC TRAILERS @printer-name 0
CONTROL @EXEC CONTINUE @printer-name
```

Setting Up an Alternate Queue

You can set up an alternate queue from the operator's console with the following commands:

```
CONTROL @EXEC CREATE PRINT LPT_EXC
CONTROL @EXEC OPEN LPT_EXC
```

The program that spools to the alternate queue should include the following statement in the Environment Division:

```
SELECT internal-name ASSIGN TO PRINTER "@LPT_EXC".
```

All print requests from that program will go to the queue LPT_EXC.

Before printing the files that have been queued, you need to pause the printer so that anything currently printing can finish. The command is:

```
CONTROL @EXEC PAUSE @printer-name
```

Change forms at this point, if necessary. To print the files that have been queued, enter the following:

```
CONTROL @EXEC STOP queue-name
CONTROL @EXEC START LPT__EXC @printer-name
CONTROL @EXEC CONTINUE @printer-name
```

The first instruction stops the original queue. The queue still accepts requests but does not send them to the printer. All users can continue to run their programs; their print requests are merely "on hold" in the queue. Issue this instruction for each PRINT queue associated with the printer. The second and third instructions start printing in the alternate queue.

When the alternate queue should no longer process requests, enter the following commands:

```
CONTROL @EXEC PAUSE @printer-name
CONTROL @EXEC STOP LPT__EXC
CONTROL @EXEC START queue-name @printer-name
CONTROL @EXEC CONTINUE @printer-name
```

The original queue is then restarted, and all the jobs that have been collecting in it are printed. Repeat this set of commands for each queue you have stopped.

When the alternate queue contains no jobs and you no longer need it at all, issue the following commands:

```
CONTROL @EXEC STOP LPT__EXC
CONTROL @EXEC DELETE LPT__EXC
```

Batch Processing

AOS and AOS/VS provide a system-controlled batch stream that you may use for noninteractive jobs. A program to be batched cannot have operator input (ACCEPT statements); if it does, execution will fail.

The command lines to compile and execute an Interactive COBOL program are:

```
QBATCH ICOBOL source-file
QBATCH[/QOUTPUT=disk-filename] ICX prog-name
```

The output from the program goes to the system printer. If your program has DISPLAY statements, include the /QOUTPUT=*disk-filename* argument. Your program will run correctly without it, but the runtime system will send unprintable characters to the printer, and the printer will display an error message.

The batch processing capabilities of the AOS and AOS/VS systems are extensive. Consult the *CLI User's Manual* under the QBATCH, QSUBMIT, and PROCESS commands for more details.

Chapter 7
Converting between RDOS and AOS or AOS/VS

The syntax of Interactive COBOL is identical across the operating systems it runs on. Those places where an Interactive COBOL statement makes use of an operating system feature, however, can differ among operating systems. This chapter describes differences that may arise in converting a program between the RDOS and AOS, AOS/VS environments.

Moving an RDOS Environment to AOS or AOS/VS

AOS and AOS/VS do not normally recognize an RDOS Interactive COBOL program that uses device or directory specifiers for external filenames.

For example, your program uses the device specifier DZ1 and the RDOS system has two directories on that device, CUSTOMERS and PRICES. AOS and AOS/VS do not recognize a directory specifier, but they will accept the name if it is an actual directory. Within the directory where you will be running the program on AOS or AOS/VS, create a subdirectory DZ1, and within that directory create two more subdirectories, CUSTOMERS and PRICES. Put all the files that RDOS had in those directories in their counterparts on AOS or AOS/VS.

When your program references the file DZ1:PRICES:BOLTS, the AOS or AOS/VS runtime system finds the directory DZ1, the subdirectory PRICES, and the file BOLTS as if it were looking at the physical device DZ1 under RDOS.

Recompiling for Efficiency

An Interactive COBOL program compiled under RDOS cannot share memory on AOS or AOS/VS if more than one user is running that program. This is because AOS and AOS/VS share program (.PD) code, which is rounded up to the next 2-KB boundary. The AOS and AOS/VS compilers produce code in this format, but the RDOS compiler does not. If many Interactive COBOL users are running the same program on one AOS or AOS/VS system, additional memory is required when the system cannot share program code.

Therefore, if an Interactive COBOL program is to be used on AOS or AOS/VS by more than one person at a time, compile it with the AOS or AOS/VS compiler. The .PD files compiled on AOS or AOS/VS execute correctly on RDOS systems because the 2-KB rounding is not included in program space on RDOS.

Transporting Files to AOS and AOS/VS

To load files created on RDOS to AOS or AOS/VS, use the RDOS LOAD command. Load text files, such as listing files and source program files, with the /C switch. The /C switch converts the terminators for processing by the destination operating system — AOS or AOS/VS for an RDOS LOAD, and RDOS for an RDOS DUMP. Use this switch for files that will be listed or printed. Do not load program object (.PD and .DD) files, indexed or relative (.NX and .XD) files, or print-ready files with the /C switch.

Under AOS and AOS/VS, all ISAM files must have element sizes that are multiples of 4. In addition, any ISAM files opened for INPUT must end on a 2-KB boundary. Use MINISAM_CONVERT.CLI on all indexed or relative files to give them to an element size that is a multiple of 4 and a file size that is on a 2-KB boundary. The format is:

```
MINISAM_CONVERT filename
```

Suppose MAIL.SR is a source program (with object files MAIL.PD and MAIL.DD) that uses the indexed file ZIP (ZIP.NX and ZIP.XD). To move these files from RDOS to AOS, dump the programs to tape. Then use the following command to load the files onto the AOS system:

```
XEQ RDOS LOAD/V @MTA0:n MAIL.SR/C MAIL.PD MAIL.DD ZIP.NX ZIP.XD
```

MTA0 is the tape drive and *n* is the tape file to which the program and files were dumped. To convert the element and file sizes of ZIP, run MINISAM_CONVERT:

```
MINISAM_CONVERT ZIP
```

Developing RDOS Programs on AOS, AOS/VS

The AOS and AOS/VS file structure is more versatile than that of RDOS. The basic rule for developing Interactive COBOL programs for RDOS under AOS or AOS/VS is: Do not use any features of the AOS and AOS/VS file system that RDOS does not have. There are three areas where the AOS and AOS/VS file system is different from RDOS.

