DataGeneral

Software Documentation

Interactive COBOL User's Guide (RDOS, DG/RDOS)

Interactive COBOL User's Guide (RDOS, DG/RDOS)

069-705014-02

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 (RDOS, DG/RDOS)
Ordering Number 069-705014
Revision 02, October 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 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 can be used to call an assembly language subroutine.

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 (AOS and AOS/VS); does round .PD file to a 2 KB boundary (RDOS, DG./RDOS).

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

Runtime Enhancements

#N system call to rename a file

#O system call runs a detached job.

#A system call terminates an ACCEPT statement.

Display size is determined at execution time with the DEFLINES utility.

Internal computation registers now support 19 digits of accuracy.

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

Local /M switch assigns the master console to a QTY line.

Global /C and /D switches allow CLI mode of execution.

Global /S switch enables spooling when runtime system is executing.

Global /P switch disables PASS.

Table of Contents

Preface

Chapter 1 The Interactive COBOL File System

1-1 RDOS and DG/RDOS File Structure
1-1 Disk Unit Names
1-1 Filenames
1-2 Directories
1-2 Primary Partitions
1-3 Secondary Partitions
1-4 Subdirectories
1-4 The Working Directory
1-5 Directory Initialization
1-6 Access to Files Outside the Working Directory
1-7 Links
1-7 Equivalences
1-7 File Access Restrictions
1-8 Assigning Data Files
1-9 Assigning Files to Disk Storage
1-10 Assigning Files to Hardware Devices
1-10 Direct to System Printer
1-10 Direct to Auxiliary Printer
1-11 Indirect Spooling to the PASS Queue
1-11 Assigning Files to Magnetic Tape
1-11 Assigning Files to Display Terminals
1-12 Disk Storage Allocation
1-12 Random Files
1-12 Contiguous Files
1-12 Selecting Random or Contiguous Allocation
1-13 Exclusive and Nonexclusive File Usage

Chapter 2 Interprogram Communications

2-1 System Calls
2-2 Abort Program: #A
2-2 Physically Close Files: #C
2-3 Debug a Program: #D
2-3 Fully Initialize a Directory or Tape Drive: #F
2-3 Hang Up the Terminal: #H

2-3 Initialize a Directory or Tape Drive: #I
2-3 Call Logon, Leaving the Terminal Inactive: #L
2-4 Message Broadcast: #M
2-4 Rename a File: #N
2-4 Run a Detached Program: #O
2-4 Use PASS: #P
2-4 Release a Directory or Tape Drive: #R
2-4 Stop Runtime System Execution: #S
2-5 Terminal Status: #T
2-5 Wait for a Specified Time: #W
2-5 Calls from the Logon Menu
2-5 Calls from Logon Option R
2-5 Calls from Inside a COBOL Program
2-6 Errors in System Calls
2-6 Assembly Language Calls
2-6 Calling Conventions
2-7 Restrictions
2-8 Support Routines
2-8 ?LFON: Turn LEF Mode On
2-8 ?LFOF: Turn LEF Mode Off
2-8 ?PSEM: Lock a Semaphore
2-8 ?VSEM: Unlock a Semaphore
2-9 ?GTBF: Get a Buffer for the Task
2-9 Building the Runtime System
2-11 Example

Chapter 3 The Interactive COBOL Runtime System

3-1 Runtime System Files
3-2 Customizing the Runtime System
3-2 Before Starting the Runtime System
3-2 FLEXSTATS
3-3 Terminal Line Characteristics
3-3 Channel Limits
3-3 Memory Restrictions
3-4 Starting the Runtime System
3-4 CLI Mode
3-5 Logon Mode
3-5 Executing COBOL Programs from Logon
3-6 The Logon Program
3-6 Ending Program Execution
3-7 Status of Terminals
3-7 Master Terminal
3-7 COBOL Program Interrupts

3-7 Terminating the Runtime System
3-8 Runtime System Failure
3-8 Operation of the PASS Queue
3-8 Assigning Files to the PASS Queue
3-9 PASS Control
3-9 Foreground Termination
3-10 Interactive Data Entry
3-10 Input Field Editing
3-10 The Up-Arrow Key
3-11 Input Field Termination
3-11 The ESC key
3-11 Function Keys
3-11 Single-Field Screens
3-11 Implementing the Function Keys
3-13 Field Validation
3-13 File Status
3-13 Data-Items Maintained by the Runtime System
3-14 Exception Status

Chapter 4 The Interactive COBOL Compiler

4-1 The Command Line
4-1 Global Switches
4-2 Local Switches
4-2 Source Listing
4-3 Warning Messages
4-3 Statistics
4-4 Compiler Error Messages

Chapter 5 The Interactive COBOL Debugger

5-1 Starting the Debugger
5-2 Using the Debugger
5-2 Terminating the Debugger
5-2 Program Calls under Debugger Control
5-3 Debugger Commands
5-3 Start or End Program Execution
5-4 Suspend Debugger Execution
5-4 Display Values

Appendix A Error Messages

A-1 File Status Codes
A-2 Exception Status Codes

A-2 Compiler and Compiler Command Line Messages
A-15 Data Validation Error Messages
A-16 Debugger Error Messages
A-16 Runtime Error Messages
A-17 Starting the Runtime System
A-18 System Failure
A-18 Fatal Program Error

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

The manual is written for the COBOL programmer who is familiar with the particular operating system being used. The programmer who is not familiar with RDOS should first consult the documentation related to this system. (See "Related Documents" at the end of this manual.) This manual is a companion to the *Interactive COBOL Programmer's Reference*. It describes the Interactive COBOL's relationship with Data General's Real-Time Disk Operating System (RDOS) and DESKTOP GENERATION Real-Time Disk Operating System (DG/RDOS).

Organization

The manual is divided into five 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 and calls to assembly language subroutines.

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.

Appendix A lists File Status codes, Exception Status codes, and compiler, data validation, 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 illustrates 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.
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.
|	Vertical bars in the margin highlight technical changes made since the last revision of this document.

Chapter 1
The Interactive COBOL File System

RDOS and DG/RDOS File Structure

RDOS and DG/RDOS store programs and data in files that are grouped into directories. In an Interactive COBOL program, an RDOS or DG/RDOS filename can be used in several ways:

- To identify a data file. The SELECT entry identifies data files to be used by a program.
- To identify a program. The CALL and CALL PROGRAM statements specify the filename of an object program to be called.
- To identify a directory. The initialize (#I), fully initialize (#F), and release (#R) system calls require you to specify the name of a directory or disk device.

In each case, the filename can include a prefix that indicates the directory in which the file is stored. Directories and prefixes are described below. The following sections explain RDOS and DG/RDOS file structures from the viewpoint of the Interactive COBOL programmer. For a complete description, see the *RDOS/DOS Command Line Interpreter,* 069-400015.

Disk Unit Names

Your system can have several physical disk units on line concurrently. Each unit is identified by a three-character mnemonic code, with one exception: the fixed and removable platters of a cartridge disk drive are considered two different disk units. The fixed platter's code is the same as the removable platter's, but with a fourth character, F, appended.

Each type of disk drive supported by RDOS or DG/RDOS has its own mnemonic code, for example:

DZ0	Disk pack drive
DP1F	Fixed platter of a cartridge disk drive
DE0	Fixed-media (Winchester-style) disk drive
DJ0	5 1/4-inch diskette drive (DG/RDOS)

The code refers to the disk unit itself, not to the storage medium. Thus, a removable pack must be accessed by a different code if it is moved from one drive to another.

Filenames

Your system stores each file under a name in the form *filename.xx*. The filename consists of one to ten characters from the set A-Z, 0-9, and $. The period and *xx* form an optional extension consisting of one or two characters from the filename character set. Filenames with this form are sometimes called *simple filenames* because no directory prefix precedes them.