- AOS and AOS/VS allow more levels of directories. If you write a program under AOS or AOS/VS that uses a pathname such as PHIL:ICOBOL:PROJECT1:PROG1, it will not run under RDOS because RDOS permits only one directory level in the pathnames you specify within an Interactive COBOL program. For example, PROJECT1:PROG1 is acceptable under all systems.
- RDOS does not begin any filenames with a colon. The filename :UDD:PROG1 is not recognized by RDOS, even though it only uses one directory level. Use partial pathnames that do not begin with a colon when you use AOS or AOS/VS to write Interactive COBOL programs for RDOS.
- AOS and AOS/VS accept more filename characters and the filenames may be longer than with RDOS. Under AOS and AOS/VS, 123.?.XYZ_$$$ is a legal filename, but not under RDOS. See the *Interactive COBOL User's Guide (RDOS, DG/RDOS)* for valid RDOS filenames and follow those conventions when developing programs under AOS or AOS/VS.
- If you have written a Logon program that opens ISAM files, it should not be used under RDOS.

Recompiling to Conserve Disk Space

The AOS, AOS/VS compilers round the .PD and .DD up to 2 KB boundaries, padding with zeros. A file compiled on AOS or AOS/VS will execute correctly on RDOS; however, the padding takes up disk space.

The global /R compiler switch causes the AOS or AOS/VS compiler *not* to round the .PD up to a 2-KB shared boundary. The format is:

```
ICOBOL/R progname
```

Transporting Files to RDOS

To move files created on AOS or AOS/VS to an RDOS system, use the command RDOS DUMP. Load text files, such as listing files and source program files, with the /C switch. The /C switch converts the terminators for processing by the destination operating system — AOS or AOS/VS for an RDOS LOAD, and RDOS for an RDOS DUMP. Use this switch for files that will be listed or printed. Do *not* load program object (.PD and .DD) files, indexed or relative (.NX and .XD) files, or print-ready files with the /C switch.

Suppose MEMO.SR is a source file developed under AOS (with object files MEMO.PD and MEMO.DD) that uses the indexed file RTN (RTN.NX and RTN.XD). To move this program and related files to RDOS, use the following AOS command to dump the files in RDOS format:

```
XEQ RDOS DUMP/V @MTA0:n MEMO.SR/C MEMO.PD MEMO.DD RTN.NX RTN.XD
```

MTA0 is the tape drive and *n* is the tape file to which the program and indexed files will be dumped.

Transporting Print-Ready Files

A print-ready file is created by an Interactive COBOL program when a disk file is assigned to PRINTER, PRINTER-1, or DISPLAY. The record format of these files differs according to the operating system that created the file.

If records are written to a file with default vertical positioning (ADVANCING 1 LINE), the print-ready file has the following record format:

<CR><NEW LINE><record><CR><CR>	RDOS or DG/RDOS
<NEW LINE><record><CR>	AOS or AOS/VS

Do not use the /C switch when transporting print-ready files to or from an AOS or AOS/VS system. Table 7-1 lists the commands to use on each system in order to print print-ready files.

File Created On	To Print on AOS, AOS/VS	To Print on RDOS, DG/RDOS
AOS or AOS/VS	QPRINT or COPY	XFER (without /A)
RDOS or DG/RDOS	QPRINT or COPY	P.A.S.S. or XFER (without /A)

Table 7-1 Print Commands for Print-Ready Files

System Call Differences

A number of system calls implemented on RDOS have no effect under AOS and AOS/VS. When the AOS or AOS/VS runtime system sees one of these calls, the Exception Status code is set to 203 and control goes to the next statement. However, you may use any of them while developing under AOS or AOS/VS. The calls that are not supported under AOS and AOS/VS are ignored. When you transport the program to an RDOS system, the calls become operative.

The following list describes the calls that are not supported under AOS and AOS/VS, the reason the call is not supported, and other ways in which the function is implemented. (See chapter 2 for the available AOS and AOS/VS system calls.)

#A Under RDOS, aborts a terminal (can only be used at the master terminal). Under AOS and AOS/VS, only users with the superprocess privilege can abort other users. If you have this privilege, you can abort another terminal from the CLI.

#C Under RDOS, "physically" closes files that have been logically closed by CLOSE, STOP RUN, etc. Under AOS and AOS/VS, such physical closing is not necessary.

#F Performs a full disk initialization under RDOS. AOS and AOS/VS do not permit this operation at all. On an AOS or AOS/VS machine, you must format disks before starting.

#I Under RDOS, initializes a directory, allowing you to access the files in it. On AOS and AOS/VS, all directories can be accessed at any time.

#M Under RDOS, allows the master terminal to send messages to the other terminals. The AOS and AOS/VS CLI command SEND performs this function.

#O Under RDOS, runs a program detached from the master console; assigns the default output to an RDOS file.

#P Allows you to use the Printer Access Scheduling System (PASS). Under AOS and AOS/VS, the operating system controls all printer activity. See the section on printer control in chapter 6.

#R Related to #I. Under RDOS, releases or closes a directory after you finish using it. Under AOS and AOS/VS, you never have to initialize a directory, so there is no need to release it.

#T Under RDOS, displays the status of all the terminals under control of the runtime system. With AOS and AOS/VS, there are CLI commands to obtain the status of a single terminal or every terminal. The following command lets you see the users that are logged on and the programs (.PR files) that are running:

```
WHO/2=IGNORE <0,1,2,3,4,5,6><0,1,2,3,4,5,6,7,8,9>
```

Appendix A
Error Messages

File Status Codes

Code	Meaning
00	Successful I/O operation.
10	AT END condition.
11	During a READ NEXT or READ PREVIOUS of an indexed or relative file, another program added a record to the file. A START statement can be used to reposition the record pointer to the most recently read record.
21	RECORD KEY error. For an indexed or relative file in sequential access mode, a WRITE statement used a RECORD KEY value that was not greater than the value used in the previous WRITE.
22	INVALID KEY error. (1) An attempt has been made to write or rewrite a record that would create a duplicate primary key. (2) An attempt has been made to UNDELETE a record that has not been deleted.
23	No record exists with the specified RECORD KEY value.
24	Index structure is full. Writing a new record would necessitate creating a new index level beyond the allowable six levels.
30	Hardware error.
34	(1) No disk space is available to write a new record. (2) The control point directory size has been exceeded.
91	OPEN error: (1) An OPEN statement referenced a file that was nonexistent, already opened, had an illegal name, for which the user had an improper ACL, or that did not have an element size that was a multiple of four. (2) An OPEN INPUT statement referenced a file that did not have a byte size that was a multiple of 2048. (3) A CLOSE statement referenced a file that had not been opened. (4) The filename already existed. (5) A nondirectory argument was in the pathname. (6) A zero-length filename was specified.
92	Access mode error: (1) File not opened. (2) WRITE or DELETE attempted to file opened for input. (3) READ attempted for file opened for output. (4) OPEN attempted for file closed with lock. (5) DELETE or REWRITE statement not preceded by a READ statement.
93	(1) Cannot delete a permanent file. (2) Directory delete error.
94	(1) File cannot be accessed because it is in use. (2) Record cannot be accessed because it is locked. (3) DELETE FILE attempted for OPEN file.
96	A directory named by the program does not exist.
97	Maximum number of OPEN files exceeded.
98	Attempt to write more than the allowable 65,534 records to a relative file.
9A	File description inconsistency. Record length, key length, or key positions specified in program does not agree with data file.
9B	After a successful OPEN of an ISAM file, the runtime system has detected possible corruption in the file. The file should be closed, which will set the reliability flags and prevent further access to the file (see 9F).
9C	Index (.NX) portion of an indexed or relative file is full.
9E	The record lock limit has been exceeded.
9F	Possible corruption of an ISAM file has been detected on an attempted OPEN of the file, i.e., the reliability flags have been set.

Exception Status Codes

Code	Meaning
000	Successful CALL or CALL PROGRAM execution.
200	Called program exceeds program size limitation.
201	Called program's revision does not match that of the runtime system.
202	Code stored under called program name is not a legal COBOL program.
203	Called program not found.
206	Read access to a called program denied.
207	Called program is active.
208	Maximum number of programs in run unit.
209	Parameter count or parameter size does not match called program.
210	Out of disk (or CPD) space during a program CALL upon creation or allocation of space for system files. (If you run out of space during a file I/O operation, File Status 34 is displayed.)
211	No channels available to open called program or to open system files required to process the call.

Compiler and Compiler Command Line Messages

ACCEPT STATEMENT

Illegal syntax for an ACCEPT statement.

ACCESS MODE CLAUSE

Illegal syntax for an ACCESS MODE clause, or the ACCESS MODE is specified as RANDOM or DYNAMIC for a nondisk device.

ADD STATEMENT

Illegal syntax for an ADD statement.

ALPHABETIC OPERAND NOT PERMITTED

An alphabetic operand is specified for a NUMERIC class test.

AT END CLAUSE

Illegal syntax for an AT END clause, or an AT END clause is illegal for an operation, e.g., a random read.

AT END OR INVALID KEY REQUIRED

Statement requires either an AT END or INVALID KEY clause when no USE procedure is defined for the file.

BLANK CLAUSE

Illegal syntax for a BLANK LINE or BLANK SCREEN clause in the Screen Section.

BLANK WHEN ZERO CLAUSE

Illegal syntax for a BLANK WHEN ZERO clause, the clause was specified for a group item or a nonnumeric (edited) item, or the item's picture string contains the asterisk character.

BLOCK CONTAINS CLAUSE

Illegal syntax for a BLOCK CONTAINS clause.

CALL STATEMENT

Illegal syntax for a CALL or CALL PROGRAM statement.

CANCEL STATEMENT

Illegal syntax for a CANCEL statement.

CLOSE STATEMENT

Illegal syntax for a CLOSE statement.

COLUMN CLAUSE

Illegal syntax for a COLUMN clause in the Screen Section.

COMPILED WITH ERRORS

This statement appears at the end of compiling a program that contains errors.

COMPILER PANIC

An internal compiler error has occurred. The system creates a breakfile. Save the breakfile and contact your Data General representative.

COMPUTER-NAME

The computer name is illegal.

COMPUTE STATEMENT

Illegal syntax for a COMPUTE statement.

CONDITIONAL

Illegal syntax for a conditional statement or sentence.

COPY FILE NOT FOUND

The file named in a COPY statement is unknown to the system.

COPY NESTED

A COPY statement exists within a COPY file.

COPY STATEMENT

Illegal syntax for a COPY statement.

CURRENCY SIGN CLAUSE

Illegal syntax for a CURRENCY SIGN clause.

DATA DESCRIPTOR CLAUSE DUPLICATE

The same clause has been used more than once in describing a data-item.

DATA DIVISION OVERFLOW

Size of the Data Division file (.DD file) exceeds 65,535 bytes.

DATA-NAME

(1) A data-name contains illegal characters. (2) In a Procedure Division Using header, at least one data-name must be present. (3) In a CALL PROGRAM USING statement, at least one data-name must be present.