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
.MC	System macro files
.SR	COBOL source files (CRT format)
.SV	Save files
.QK	Compiler statistics file (ICEDIT)
.OL	Overlay files
.CO	COBOL source files (card format)
.TX	Text files (ICEDIT)
.LS	Listing files
.DL	Delete files (ICEDIT)
.CU	Cut files (ICEDIT)
.SS	SCREEN source file
.AX, .SX, .DX	SCREEN descriptor files
.LJ	LJE command file
.RB	Relocatable binary file
.DR	Partitions and subdirectories
.VM	Virtual memory file

Table 1-1 System Filename Extensions

If you omit the filename extension, you may also omit the period. For instance, the system sees ACCOUNT and ACCOUNT. as identical filenames. Particular programs, however, may distinguish between them. Typically, this occurs in the handling of default filename extensions. Table 1-1 lists the system extensions.

Directories

On each disk, the operating system stores and retrieves data records from files. To allow several users to coexist conveniently, organize the files into directories. The files that reside in one directory are logically distinct from those in another. Thus, the same filename can be used for two files if they reside in different directories. RDOS and DG/RDOS support two types of directories: partitions and subdirectories.

Primary Partitions

On a disk whose only directory structure is the one that was created during disk initialization (DKINIT utility), all files are stored in a directory called the *primary partition.* The disk's mnemonic code is also the name of the primary partition (see Figure 1-1).

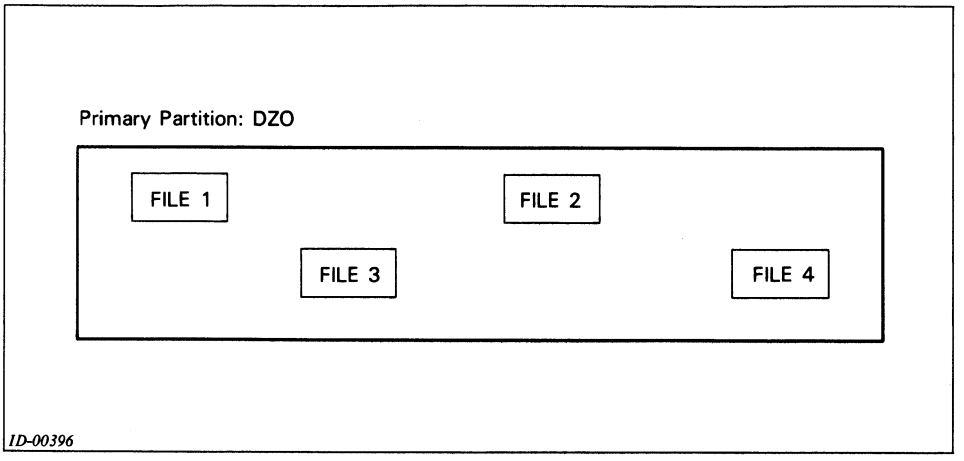


Figure 1-1 *Primary Partition*

In Figure 1-1 all four files reside in the primary partition, DZ0. In program code or in a data-entry field, a *directory prefix* indicates the location of a file. Use the following format to identify a file in the primary partition, where the partition name is the device name:

primary-partition-name:simple-filename

Thus, the four files in Figure 1-1 can be specified as:

```
DZO:FILE1, DZO:FILE2, DZO:FILE3, DZO:FILE4
```

You can omit the directory prefix to identify files (see "The Working Directory," below).

Secondary Partitions

You can set aside one or more fixed-size portions of the primary partition as *secondary partitions*. Create a secondary partition with the CLI command CPART, assigning it a certain size and a root filename; the operating system automatically adds the .DR extension to the partition name.

Conceptually, a secondary partition is a certain amount of file storage space (see Figure 1-2). To the operating system, however, a secondary partition is a fixed-size file that resides in the primary partition and contains other files. The LIST/E command flags secondary partition names with the file characteristic code Y, which indicates that the file is a directory.

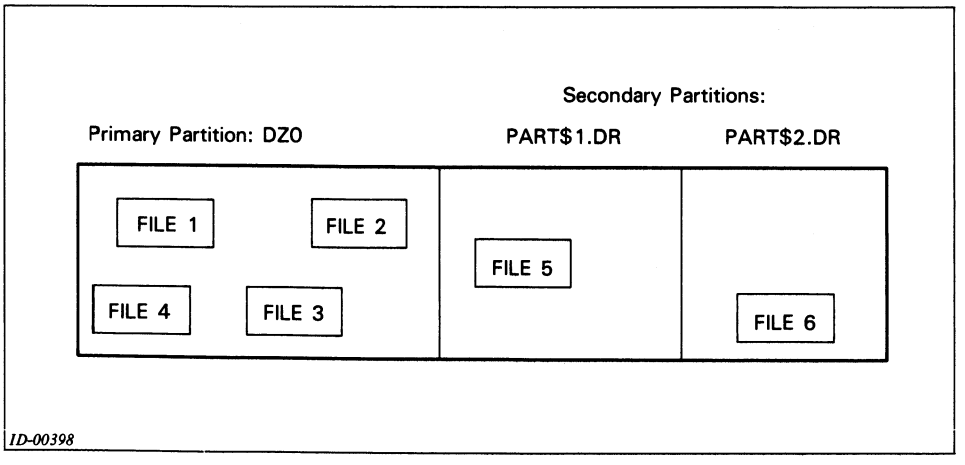


Figure 1-2 *Secondary Partitions*

Files that reside in a secondary partition are identified like those in the primary partition. The operating system does not assign a .DR extension to these files.

```
secondary-partition-name:simple-filename
```

The files illustrated in Figure 1-2 can be specified as:

DZ0:FILE1	DZ0:FILE4
DZ0:FILE2	PART$1:FILE5
DZ0:FILE3	PART$2:FILE6

Subdirectories

Within any primary or secondary partition, create subgroupings of files called subdirectories with the CLI command CDIR. As with secondary partitions, assign a root filename to each subdirectory, to which the operating system automatically adds the extension .DR. Unlike a secondary partition, a subdirectory does not have a fixed size but allocates space from its partition. The size of a subdirectory varies, depending on the number and size of the files that currently reside in it. RDOS and DG/RDOS support one subdirectory level: a subdirectory may not be created inside another subdirectory.

Files that reside in subdirectories and files that reside in partitions are identified in a similar manner; that is, by using a directory prefix without the .DR extension:

```
subdirectory-name:simple-filename
```

Optionally, a file in a subdirectory may be identified with both its partition and subdirectory names:

```
partition-name:subdirectory-name:simple-filename
```

Only the subdirectory itself must be specified. In some cases, not even the subdirectory must be specified (see "The Working Directory" below).

A subdirectory is implemented as a file that resides in a partition and contains a list of filenames (see Figure 1-3). The LIST/E command flags subdirectories with the file characteristic code Y, which indicates that the file is a directory.

Files FILEA and FILEB in Figure 1-3 can be identified as:

SUB$1:FILEA (or DZ0:SUB$1:FILEA)
SUB$2:FILEB (or PART$1:SUB$2:FILEB)

The Working Directory

At any moment, you are in a particular directory, called the current, or working, directory. The CLI command DIR changes the working directory; COBOL programs cannot change it. The directory from which you invoke the Interactive COBOL runtime system becomes the working directory for all COBOL programs. In the COBOL environment, RDOS and DG/RDOS files are referenced as follows:

- If the file is located in the working directory, a program need only specify a simple filename.
- If the file is located in another directory, a program or operator must include a directory prefix in one of the following forms:

  ```
  partition-name:simple-filename
  subdirectory-name:simple-filename
  partition-name:subdirectory-name:simple-filename
  ```