DATA-NAME AMBIGUOUS: data-name

The data-name requires further qualification.

DATA-NAME UNDEFINED: data-name

(1) No description for the name exists in the Data Division. (2) Linkage Section items are referenced in the Screen Section FROM, TO, and USING phrases, or as ALTERNATE RECORD KEYS in the SELECT clause, but have not been specified in the PROCEDURE DIVISION USING list. These names are all flagged following the Procedure Division Using header. (3) Data-names are referenced in the Procedure Division and defined in the Linkage Section but are not used in the Procedure Division Using header.

DATA RECORDS CLAUSE

Illegal syntax for a DATA RECORDS clause.

DECIMAL-POINT CLAUSE

Illegal syntax for the DECIMAL-POINT IS COMMA clause.

DEVICE

The device is not DISK, PRINTER, KEYBOARD, or DISPLAY in a SELECT clause.

DISPLAY STATEMENT

Illegal syntax for a DISPLAY statement.

DIVIDE STATEMENT

Illegal syntax for a DIVIDE statement.

DIVISION MISSING

An expected division header was not found.

DUPLICATE VARIABLE LINE OR COLUMN CLAUSE

You have more than one LINE clause or COLUMN clause in a Screen line.

ELSE WITHOUT IF

An ELSE clause has been encountered without a preceding IF statement.

ERROR OR LISTING FILENAME SAME AS SOURCE

The name of the error or listing file cannot be the same as the name of the source file.

EXIT STATEMENT

The EXIT sentence is not the only statement in the paragraph.

EXPONENT MUST BE AN UNSIGNED INTEGER WITH 5 OR FEWER DIGITS

The first operand following an exponentiation operator is a literal or identifier with sign or fraction, or has too many digits.

FATAL ERROR -- COMPILATION ABANDONED

The previous error is considered catastrophic and compilation cannot continue. The source file will continue to be read and listed.

FD DUPLICATE: filename

An FD has already been processed for this file.

FD ENTRY DUPLICATE

The same clause has been used more than once in describing a file.

FIELD CLAUSE MISSING

A picture string is specified for a screen field but no FROM, TO, or USING clause is present.

FIGURATIVE CONSTANT

ALL is not followed by a nonnumeric literal or figurative constant.

FILE-CONTROL CLAUSE DUPLICATE

The same clause has been used more than once in a SELECT clause.

FILE-CONTROL CLAUSES INCONSISTENT

ACCESS MODE, ORGANIZATION, and RECORD (RELATIVE) KEY clauses have been used in an illegal combination.

FILE-CONTROL ENTRY AMBIGUOUS: data-name

The FILE STATUS or KEY data-name requires further qualification.

FILE-CONTROL ENTRY SUBSCRIPTED: data-name

The FILE STATUS or KEY data-name has, or is subordinate to, an OCCURS clause.

FILE-CONTROL ENTRY UNDEFINED: data-name

The FILE STATUS or KEY data-name is not defined in the Data Division.

FILE-CONTROL PARAGRAPH MISSING

The FILE-CONTROL paragraph is required.

FILE DESCRIPTION UNDEFINED: filename

No FD was encountered for a file named in a SELECT clause.

FILE DOES NOT EXIST filename

The source file does not exist in the current directory or in those included in the searchlist. A specified indexed file does not exist or contained a filename extension.

FILE NAME

A filename contains an illegal character.

FILE NAME AMBIGUOUS

The filename in the SAME RECORD AREA clause is ambiguous, or the same filename is used in two SELECT clauses.

FILE NAME UNDEFINED: filename

No SELECT clause exists for this filename.

FILE STATUS: data-name

The FILE STATUS item must be a two-character alphanumeric item defined in the Data Division.

FILLER NOT ELEMENTARY

Filler items must not be further subdivided.

IDENTIFICATION DIVISION MISSING

The Idenitifcation Division header is required.

IDENTIFIER

An identifier is expected but not found.

IDENTIFIER OR LITERAL

An identifier or literal is expected but not found.

IF STATEMENT

Illegal syntax for an IF statement.

ILLEGAL CHARACTER

An illegal character appears in the source program; for example, a percent sign that is not in a literal string.

ILLEGAL GLOBAL SWITCH

Illegal global switch in the command line.

ILLEGAL LOCAL SWITCH

The only legal local switch is /I.

ILLEGAL SEPARATOR

Illegal separator in the command line.

ILLEGAL SWITCH COMBINATION /A /I

These two switches may not be used together in a command line.

ILLEGAL SWITCH COMBINATION /N/U

These two switches may not be used together in a command line.

ILLEGAL TERMINATOR

Illegal terminator in the command line.

IMPERATIVE STATEMENT REQUIRED

An imperative statement must appear after AT END, ON SIZE ERROR, ON ESCAPE, ON EXCEPTION, etc.

INDEX NAME

An index-name is expected but not found.

INPUT FILE IS NOT ISAM

The /I switch is in the command line but the input file is not an ISAM file.

INSPECT STATEMENT

Illegal syntax for an INSPECT statement.

INTEGER

A noninteger has been encountered in picture string parentheses, or an integer in an ADVANCING clause has a value greater than 80.

INVALID KEY CLAUSE

Illegal syntax for an INVALID KEY clause.

I-O-CONTROL CLAUSE

Illegal syntax for an I-O-CONTROL clause.

I/O STATEMENT ILLEGAL FOR ACCESS MODE

Inconsistency in an I/O statement, e.g., indexed READ to a file opened in sequential mode.

I/O STATEMENT ILLEGAL FOR DEVICE

Inconsistency in an I/O statement, e.g., indexed READ to a file opened in sequential mode.

I/O STATEMENT ILLEGAL FOR ORGANIZATION

Inconsistency in an I/O statement, e.g., OPEN EXTEND for a relative file.

ISAM RECORD SIZE EXCEEDS 132 CHARACTERS

The ISAM record size is too large.

ISAM SOURCE FILE HAS INCOMPATIBLE REVISION NUMBER

The ISAM source file revision number is incompatible.

JUSTIFIED CLAUSE

JUSTIFIED has been specified for a group, numeric, or numeric-edited item.

KEY CLAUSE

Illegal syntax for a KEY clause in a START statement.

LABEL RECORD CLAUSE

Illegal syntax for a LABEL RECORDS clause.

LABEL RECORDS CLAUSE MISSING

A LABEL RECORDS clause is required for every FD.

LENGTH NOT = 2

The FILE STATUS item must be PIC XX.

LEVEL 88 ERROR

A level 88 entry contains a syntax error.

LEVEL 88 THROUGH CLAUSE

A level 88 entry contains a THROUGH clause but the second value is missing.

LEVEL NUMBER

(1) A level number is expected but not found. (2) All data-items referenced in a Procedure Division Using header must have a level number of 77 or 01. (3) All data-items used in a CALL PROGRAM USING statement must have a level number of 77 or 01.

LEVEL 77 NOT ELEMENTARY

A level 77 entry may not be subdivided.

LINE CLAUSE

Illegal syntax for a LINE clause in the Screen Section.

LINE EXCEEDS 132 CHARACTERS

The source line read had 133 characters without a terminator.

LINE/COLUMN LIMIT EXCEEDED

Data has been defined past line or column 255 in the Screen Section.

LINKAGE DATA BLOCK TOO LARGE

The Linkage Section exceeds the maximum size.

LITERAL CATEGORY

The category of the literal in a VALUE clause does not match the category of the data-item.

LITERAL EXCEEDS ITEM SIZE

The size of the literal in a VALUE clause exceeds the size of the data-item.

LITERAL IS SIGNED

The numeric literal in a VALUE clause is signed, but the data-item specified no operational sign.

MARGIN A MUST BE BLANK

On a continuation line, area A must be blank.

MAY NOT BE USED IN A "CALL PROGRAM": data-name

The named item may not be passed to a called program.

MEMORY CAPACITY EXCEEDED

The symbol table (user defined names and literals) has exceeded the available memory.

MISSING "=" SIGN

Illegal syntax for a COMPUTE statement.

MISSING SECTION

A required section, such as the Configuration Section, is missing in the source program.

MOVE STATEMENT

Illegal syntax for a MOVE statement.

MOVE UNDEFINED FOR OPERANDS

Category of operands is illegal, e.g., moving a numeric edited item to a numeric item.

MULTIPLY STATEMENT

Illegal syntax for a MULTIPLY statement.

NAME DUPLICATE: name

(1) A user-defined word has already been defined as the same type (filename, data-name, etc.) and cannot be distinguished by qualification. (2) A data-name has been used more than once in a Procedure Division Using header.

NO data-item DEFINED IN LINKAGE SECTION

(1) When a Linkage Section occurs in a program, at least one data-item must be defined within the section. (2) When a Procedure Division Using header occurs in a program, a Linkage Section and at least one subordinate item must be defined in the calling program.

NON-NUMERIC LITERAL LESS THAN 1 OR GREATER THAN 132 CHARACTERS

Nonnumeric literals must have at least one character but not more than 132.

NOT A GROUP ITEM

Incorrect use of a CORRESPONDING verb. One of the operands is not a group item.

NOT A LINKAGE SECTION ITEM

In a Procedure Division Using header, all items in the list must be defined in the Linkage Section of the calling program.

NOT ALPHANUMERIC

Identifier must be alphanumeric.

NOT AN ELEMENTARY ITEM

Incorrect use of a CORRESPONDING verb. One of the corresponding elements that is required to be elementary is not.

NUMERIC (EDITED) ITEM EXCEEDS 18 DIGITS

Numeric items must not be longer than 18 digits.

NUMERIC LITERAL EXCEEDS 18 DIGITS

Numeric literals must not be longer than 18 digits.

NUMERIC OPERAND NOT PERMITTED

Numeric operands are illegal for this statement.

NUMERIC OPERAND REQUIRED

A numeric operand was expected but not found.

NUMERIC OR NUMERIC EDITED OPERAND REQUIRED

A numeric or numeric edited item was expected but not found.

OCCURS CLAUSE

Illegal syntax for an OCCURS clause.

OCCURS DEPTH

OCCURS clauses are only permitted to a depth of 3.

OCCURS NOT PERMITTED AT LEVEL 01 OR 77

OCCURS may only be specified from level 02 to 49.

ON EXCEPTION CLAUSE

Illegal syntax for an ON EXCEPTION clause.

OPEN STATEMENT

Illegal syntax for an OPEN statement.

OPERAND IS USAGE INDEX

One of the operands in a CORRESPONDING verb is an index item; this is not permitted.

OPERAND MUST NOT BE SIGNED NUMERIC OR COMP

A signed or computational item cannot be used in this statement.

OPERAND NOT INTEGER

An integer was expected but not found.

OPERAND NOT LENGTH 1

The operand must be one character long.

OPERAND NOT USAGE DISPLAY: data-name

A USAGE DISPLAY operand is required by this statement.

OPERATIONAL SIGN MISSING

A SIGN clause has been specified, but the picture string does not contain an S character.

ORGANIZATION CLAUSE

Illegal syntax for an ORGANIZATION clause.

PERFORM STATEMENT

Illegal syntax for a PERFORM statement.