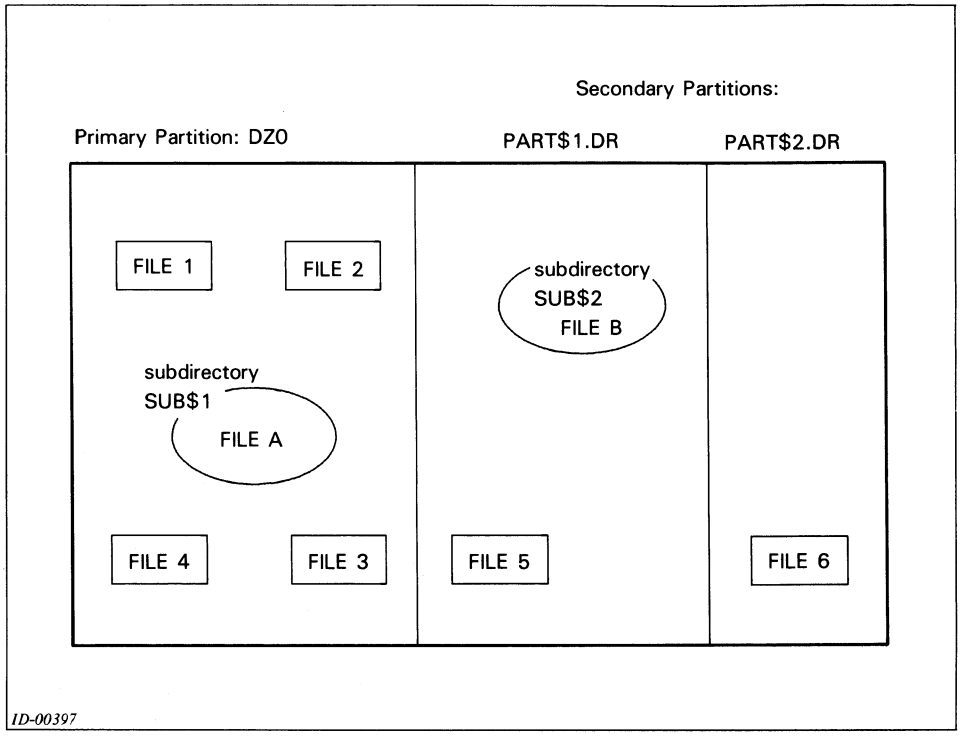


Figure 1-3 *Subdirectories*

For example, if you invoke the runtime system from directory DZ0, you would access the files in Figure 1-3 in the following form:

```
FILE1          PART$1:FILE5    SUB$1:FILEA
FILE2          PART$2:FILE6    SUB$2:FILEB
FILE3
FILE4
```

Note that FILEA does not reside in the working directory. If you invoke the runtime system from PART$1, the files are accessed as follows:

```
DZ0:FILE1      FILE5           SUB$1:FILEA
DZ0:FILE2      PART$2:FILE6    SUB$2:FILEB
DZ0:FILE3
DZ0:FILE4
```

Directory Initialization

A directory must be initialized, or identified to the operating system, before it can be used by COBOL programs. The CLI command INIT performs initialization. DIR initializes a directory and makes it the working directory. The operating system resides in the *master directory,* which is automatically initialized upon system startup.

The operating system can maintain a certain number of concurrently initialized partitions and subdirectories. This number is specified when the system is generated or configured. To support the usage of a large number of directories, the system allows a directory to be released, making its slot in the directory control table available to another one.

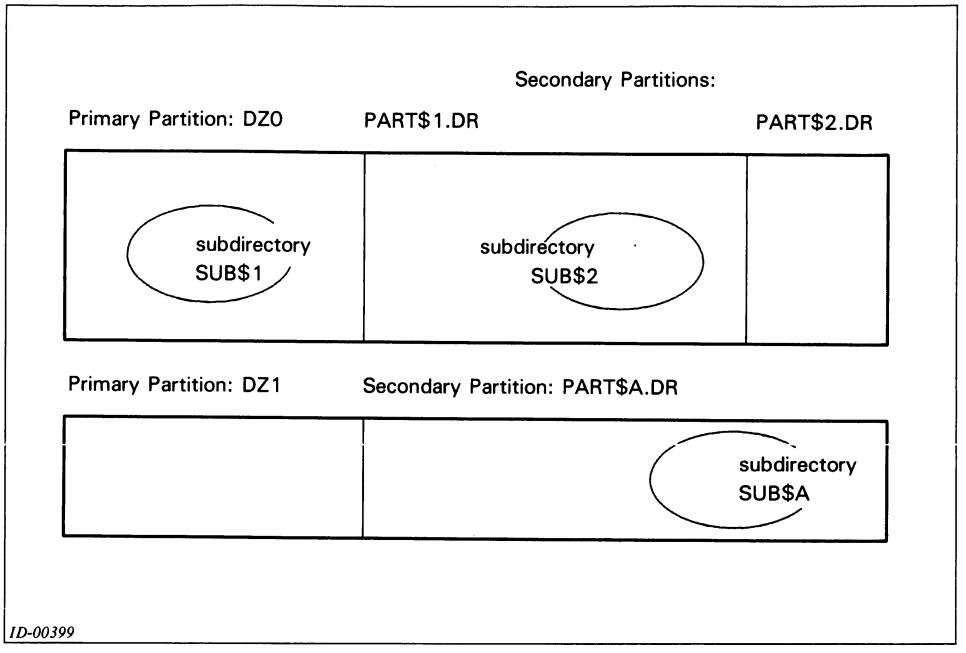


Figure 1-4 *Directories in a two-disk system*

A COBOL program can initialize and release directories with system calls. The system call argument must be a data-name or a nonnumeric literal.

```
CALL PROGRAM "#Idirectory-name" (Initialize a directory)
CALL PROGRAM "#Rdirectory-name" (Release a directory)
```

If the partition or subdirectory to be initialized resides in the working directory, its simple name is enough; otherwise, a partition-name prefix is required. A partition must be initialized before any of its subdirectories.

Figure 1-4 depicts a two-disk system. Suppose that the runtime system is started from directory DZ0, with no other directories initialized. The following system call sequence initializes all other directories:

```
CALL PROGRAM "#ISUB$1".
CALL PROGRAM "#IPART$1".
CALL PROGRAM "#ISUB$2".
CALL PROGRAM "#IPART$2".
CALL PROGRAM "#IDZ1".
CALL PROGRAM "#IDZ1:PART$A".
CALL PROGRAM "#IPART$A:SUB$A".
```

From directory DZ0, the DZ1 prefix is required in DZ1:PART$A because PART$A does not reside in the working directory. Likewise, the partition-name prefix PART$A is required to inform the operating system of the location of subdirectory SUB$A.

A prefix is unnecessary when releasing directories. The operating system always knows the location of an initialized directory.

Access to Files Outside the Working Directory

In addition to the directory prefix facility described above, your operating system two features, links and equivalences, that access files which are not in the working directory.

Links

RDOS and DG/RDOS provide an alternative to directory prefixes for accessing files outside the working directory. The CLI command LINK can make a file that actually resides in one directory appear to reside in one or more other directories as well. Links can be established only through the CLI, not by Interactive COBOL programs. Whenever the link filename is used in a COBOL program or elsewhere, a file in another directory—the *resolution file*—is the one actually processed.

The link command has two forms:

```
LINK resolution-filename/2
LINK link-filename [directory:]resolution-filename
```

The first form creates a link from the current directory to a file in the parent directory. The link has the same name as the resolution file. Use the second form to give the link file a different name from the resolution file or to specify a resolution file that is not in the parent directory.

For example, the command `LINK ALINK.SV DZ0:APROG.SV` creates a link between ALINK in the working directory and the program APROG in DZ0.

To eliminate links, use the UNLINK command. The DELETE command follows the link and deletes the resolution file.

Equivalences

In a multidisk system, primary partitions have system-defined mnemonic codes (DZ0, DP0, etc.). With the EQUIV command, you can assign a filename to any primary partition except the master directory (the one from which the operating system was started).

The format of the EQUIV command is

```
EQUIV newname oldname
```

Properly applied, EQUIV gives your programs device-independence. You can write a generic device specifier into your programs and use EQUIV to assign the generic specifier to a specific device at runtime.