PERIOD MISSING

A period is required.

PICTURE MISSING

FROM, TO, or USING clauses have been specified for a screen item, but no PICTURE is specified.

PICTURE STRING

Illegal syntax for a picture string.

PREVIOUS ITEM WAS ELEMENTARY

The current data or screen item has a higher level number than the previous one, yet the previous item was not a group item.

PREVIOUS ITEM WAS GROUP

The current data or screen item has a level number equal or lower than the previous one, yet the previous item was not elementary.

PROCEDURE DIVISION MISSING

The Procedure Division header is required.

PROCEDURE DIVISION NOT SECTIONED

The Procedure Division did not begin with a section name, but one is encountered later.

PROCEDURE DIVISION OVERFLOW

The size of the Procedure Division (.PD) file exceeds 65,535 bytes.

PROCEDURE NAME

The procedure-name contains an illegal character.

PROCEDURE NAME AMBIGUOUS: procedure-name

Reference is made to an unqualified paragraph name, but none exists in the current section and two or more exist in other sections.

PROCEDURE NAME DUPLICATE: procedure-name

Two procedure-names exist that cannot be differentiated by qualification.

PROCEDURE NAME MISSING

The Procedure Division must start with a procedure-name (after the Declaratives, if any). A section name must be followed by a paragraph name.

PROCEDURE NAME UNDEFINED: procedure-name

The procedure-name referred to does not exist.

PROGRAM-ID PARAGRAPH MISSING

The PROGRAM-ID paragraph is required.

QUOTE MISSING

A line ends with an open nonnumeric literal, and the next line is not a continuation line or does not begin with a quote in area B.

RECORD CONTAINS CLAUSE

Illegal syntax for a RECORD CONTAINS clause.

RECORD KEY CLAUSE

Alternate key clause is used illegally, or more than four alternates have been specified.

RECORD KEY EXCEEDS MAXIMUM LENGTH

The record key cannot exceed 100 characters.

RECORD KEY MUST BE ALPHANUMERIC: data-name

The RECORD KEY of an indexed file must be alphanumeric.

RECORD KEY MUST BE DEFINED IN FD: data-name

The record key of an indexed file must be defined in one of the record descriptions of that file.

RECORD NAME

The data-name referred to is not a record name, which is required by this statement.

RECORD SIZE EXCEEDS DEVICE LIMITATION

Files assigned to PRINTER, KEYBOARD, or DISPLAY may have a maximum record size of 132 characters. Indexed or relative files may have a maximum record size of 4096 characters.

RECORD SIZE OUTSIDE LIMITS OF CONTAINS CLAUSE

The actual record size exceeds the maximum specified in the RECORD CONTAINS clause or is less than the minimum.

RECORDING MODE

The RECORDING MODE may be specified only for disk files; the VARIABLE clause is only permitted for sequential disk files.

REDEFINED AREA SIZE

The size of the redefining area must equal the size of the redefined area, except at level 01.

REDEFINES NOT PERMITTED: data-name

The data-name being redefined either has a REDEFINES clause itself, has an OCCURS clause, or is subordinate to an item containing a REDEFINES or OCCURS clause.

REDEFINES NOT PERMITTED AT THIS LEVEL

REDEFINES is not allowed at level 01 in the File Section.

REFERENCE LIMIT EXCEEDED

More than 2294 procedure or data references have been made. You probably will not exceed this number, unless you have compiled your program with the /I switch, in which case you are limited to 764 references. Most likely, you have exceeded the data reference limit. To reduce this number,

- Use tables whenever possible (put switches in a table, for example).
- Make any numeric literals consistent — for example, the numeric literals 1, 01, and 001 have the same values but create three different data references.
- If your program contains long nonnumeric literals, use a continuation line and one data-name rather than multiple data-names.
- Structure your program with CALL statements.

RELATIONAL UNDEFINED FOR OPERANDS

The relational condition is not permitted for certain categories of operands.

RELATIVE KEY DEFINED IN FD: data-name

The RELATIVE KEY data-name must not be defined within any of the record descriptors of the file.

RELATIVE KEY MUST BE PIC 9(4) to 9(18) COMP: data-name

Relative keys must be defined as PIC 9(4) to 9(18) COMP.

REWRITE STATEMENT

Illegal syntax for a REWRITE statement.

SCREEN DESCRIPTOR CLAUSE DUPLICATE

The same clause has been used twice in one data-item's description.

SCREEN HAS NO INPUT FIELDS

A screen-name appears in an ACCEPT statement, but no input fields are defined in it.

SCREEN NAME MISSING

A level 01 screen item must define a screen-name.

SCREEN NOT PERMITTED

A screen-name may not be used as the identifier in a statement of the form ACCEPT id FROM. . . .

SECTION MISSING

A section header was expected but not found.

SEPARATOR

Separators must be followed by a space.

SET STATEMENT

Illegal syntax for a SET statement.

SIGN CLAUSE

Illegal syntax for a SIGN clause, or conflict with a group SIGN clause.

SIZE CLAUSE DUPLICATE

There is a duplicate in the SIZE clause.

SIZE ERROR CLAUSE

Illegal syntax for a SIZE ERROR clause.

START STATEMENT

Illegal syntax for a START statement.

STOP STATEMENT

Illegal syntax for a STOP statement.

SUBSCRIPT

All subscript data-items must be integer.

SUBSCRIPT COUNT

The number of subscript levels in the table and the number of levels in the statement referencing the table must be the same.

SUBSCRIPT IS NOT INTEGER

All subscript data-items must be integer.

SUBSCRIPT NOT PERMITTED: data-name

This item neither contains nor is subordinate to an item with an OCCURS clause.

SUBSCRIPT REQUIRED: data-name

The referenced data-name must have a subscript to indicate which occurrence of the item is to be accessed.

SUBTRACT STATEMENT

Illegal syntax for a SUBTRACT statement.

SWITCH LITERAL

Illegal syntax for a switch literal, or the literal is not A to Z.

UNMATCHED PARENTHESIS

Parentheses must appear in paired sets.

UNRECOGNIZABLE WORD

The word separator or the terminator is not legal in the current context of the program.

UNSUPPORTED FEATURE

You have attempted to use an Interactive COBOL 1.30 enhancement when compiling with the /I switch.

USAGE CLAUSE

Illegal syntax for a USAGE clause or conflict with a group USAGE clause.

USE PROCEDURE DUPLICATE

A USE procedure has already been defined for this file or class of files.

USE STATEMENT

Illegal syntax for a USE statement.

VALUE CLAUSE

Illegal syntax for a VALUE clause.

VALUE NOT PERMITTED IN FILE SECTION

A VALUE clause cannot be used in the File Section.

VALUE NOT PERMITTED IN LINKAGE SECTION

A VALUE clause must not be used for any Linkage Section data-item.

VALUE NOT PERMITTED IN OCCURS

A VALUE clause must not be used in an item with an OCCURS clause or an item subordinate to an OCCURS clause.

VALUE NOT PERMITTED IN REDEFINES

A VALUE clause must not be used in an item with a REDEFINES clause or an item subordinate to a REDEFINES clause.

VALUE SPECIFIED FOR GROUP

A VALUE clause has already appeared for an item to which this is subordinate.

WORD DUPLICATE: name

This user-defined word has already been defined as another type (i.e., filename, data-name, etc.). Both definitions are accepted and correct usage is determined by context.

WORD EXCEEDS 30 CHARACTERS

A programmer-defined word exceeds 30 characters. The word is truncated on the right to 30 characters.

WRITE STATEMENT

Illegal syntax for a WRITE statement.

WRONG NUMBER OF ARGUMENTS SPECIFIED

No source file is specified, more than one source file is specified, or other syntactical errors have been made in the compiler command line.

Data Validation Error Messages

Interactive COBOL provides data validation for interactive screen management. If the data entry does not match the PICTURE for the field, (1) an error message is displayed at the bottom of the screen, (2) the cursor is positioned to the first invalid character of the field, and (3) the system waits for reentry of the data. Below is a list of the data validation error messages and an explanation of each.

Character Must Be Alphabetic

The permissible characters are the uppercase letters A-Z and the blank.

Character Must Be Alphanumeric

An alphanumeric field can contain only graphic (noncontrol) characters.

Character Must Be Numeric

Permissible characters in a numeric field are digits, a positive or negative sign, and the decimal point.

Data Entry Is Required

At least one character must be entered.

Field Does Not Permit A Decimal Point

This field must contain an integer.

Field Does Not Permit A Sign

The PICTURE clause does not allow a sign.

Full Field Is Required

A character must be entered in each character position of the field.

Illegal Embedded Blanks

Blanks preceded and followed by other characters are not permitted in a numeric field.

No Digits Entered

The field is numeric and cannot be null.

Sign Must Be Leftmost Character

The sign must be in the first character position of the field.

Sign Must Be Rightmost Character

The sign must be in the last character position of the field.

Too Many Decimal Places Entered

The number of digits to the right of the decimal point exceeds the number specified in the PICTURE clause.

Too Many Decimal Points

Only one decimal point is permitted in a numeric field.

Too Many Integer Places Entered

The number of digits to the left of the decimal point exceeds the number specified in the PICTURE clause.

Too Many Signs Entered

Only one sign is allowed in a numeric field.

Debugger Error Messages

If the programmer enters an unacceptable command while using the Interactive COBOL debugger, one of the following messages may appear:

Transmission Error, Reenter Last Character

If the terminal has a dial-up connection, the character may not have been successfully transmitted over the line. If the terminal has a local connection, the BREAK key may have been pressed or the setting of the terminal's parity switch may be incorrect.

Illegal Character In Numeric-field

An illegal character has been specified in the SET command.

Illegal Command

The syntax or spelling of a command is unacceptable.

Subscript Error

A subscript-related error has occurred with SET or DISPLAY.

Undefined Name

An unrecognized name is specified in the command.

Wrong Data Type

The data type specified with SET is incorrect.

Runtime Error Messages

This section lists and defines error messages that can appear because of problems in starting the runtime system, a failure in a system call, a fatal error in an Interactive COBOL program, or a failure in the runtime system itself.

Starting the Runtime System

Argument Length > Item Length

You have attempted to pass arguments from the CLI to an Interactive COBOL program, and arguments are too long for the corresponding PICTUREs in the Linkage Section.

Insufficient Disk Space for VM File

Delete files that are not essential.

Minisam error: Incompatible Revision of MLS Active
Minisam error: Incompatible Revision of MLS32 Active

Your revisions of ICX and MLS are incompatible.

Minisam error: MLS Is Not Active
Minisam error: MLS32 Is Not Active

The MINISAM server process has not been initiated.

Not A Legal COBOL Program File

The file identified by the program name is not a valid program.

Program Not Found

No file having the program name has been located.

Program Too Large

The specified program exceeds the program size limit.

Read Access To Program Denied

The access control list (ACL) for the .PD or .DD file prevents the program from being executed.

Revision Incompatibility

The program was compiled with a noncurrent compiler. Recompile the program with a current compiler.

Wrong Number of Arguments

The number of arguments in the command line exceeds the number of items in the Linkage Section, or you have attempted to pass arguments to a COBOL program that does not have a Linkage Section.

System Failure

If the runtime system fails, the following message is displayed:

Runtime System Panic

The runtime system closes all files and creates in the current directory a breakfile that contains an image from memory at the time of the failure.

A possible source of the failure is that the executing program was compiled with errors. In this case, correct the program and delete the breakfile. Otherwise, save the breakfile, record the error message, and contact your DG representative. Rename the breakfile so that it does not begin with a question mark or it will be deleted the next time the operating system is booted.

Fatal Program Error

In the event of a fatal error in the execution of a COBOL program, the runtime system displays an error message on the bottom line of the terminal screen and halts program execution. Error messages have the form:

ERROR: error text COBOL PC = relative-procedure-address

The relative procedure address is a five-digit number that the Interactive COBOL compiler inserts in an output listing in place of a procedure name's line number. It gives the approximate location of the statement that caused the error. The text of the runtime system error messages and an explanation of each are given below.

Fatal COBOL Program I/O Error. COBOL PC =

An I/O error for which no USE procedure applies has occurred.

Index Register Overflow. COBOL PC =

An index or subscript is negative or exceeds 65,535.

Invalid Operation For Batch Mode. COBOL PC =

A program that requires interactive input is being run in batch mode.

PERFORM *N* Times, *N* Too Large. COBOL PC =

The value of *N* exceeds 32,768.

PERFORM Stack Overflow. COBOL PC =

Thirty active PERFORM statements are permitted. The number of active PERFORM statements has exceeded this limit.

Program Stopped By Console Interrupt. COBOL PC =

You have pressed CRTL-C and CTRL-A, CTRL-B, or CTRL-E during execution.

Subscript Out Of Range. COBOL PC =

An index or subscript is zero or greater than the maximum occurrence value of the item.

Undefined Procedure. COBOL PC =

An attempt has been made to execute an undefined procedure.

Related Documents

Interactive COBOL Documents

Interactive COBOL Programmer's Reference **093-705013**

Provides the experienced programmer with information required to write Interactive COBOL programs. The Identification, Environment, Data, and Procedure divisions are explained in detail, as well as the Screen Section, an Interactive COBOL enhancement. A syntax summary section provides a quick reference.

Interactive COBOL Utilities (AOS, AOS/VS) **069-705021**

Describe the Interactive COBOL utilities on your operating system. Summarizes the uses and contexts of the utilities, and includes an alphabetical reference that provides detailed operating instructions and examples.

ICEDIT: Interactive COBOL Editor **055-004**

Explains the Interactive COBOL text editor used to write Interactive COBOL source code and documentation. It describes how to enter and execute ICEDIT commands that create, modify, and delete source code. An alphabetized command reference and command summary table are provided.

SCREEN: Screen Format Editor **055-006**

Explains the SCREEN program, a special purpose editor for designing, coding, and displaying screen formats. The manual describes how the programmer can compose a screen image by typing in literal and data fields as they will appear to the program user. The Interactive COBOL source code for this image is generated automatically.

CRT/EDIT: Display Terminal Text Editor **055-000005**

Describes the use and operations of CRT/EDIT, a string-oriented editor designed for creating, modifying, and maintaining programs. It produces source program files that can be submitted to the Interactive COBOL compiler. The editor may also be used to produce prose text. The manual provides an overview of the editor and command reference sections that describe basic and advanced commands.

AOS and AOS/VS Documents

Learning to Use Your Advanced Operating System (AOS) **069-000018**

Summarizes AOS system utilities and leads the reader through practice sessions. It explains the steps required to run FORTRAN, COBOL, and assembly language programs.

Command Line Interpreter User's Manual (AOS and AOS/VS) **093-000122**

Describes the command line interpreter (CLI), which is the primary interface to the system for the system manager, operator, and most users. The manual explains how users can run utilities, execute programs, maintain files, and build command "macros" by means of the CLI.

AOS Programmer's Manual **093-000120**

Serves as a primary reference for assembly language programmers. It describes in detail the external features of AOS, including the AOS system calls and information required for an in-depth understanding of the operating system. Other topics included are process concepts, memory management, file I/O, and multitasking.

Learning to Use Your AOS/VS System **069-000031**

Serves as a basic introduction to the Advanced Operating System/Virtual Storage for programmers and nonprogrammers. It summarizes the system utilities and gives an overview of AOS/VS products. It also leads the reader through practice sessions with editors and illustrates the creation and execution of programs written in FORTRAN, COBOL, BASIC, and assembly language.

How to Generate and Run AOS/VS on Your ECLIPSE(R) MV/Family Computer **093-000243**

Explains how to start up, run, and shut down an AOS/VS system. Describes how to buld an AOS/VS system tailored to a user's hardware and software.

AOS/VS Programmer's Manual
Vol 1: System Concepts **093-000355**
Vol 2: System Calls **093-000241**

Serves as a primary reference for assembly language programmers. It describes the external features of AOS/VS, including system calls and information required for understanding the operating system. Other topics included are virtual memory concepts, interprocess communication, file structure and maintenance, file I/O, multitasking, and binary synchronous communication.

SGU User's Guide **093-000305**

Explains how to operate SGU, the Screen Generator Utility, which helps COBOL programmers design screen menus and formats. The manual describes how SGU creates files that contain data structures to be included in a COBOL program's Screen Section.

Using AOS on DESKTOP GENERATION Systems **069-000058**

Explains how to install and operate AOS software on your system.

Index

D

E

F

I

K

L

S

T

U

V

W

X

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____
_____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. PRICES
Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. PAYMENT
Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. SHIPMENT
Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. TERM
Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. DISCLAIMER OF WARRANTY
DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. LIMITATIONS OF LIABILITY
IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

DataGeneral

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

Data General Corporation, Westboro, MA 01580

069-705015-02