Use the EQUIV command before a device has been initialized. The new name exists only until the device is released. After a device is released, it reverts to its original specifier. Do not use EQUIV on a secondary partition, subdirectory, or a master device.

For example, the command `EQUIV TAPE MT0` establishes an equivalence. `INIT TAPE` initializes the tape drive. Note that all references to TAPE are resolved to the first tape drive, MT0.

File Access Restrictions

One or more file attributes can be assigned to each file. Each attribute restricts all users and programs from performing a certain type of file-access function. Absence of an attribute means that users and programmers can perform the file-access function. Table 1-2 lists and defines file attributes.

Code	Attribute	Definition
A	Attribute-protected	The file's attributes may not be altered. (May be assigned only with a system call, not from the CLI.)
N	No resolution	The file may not be accessed through a link.
P	Permanent file	The file may not be deleted or renamed.
R	Read-protected	No program may read data from the file.
S	Save file	The file is an executable core image.
W	Write-protected	No program may write data to the file.

Table 1-2 RDOS and DG/RDOS File Attributes

Interactive COBOL program files created by the COBOL compiler, and data files created by the runtime system are originally assigned no attributes. Thus, access to these files is unrestricted when they are created. The CLI CHATR command changes the attributes of any file. For instance, a developer might wish to make all COBOL object program files permanent.

RDOS and DG/RDOS implement selective access to files through the link attribute facility. The CLI command or system call CHLAT assigns file attributes to a link entry. The link to a file can have different attributes than the actual file. For example, COBOL program operators access a data file through a link, while the system manager has direct access. The link entry is assigned the W attribute to restrict the COBOL operators from writing or deleting records; the file itself does not have this attribute, thus allowing the system manager to perform these operations.

Assigning Data Files

Throughout an Interactive COBOL program, a data file is referenced by a COBOL data-name, its *internal name.* In the SELECT clause, a file's internal name is assigned to an *external name,* an RDOS (or DG/RDOS) filename. In an Interactive COBOL program, this assignment is made whenever an OPEN statement is executed, perhaps several times during a single program execution. Moreover, the external filename can be stored in a data-item rather than coded as a literal, thus allowing you to supply a filename at runtime.

If the external filename option is omitted from the SELECT clause, the system filenames are supplied by default. Table 1-3 lists these filenames.

Device	Filename
PRINTER	$LPT
PRINTER-1	$LPT1
DISPLAY	$TTO
KEYBOARD	$TTI
DISK	First 10 characters of the internal filename; $ replaces -.

Table 1-3 Default Filenames

Assigning Files to Disk Storage

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

SELECT filename ASSIGN TO DISK [, id-lit]

Filename is the name of the COBOL, or internal, file. If the identifier or literal is omitted, the runtime system automatically creates an external name by (1) truncating the internal name, if necessary, after the tenth character and (2) substituting dollar signs for hyphens. Several examples follow:

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

Use the SELECT clause with an identifier or literal to define an external filename.

When using a literal, the name of the file (and therefore its location) is fixed and cannot be changed without recoding and recompiling the program. For example, DATAFILE-INT is assigned to the disk file EXT$FILE in the following entry:

```
SELECT DATAFILE-INT ASSIGN TO DISK, "EXT$FILE".
```

Using an identifier to store the external filename as a data-item, rather than coding it as a literal, allows more flexibility. The value of the identifier must be a valid external filename, or the file must be left-justified in the identifier. Using the disk specifier prefix with the filename allows the file to be moved between disk drives without having to recode and recompile the program.

Using an identifier to specify an external filename means that a program is not bound to a particular data file or set of files. In the following example, at different times during program execution, one internal file is assigned to different external files:

```
SELECT DATAFILE-INT ASSIGN TO DISK, EXT-FILE-NAME.

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

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

    MOVE "DATAFILE3" TO EXT-FILE-NAME
    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 assign a filename interactively at runtime. For example:

```
SELECT DATAFILE-INT ASSIGN TO DISK, EXT-FILE-NAME.
    ...
    ACCEPT OP-RESPONSE.
    MOVE OP-RESPONSE TO EXT-FILE-NAME.
    OPEN OUTPUT DATAFILE-INT.
    ...
    CLOSE DATAFILE-INT.
```

Assigning Files to Hardware Devices

A program may perform I/O operations to a physical device instead of to a disk-resident data file. In such situations, use the RDOS or DG/RDOS name of the physical device as the external filename in the SELECT entry (see Table 1-4).

Device	External Filenames
Printers	$LPT, $LPT1
Primary console keyboard	$TTI
Primary console display	$TTO
Secondary console keyboard	$TTI1
Secondary console display	$TTO1
Terminals and printers	QTY:n
connected to ALM lines	QTY1:n (n = 0-63)
Magnetic tape drives*	MT0, MT1

*Interactive COBOL programs cannot use magnetic tape directly. The tape device names are included for completeness (see "Assigning Files to Magnetic Tape," below).

Table 1-4 RDOS and DG/RDOS Device Filenames

Direct to System Printer

A program can open either of the two system printers, $LPT and $LPT1, as an output file. Records written by the program are sent directly to the printer. Use the following ASSIGN format:

```
ASSIGN TO  { PRINTER    [, "$LPT" ] }
           { PRINTER-1  [, "$LPT1"] }
```

If you omit the external name, the system assigns the appropriate printer name, and, if PASS is enabled, it puts the file on the PASS queue. When the OPEN statement is executed, other system users cannot use the printer. An attempt by another COBOL program to open the printer produces a File Status code of 94.

If the system has only one printer, or if the second printer is off line, use the following link command to reroute jobs that have been assigned to PRINTER-1 to PRINTER:

```
LINK $LPT1 $LPT
```

Direct to Auxiliary Printer

A program can open a printing terminal or serial printer on a QTY line as an output file. Records written by the program are sent directly to the printer. Use the following ASSIGN format:

```
ASSIGN TO  { PRINTER   }    "QTY:n"
           { PRINTER-1 }
```

The number *n* is the QTY line number; it can range from 0 to 63.

Indirect Spooling to the PASS Queue

In a system with many users, it is impractical for one or two users to exclude all others from access to the system printer(s). Therefore, printer-ready output can be spooled to a disk file. The filename is automatically placed on an operator-controlled dispatching list, the PASS queue. This process does not tie up a printer. Using this indirect method, any number of programs can access the printer concurrently. Use the following ASSIGN format:

```
ASSIGN TO  { PRINTER   }    , id-lit
           { PRINTER-1 }
```

The PASS (Printer Access Scheduling System) queue is a list of files that programs have created and assigned to PRINTER or PRINTER-1. At the master terminal for the ground, you can select files for printing in any desired order. PASS also allows printing of multiple copies and parts of files. If the PASS queue is full when the OPEN statement executes, File Status code 99 is returned. See chapter 3 for a description of PASS.

Assigning Files to Magnetic Tape

Interactive COBOL does not support the ASSIGN TO TAPE clause. However, if the external filename specifies a magnetic tape file (MT0 or MT1) and the file type is sequential, magnetic tape files may be accessed.

Assigning Files to Display Terminals

Interactive communication between a program and a display terminal usually is controlled by DISPLAY and ACCEPT statements through Screen Section data structures. In certain cases, however, the display screen may be treated as an output file or the keyboard as an input file. For these purposes, the SELECT entry includes the clause ASSIGN TO DISPLAY or ASSIGN TO KEYBOARD. As with printer assignments, DISPLAY and KEYBOARD assignments can be direct or indirect. Files assigned to KEYBOARD or DISPLAY are treated as data-sensitive.

The direct method uses the actual keyboard or display for reading input or writing output. Send a report directly to the display screen with this method.

With the indirect method, a disk file provides input or receives output. For output, the indirect method closely resembles spooling printer output to a disk file. The only difference is that disk files assigned to DISPLAY are not placed on the PASS queue. For input, you can use the indirect method to read a line-sequential file (one produced by a text editor, for instance).

The ASSIGN format for the direct and indirect methods is the same:

```
ASSIGN TO  { DISPLAY  }    , id-lit
           { KEYBOARD }
```

However, with the direct method, the identifier or literal specifies an actual terminal in the form QTY:*nn*. With the indirect method, the identifier or literal specifies a disk-filename.

Disk Storage Allocation

Interactive COBOL data files can be implemented as random files or contiguous files. In an Interactive COBOL program, the INDEX SIZE and DATA SIZE clauses specify contiguous files (only the DATA SIZE clause is used with sequential files). The absence of these clauses indicates a file that is randomly allocated.

Random Files

A random file consists of disk blocks that need not be physically adjacent to each other on the disk. The operating system automatically maintains an index structure to keep track of the various blocks it assigns to the file. This index structure is completely independent of the index portion of an indexed or relative data file.

COBOL data files with random allocation may grow as needed. The operating system can assign any currently unused block in the partition to a data file that requires additional storage space. This dynamic file growth is handled by the Interactive COBOL runtime system and the operating system; it is transparent to COBOL programs.

Contiguous Files

A contiguous file consists of a fixed number of disk blocks with consecutive disk addresses. When a COBOL program tries to create such a file, the operating system must be able to locate the specified number of available consecutive blocks. Moreover, contiguous files cannot grow dynamically. If a program tries to write additional records to a contiguous file whose index portion or data portion is full, an error occurs. Though storage allocation is less flexible than random files, contiguous files may provide better data access (depending on the application), since record retrieval from a random file requires the use of the random file index structure.

Selecting Random or Contiguous Allocation

The presence or absence of DATA SIZE or INDEX SIZE clauses in the file's SELECT entry determines whether the file is stored randomly or contiguously.

To create a sequential file that has contiguous allocation, include the DATA SIZE IS *integer* clause in the SELECT entry. This creates a contiguous file containing the specified number of blocks when the program executes an OPEN OUTPUT statement. If available disk space does not permit the allocation of a contiguous file, the OPEN statement causes an I/O error with a File Status of 98. Absence of the DATA SIZE clause specifies random allocation.

At the operating system level, ISAM files are implemented in two portions: *filename*.NX and *filename*.XD. The clause DATA SIZE IS *integer* causes the operating system to allocate the data (.XD) portion contiguously. The clause INDEX SIZE IS *integer* causes the system to allocate the index (.NX) portion contiguously. Allocation takes place when the program executes an OPEN OUTPUT or OPEN I-O statement (only if the file is created, not if it already exists). The .NX and .XD portions may be allocated have differently. If the DATA SIZE IS clause is absent, the .XD portion is allocated randomly. If the INDEX SIZE IS clause is absent, the .NX portion is allocated randomly.

If a random file cannot grow dynamically because disk space is exhausted, the WRITE statement causes an I/O error with a File Status of 34.

Several Interactive COBOL utilities help allocate, monitor, and maintain files. The FILESTATS utility helps you decide how many blocks to allocate to a sequential data file and to each portion of an indexed or relative file. The ANALYZE utility helps monitor file usage and determine when a contiguous file's space is nearly exhausted. The REORG utility can make a copy of a contiguous file with a larger allocation, if disk space is available. REORG also can convert files between contiguous and random allocation (see the *Interactive COBOL Utilities* manual).

Exclusive and Nonexclusive File Usage

The Interactive COBOL runtime system always opens an indexed or relative data file exclusively. When the runtime system opens a file exclusively, no program running in the other ground (including a COBOL program) can access the data file. To prevent programs in the same ground from using an indexed or relative data file, use the COBOL OPEN EXCLUSIVE statement.

The Interactive COBOL runtime system does not open a sequential file exclusively unless you specify the EXCLUSIVE option in the OPEN statement. The COBOL program, not the runtime system, must prevent two COBOL programs on different terminals or in different grounds from using the same sequential file.

Chapter 2
Interprogram Communications

Interactive COBOL provides two statements that allow an Interactive COBOL program to call another program. The CALL PROGRAM statement lets an Interactive COBOL program invoke a group of operating and runtime system utility functions. The CALL statement lets an Interactive COBOL program call another Interactive COBOL program or an assembly language subroutine. Calls to Interactive COBOL programs are discussed in the *Programmer's Reference.* Other calls are discussed in this chapter.

System Calls

The CALL PROGRAM statement can be used for a system call or to call another Interactive COBOL program. The system-call form 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. (For convenience, several of the utility functions are available through the standard Logon program.)

In the standard CALL PROGRAM form, control does not return to the calling progam. However, control returns to the calling program after execution of a system call. The runtime system updates the EXCEPTION STATUS data-item and, in some cases, a FILE STATUS data-item when it executes a system call. File and Exception Status codes are listed in appendix A.

From a programmer's point of view, it does not matter whether a call invokes a function performed by the operating system or by the runtime system. However, the programmer does need to consider the fact that some system calls can be performed only by programs running at the master terminal (the terminal with line number 00).

System calls can be executed from three environments:

- Directly from the Logon menu
- After entering R from the Logon menu
- From a COBOL program

Table 2-1 lists the system calls in alphabetical order; extended descriptions of the calls appear below.

Letter	Function
#A	Abort a COBOL program running at another terminal (not in CLI mode; master terminal only)
#C	Physically close files not currently in use
#D	Run a COBOL program under debugger control
#F	Perform a full initialization on a disk's primary partition or a on magnetic tape
#H	(1) Hang up a dial-up line (2) Deactivate a local terminal
#I	Initialize a directory or magnetic tape
#L	Chain to Logon
#M	Send a message to all terminals logged on to the runtime system (not in CLI mode)
#N	Rename a file
#O	Allows you run a program that is not attached to any terminal and assign default output to a file (not in CLI mode; master terminal only)
#P	Use PASS (not in CLI mode; master terminal only)
#R	Release a directory or magnetic tape
#S	Shut the runtime system down, returning control to the CLI (master terminal only)
#T	Display the status of all Interactive COBOL terminals in the ground
#W	Pause for a specified period of time

Table 2-1 System Calls

Abort Program: #A

The master terminal operator can abort a COBOL program running at any other terminal in the same ground. When invoked, this utility prompts the operator to enter the line number of the terminal whose program is to be aborted. When the master terminal operator types a number and presses a terminator key, the following actions occur:

1. At the specified terminal, no warning is issued. The runtime system closes any open files, stops the program, and displays the message JOB ABORTED BY OPERATOR.
2. If the program is executing an ACCEPT statement, the abort does not take effect immediately but occurs as soon as the operator at the specified terminal presses a terminator key. Data on the screen at the time the abort takes effect may be lost.
3. The terminal is deactivated, but the operator can continue working by pressing CR to restart the Logon program.
4. At the master terminal, a confirming message appears. The operator can abort another program by pressing CR, then entering another number. Pressing ESC at any time returns control to Logon.

This call cannot be used if you have brought up the runtime system in CLI mode.

Physically Close Files: #C

The CALL PROGRAM and CLOSE FILE statements close files and flush the updated contents to disk. The operating system channels remain open and the files, though logically closed, are still physically open. Files remain physically open until one of the following occurs:

- The runtime system is terminated
- The runtime system file table is full and a request is made for another file

- A #C system call is issued

The syntax for #C is as follows:

CALL PROGRAM { "#C" / "#C filename" }

Used without a filename argument, #C physically closes all files not currently in use. If no files can be closed, Exception Status code 048 is returned. If a filename argument is provided, only the specified file is physically closed. If the file is not open (i.e., it did not need to be closed), the Exception Status code 013 is returned. If the file is in use, code 048 is returned. If *filename* is not valid, code 001 is returned. A successful program call returns code 000. #C is useful when users in another ground need access to files which have been previously accessed by the runtime system running in the current ground.

Should the runtime system or operating system terminate abnormally before the files are physically closed, the files' use counts in the system directory entry are greater than zero. In this case, issue the CLEAR/V *filename*.<NX,XD> command. All files that were logically closed by the runtime system should be intact; however, run ISAMVERIFY to determine file integrity.

Debug a Program: #D

The call #D*program-name* lets you 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.

Fully Initialize a Directory or Tape Drive: #F

The #F is equivalent to the CLI command INIT/F. If the form is #F *disk-specifier*, the primary partition files SYS.DR and MAP.DR are rebuilt, effectively destroying all data and programs currently on the disk. If the form is #F *tape-drive*, the system rewinds the tape and writes two EOFs at the beginning, effectively erasing the tape by allowing files to be written on it.

Hang Up the Terminal: #H

With dial-up terminals, the call hangs up the telephone line. With local terminals, the call logs the terminal off the system, blanking its screen. The user may log on again, dialing in again or by pressing uppercase F (foreground) or B (background). However, if the runtime system's maximum number of terminals has been reached, a message to that effect appears.

Initialize a Directory or Tape Drive: #I

The form #I *directory-name* initializes a directory. A directory must be initialized before any of its files or subdirectories can be accessed. The form #I*tape-drive* initializes a tape drive.

Call Logon, Leaving the Terminal Inactive: #L

This utility calls the Logon program but puts the terminal in inactive status. It remains inactive until Logon executes a CALL PROGRAM statement. (CALL PROGRAM "LOGON" leaves the terminal active.)

Message Broadcast: #M

This call lets you send a message to all terminals logged on to the runtime system in the current ground. You can type up to 59 characters in a standard COBOL input field. When you press a terminator, the runtime system displays the message on line 24 of the other terminals. The message remains on the screens until overwritten by another message or by a BLANK SCREEN or BLANK LINE function. Pressing ESC at any time returns control to Logon.

This call cannot be used if you have brought up the runtime system in CLI mode.

Rename a File: #N

This call lets you rename a file. The syntax is as follows:

```
CALL PROGRAM "#N old-filename new-filename".
```

The filenames can be identifiers or nonnumeric literals. You can use pathnames; however, if *old-filename* is a pathname, *new-filename* must also be a pathame. Separate the filenames by at least one space. 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.

Run a Detached Program: #O

This call allows you to run a program detached from a terminal. Its form is

```
CALL PROGRAM "#O program-name output-file-name".
```

Program-name must be an external COBOL program-name. One or more spaces must separate *program-name* and *output-file-name*. *Output-file-name* is the name of the default output file. All display output and error messages are written to it (note that the file will contain control codes). The file is created if it does not exist and is appended to if it does exist. This call cannot be used if you have brought up the runtime system in CLI mode.

Use PASS: #P

The Printer Access Scheduling System (PASS) is described in chapter 3. The #P call is valid only at the master terminal and cannot be used if you have brought up the runtime system in CLI mode.

Release a Directory or Tape Drive: #R

#R is equivalent to the CLI command RELEASE, which logically removes a tape drive or directory from the system. RDOS and DG/RDOS have a maximum number of directories that can be initialized concurrently; therefore, it is good practice to release a directory if no further access to it is anticipated. The form #R*directory-name* releases a directory and any subordinate directories. #R*tape-drive* rewinds the tape.

Stop Runtime System Execution: #S

The stop function terminates runtime system execution and returns control of the ground to the CLI. You can use this function only at the master terminal and if all other terminals and printers in the ground are inactive. (The #H, #L, and #A system calls deactivate a terminal.) If any terminal is still active, an error message to this effect is displayed. If this occurs, pressing ESC returns control to Logon or the CLI (if in CLI mode).

Terminal Status: #T

This call displays the active or inactive status of the COBOL terminals in the same ground. Press CR to update the display or press ESC to return control to Logon or the CLI (if in CLI mode).

The following information is displayed:

- Terminal number, status, and type (dial-up or local)
- COBOL PC
- The number of ISAM files presently open and the number that can be opened
- The number of records presently locked and the number that can be locked
- The number of terminals presently logged on and the number that can be logged on
- The number of input/output buffers available to the runtime system

Wait for a Specified Time: #W

#W*integer* suspends program execution for a time period expressed in tenths of a second. The default integer is 30, producing a three-second pause. The maximum integer is 65,535, producing a pause of 109 minutes and 13.5 seconds. No CPU time is used during the pause.

Calls from the Logon Menu

Any system call that can be executed from a COBOL program can be executed from Logon. Both environments simply chain to a program call. Logon can execute additional system calls if the entry is added to the Logon menu and code is added to the Logon program.

For example, to execute #I (initialize a directory), an appropriate entry needs to be added to the Logon menu, and *R* has to be added to the table of valid responses. Since #I takes a directory-name argument, Logon should accept a string defining the directory to be released. In general, if a system call requires arguments, an ACCEPT *screen-name* statement with an input field has to be performed. Logon has to build a structure that includes the system call name ("*#letter*") and the necessary arguments. The utility is actually called by executing a CALL PROGRAM statement.

You can prevent users from executing system calls from the Logon menu by preventing the Logon program from recognizing certain symbols.

Calls from Logon Option R

The R option (Run a COBOL program) from the Logon menu displays a screen that receives as an argument the name of the COBOL program to be executed. The Logon program then chains to this program. Thus Logon chains to any system call recognized by the runtime system. Even those system calls hidden from the user by the developer can be executed. To execute a system call, respond to the program name prompt by entering the system call name and any arguments. For example, to initialize a directory, type "#I*directory-name*".

Calls from Inside a COBOL Program

The CALL PROGRAM statement causes a chain to the desired system call. There are two possible formats for system calls from a user's program. You can make an explicit call, or a use structure that defines both the call and any arguments. For example, the following call physically closes a file:

```
CALL PROGRAM "#Cfilename".
```

To add flexibility, you could define a structure such as

```
01  UTIL-STRUC.
    02  UTIL-NAME      PIC XX.
    02  ARG1           PIC X(56).
    02  FILLER         PIC X     VALUE LOW-VALUE.
```

Then invoke a system call by executing the following statement:

```
CALL PROGRAM UTIL-STRUC.
```

For a #C system call, execute the following code:

```
MOVE "#C" TO UTIL-NAME.
MOVE "FILENAME" TO ARG1.
CALL PROGRAM UTIL-STRUC.
```

Errors in System Calls

Errors are returned as 3-character Exception Status codes. When an attempt is made to CALL PROGRAM "*#x*" where *#x* is not a system call that supported by the runtime system, the message CALLED PROGRAM NOT FOUND (Exception code 203) is returned. See appendix A for a list of Exception Status codes and their meanings.

Assembly Language Calls

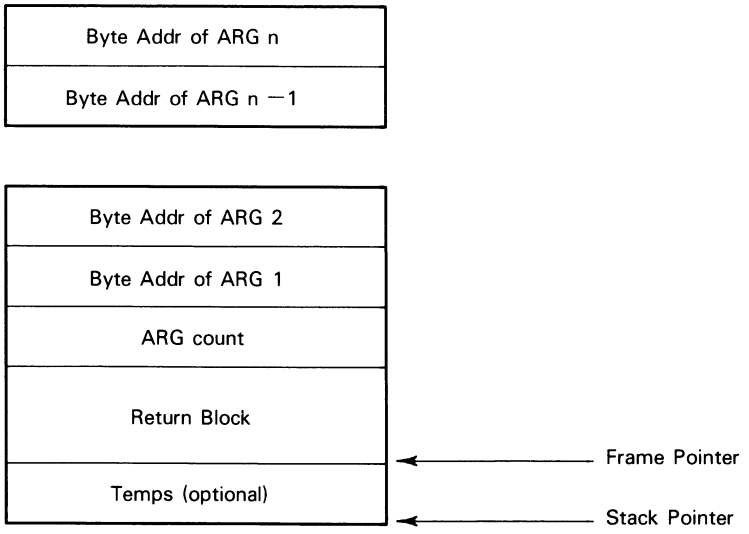
An Interactive COBOL program can use the CALL statement to call an assembly language subroutine. The subroutine must be written in 16-bit Eclipse instructions. For the call to complete successfully, follow the conventions listed below.

Calling Conventions

A name table must be defined that gives the names and addresses of all assembly language routines called by the Interactive COBOL program. The name table must be defined using the label "NAMTB," which must be made an external for the runtime system to find it. Each routine 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. There are no restrictions on the characters allowed in this text string, except that they must match the characters in the identifier or literal used in the CALL statement. The second word in the name table entry is the address of the actual routine.

Begin the called routine with a SAVE instruction. This saves the contents of accumulators (ACs) 0, 1, and 2, the frame pointer and the return address on the stack. It also creates a new stack frame, the address of which is loaded in AC3.

The parameters are accessed by indexing off the frame pointer in AC3. Figure 2-1 shows the contents of the stack after the SAVE instruction.



ID-02165

Figure 2-1 *Stack contents after the save*

The maximum number of parameters that can be passed to an assembly language subroutine is 32.

End all exit code paths in the subroutine with a RTN instruction. This restores the accumulators and returns control to the statement following the CALL.

Define external names if the assembly language routines are in different source files. To ensure that the external names do not conflict with external names defined by the runtime system, thereby causing link errors, use names that start with the characters *UU*. The runtime system will not use external names that begin with UU.

Restrictions

Be aware of the following restrictions when coding assembly language routines:

- All code and data must be in NREL. ZREL is reserved for patches to the runtime system and should not be used.
- The name table (NAMTB) must be located at the first address in the first module linked into the runtime system.
- Since the Interactive COBOL runtime system does not use the standard RDOS task scheduler, do not use the RDOS task calls documented in the *RDOS Reference Manual*. If an assembly language subroutine uses these calls, unresolved references occur when the runtime system is built.
- To use I/O mode to perform I/O instructions, turn off LEF mode by calling the routine ?LFOF. Restore LEF mode by calling the routine ?LFON before the subroutine does a system call. ?LFOF and ?LFON are documented below.

  Warning: If a system call is made while LEF mode is turned off, the runtime system will trap.
- The runtime system does not save the state of the floating point accumulators when a system call is made. To execute floating point operations and save the state of the floating point accumulators, you must save them on the stack before issuing the system call.

- Whenever an assembly language subroutine issues a system call that requires an address or a byte pointer to an area in memory, that memory area cannot be an area in the COBOL program or on the ECLIPSE stack. If a system call is issued using an address that points to an area in the ECLIPSE stack or to an area in the COBOL program, unpredictable errors will occur.

  The memory area can be defined within the assembler module or in the buffer whose address is returned by the support routine ?GTBF. ?GTBF is documented below.

Support Routines

The runtime system includes a set of five routines that an assembly language subroutine can call. The interface to these routines is described below. Each subroutine called must be defined as an external with the .EXTN statement.

?LFON: Turn LEF Mode On

Format:
```
EJSR   ?LFON
normal return
```

Input: None

Output: None

Description: Turns LEF mode on (turns I/O mode off)

?LFOF: Turn LEF Mode Off

Format:
```
EJSR   ?LFOF
normal return
```

Input: None

Output: None

Description: Turns LEF mode off (turns I/O mode on)

?PSEM: Lock a Semaphore

Format:
```
EJSR   ?PSEM
semaphore address
normal return
```

Input: None

Output: None

Description: ?PSEM and ?VSEM provide a general mechanism for mutual exclusion or resource allocation of a critical section of code. The semaphore address that is passed inline following the EJSR instruction is the address of a single word of memory used by ?PSEM and ?VSEM. Initialize this word to a value of 1 for critical sections or to the number of resources for resource allocation.

?PSEM locks a semaphore. If the semaphore is already locked, the task is pended until a ?VSEM call is issued by another task.

?VSEM: Unlock a Semaphore

Format:
```
EJSR   ?VSEM
semaphore address
normal return
```

Input: None

Output: None

Description: Unlocks a semaphore. If other tasks are waiting for the semaphore (as a result of calling ?PSEM), the task that called ?PSEM first is readied (i.e., the tasks are readied on a first-come-first-serve basis). ?PSEM and ?VSEM provide for mutual exclusion to a critical section of code.

?GTBF: Get a Buffer for the Task

Format:
```
EJSR   ?GTBF
normal return
```

Input: None

Output: AC1 = Length of buffer in words
AC2 = Word address of buffer

Description: Returns the address and length of a resident buffer to be used by a task. A different buffer is returned for each task, thus these buffers may be used to pass data to and from system calls. For example, to open a file the filename could be placed in this buffer before calling .OPEN. The runtime system uses this buffer, therefore its contents may be changed between calls to assembly language subroutines.

Building the Runtime System

Link all assembly language routines into the runtime system with the ICXLINK macro (described below). All assembly language subroutines are available to all programs executing under control of the runtime system. Since the subroutines are bound into the runtime system, they reduce the space available for programs (.PD and .DD files) and file buffers. The FLEXSTATS utility prompts for the size of the subroutines and takes this into account when determining whether a specified configuration is legal.

The following files are needed to build the runtime system:

- ICX.LB (a library file containing most of the resident code for the runtime system)
- a set of .RB files for the overlays and the debugger
- DICXLINK and ICXLINK, two CLI macros that build the runtime system with and without a debugger, respectively.

To build the runtime system, execute a command line of the form:

```
DO {CXLINK  } save-name load-addr {mod-name}...
   {DICXLINK}
```

save-name is the name of the .SV file to be generated. Specify the name without an extension. Do not specify ICX as the save-name; this causes the standard ICX.SV and ICX.OL files to be deleted.

Load-addr is the address (in octal) at which the assembler routines will be loaded. The load address must be correct, or the runtime system will not execute properly. To determine the load address, first determine the combined size (in words) of all of the assembler modules to be linked in. If you have invoked ICXLINK, subtract the size from 77400 (octal) to give the load address (in octal). If you have invoked DICXLINK, subtract the size from 65400 (octal).

Mod-name is the name of your assembler modules to be included. You must specify at least one module: the module that contains the name table (NAMTB). NAMTB must be at the first address in that module. Up to seven modules can be included.

DICXLINK works like ICXLINK except that an assembler debugger is included in the runtime system to debug the assembler routines.

To debug the runtime system, enter the following at the R prompt:

```
DEB runtime-system-name
```

You can use the options described in "Starting the Runtime System" in chapter 3. However, because the runtime system occupies resident memory, certain combinations will be too large to bring up.

The debugger, which is loaded into the runtime system, is similar to the standard RDOS symbolic debugger. Differences are listed below; otherwise, the commands are the same as those described in *RDOS/DOS Debugging Utilities*.

- The way in which preceding memory locations are displayed is changed. The up-arrow key (CTRL-W) closes the current memory location and displays the contents of the preceding memory location. Successive preceding locations can be opened and displayed by repeatedly pressing the up-arrow key. Typing the system terminator (NEW LINE or CR) closes the currently opened memory location. The carat key (SHIFT-6) has no function.
- If you attempt to set a break point and no break point registers are avaliable, a bell and a question mark are displayed.
- Continue execution address register has been added. The ESC-G register contains the address where execution stopped at a break point and will proceed after you enter the ESC-P command.
- The down-arrow key (CTRL-Z) closes the currently opened memory location and opens the next memory location. This key has the same function as the non-terminator CR or NEW LINE (this is system dependent). Repeated pressing of the down-arrow key opens and displays successively higher memory locations until you press the system terminator, which closes the currently open memory location.

For example, assume that one assembler module named ASMEX is to be linked into the runtime system. Further assume that the size of ASMEX is 74 (octal) words. You want to name the save file ALTICX.SV. To create a version of the runtime system that includes ASMEX and that can be debugged, the command line would be:

```
DO DICXLINK ALTICX 65304 ASMEX
```

Four files are produced:

ALTICX.SV (the save file)
ALTICX.OL (the overlay file)
ALTICX.ER (the runtime system error message file)
ALTICX.MP (the load map)

Examine ALTICX.MP to verify that the link worked correctly. In particular, all external references should be resolved, and if the load address was calculated properly, NMAX should be less than 77400 (octal).

To debug the runtime system, enter the following at the R prompt:

```
DEB ALTICX
```

After debugging the program, use ICXLINK to produce a version of the runtime system without the debugger. The command line is:

```
DO ICXLINK ALTICX 77304 ASMEX
```

Examine ALTICX.MP to make sure that all external references are resolved. If the load address is correct, then NMAX is 77400 (octal) when ICXLINK is run.

Example

The following example shows an Interactive COBOL program (RDOSASMCALL) calling an assembly language subroutine. The COBOL program allows you to enter the name of a file. The assembly language subroutine (ASMEX) returns the size of the file or an error code (if the file does not exist, for example). The COBOL program then displays the size of the file or the error code.

For this example, MAC.PS was created from the standard parameter files accompanying RDOS or DG/RDOS by using the following command line:

```
MAC/S NBID NEID NCID NFPID OSID PARU
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RDOSASMCALL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ICOBOL.
OBJECT-COMPUTER. ICOBOL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ERROR-CODE          PIC 9(4) COMP.
01  ERROR-CODE-DISPLAY  PIC 9(4).
01  FILENAME-STRUCTURE.
    03  FILENAME        PIC X(40).
    03  TERMINATOR      PIC X VALUE LOW-VALUE.
01  FILE-SIZE           PIC 9(9) COMP.
01  FILE-SIZE-DISPLAY   PIC 9(10).
PROCEDURE DIVISION.
START-PROGRAM.
    DISPLAY "Enter filename: " NO ADVANCING.
    ACCEPT FILENAME; ON ESCAPE GO TO FINISH-PROGRAM.
    INSPECT FILENAME REPLACING FIRST SPACE BY LOW-VALUE.
    CALL "GET-FILE-SIZE" USING
        ERROR-CODE, FILENAME-STRUCTURE, FILE-SIZE;
        ON EXCEPTION GO TO CALL-FAILED.
CHECK-RESULT.
    IF ERROR-CODE = 0
        MOVE FILE-SIZE TO FILE-SIZE-DISPLAY
        DISPLAY "File size is"  FILE-SIZE-DISPLAY "bytes."
    ELSE
        MOVE ERROR-CODE TO ERROR-CODE-DISPLAY
        DISPLAY "GET-FILE-SIZE failed with error: " ERROR-CODE-DISPLAY.
    DISPLAY " ".
    GO TO START-PROGRAM.
FINISH-PROGRAM.
    STOP RUN.
CALL-FAILED.
    DISPLAY "CALL to GET-FILE-SIZE failed.".
    STOP RUN.
```

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

```
; ASMEX - Example program to demonstrate ICOBOL's ability to call
;         assembler subroutines

        .TITLE  ASMEX
        .RB     ASMEX.RB
        .NREL
        .TXTM   1

; Set up the name table

        .ENT    NAMTB
NAMTB:
        GFSIZ*2
        UUGFS
        -1

; Define the ?GTBF routine as an external
        .EXTN   ?GTBF

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

; Define frame pointer
.DUSR FP= 41

; GET-FILE-SIZE - Return the size in bytes of a file

; Expects:
;       ARG1 - Error code (returned) (two bytes)
;               -1 if argument count is invalid
;                0 if call to .RSTAT succeeds
;               RDOS error code if .RSTAT fails
;       ARG2 - Filename (null terminated)
;       ARG3 - File size in bytes (returned) (four bytes)

GFSIZ:  .TXT    "GET-FILE-SIZE<0>"

        MAXFN=  40.             ; Use 40. for max filename length
        BUFAD=  1               ; Stack offset to save buffer addr

        .ENT    UUGFS

UUGFS:  SAVE    BUFAD

        LDA     0,ARGC,3        ; Get the argument count
        LEF     1,3             ; Stuff 3 in AC1
        SUB#    0,1,SNR         ; Is the arg. count 3?
        JMP     GFS2            ;   Yes, arg. count is valid
        MOV#    0,0,SNR         ; Is the arg. count 0?
        RTN                     ;   Yes, no error code arg.
        ADC     2,2             ;   No, return -1 as the
        JMP     GFSER           ;       error code
```

```
GFS2:   EJSR    ?GTBF           ; Get address of buffer to use
        STA     2,BUFAD,3       ;   and save it on the stack
        MOVZL   2,2             ; Byte pointer to buffer
        LEF     0,MAXFN         ; Get max number of bytes for
        MOV     0,1             ;   filename and copy to AC1
        LDA     3,ARG2,3        ; Load filename byte ptr.
        CMV                     ; Copy filename to buffer
        LDA     3,FP            ; Reload the frame pointer
        LDA     2,BUFAD,3       ; Reload buffer address
        MOVZL   2,0             ; Put filename byte ptr in AC0
        LEF     1,(MAXFN/2),2   ; Put UFD buffer after filename buffer
        .SYSTM                  ; Get the file status
        .RSTAT
        JMP     GFSER           ; Error return
        MOV     1,2             ; Put UFD address in AC2
        LDA     0,UFTBC,2       ; Get byte count in last block
        LDA     1,UFTBK,2       ; Get block # of last block
        LDA     2,BLKSZ         ; Get block size in bytes
        MUL                     ; Calculate file size
        LDA     2,ARG3,3        ; Reload byte ptr. to file size
        MOVS    0,3             ; Get first byte in AC3
        STB     2,3             ; Return first byte of file size
        INC     2,2             ; Increment pointer
        STB     2,0             ; Return second byte of file size
        INC     2,2             ; Increment pointer
        MOVS    1,3             ; Get third byte in AC3
        STB     2,3             ; Return third byte of file size
        INC     2,2             ; Increment pointer
        STB     2,1             ; Return fourth byte of file size
        LDA     3,FP            ; Reload frame pointer

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

BLKSZ:  512.

        .END
```

Figure 2-3 *Assembly Language Subroutine*

Chapter 3
The Interactive COBOL Runtime System

The Interactive COBOL runtime system is a program that acts as a monitor or executive for Interactive COBOL object programs. Several terminals can execute programs concurrently under runtime system control. The runtime system can run in the background, in the foreground, or in both grounds at once, thus sustaining two independent multiterminal COBOL environments.

The runtime system performs the following functions:

- Activates a number of tasks, as specified in the CLI command that invokes it.
- Uses a COBOL program named Logon as the default for program execution.
- Implements file sharing and locking to provide data integrity.
- Interprets input-editing characters entered at terminal keyboards.
- Validates input against Screen Section or Working-Storage PICTURE statements.
- Displays error messages regarding input errors and program execution errors.
- Maintains data-items that contain codes to indicate the status of program execution: identity of the last terminator key pressed and result of the most recent I/O operation, or CALL or CALL PROGRAM operation.
- Maintains data-items that store information regarding the operating environment.
- Allows you to interrupt program execution. (Interrupts must be enabled in the CLI command that starts the runtime system.)
- Controls the Interactive COBOL debugger.

Runtime System Files

The runtime system consists of several files:

ICX.SV	Save file
ICX.OL	Overlay file
ICX.MP	Load map, an information file that may be needed to apply patches.
ICX.ER	Error message texts
ICX.LD	Created by DEFLINES utility; contains line characteristics for QTY lines

If necessary during execution, the runtime system creates and manages these additional files:

ICX.LP	PASS queue (runtime system executed in the background)
FICX.LP	PASS queue (runtime system executed in the foreground)

The following temporary files pass data to the Linkage Section of a called program. They are always deleted and re-created:

ICX.UF	(runtime system executed in the background)
FICX.UF	(runtime system executed in the foreground)

The following temporary files maintain current values for programs called at a terminal. *nn* is the terminal number.

ICX*nn*.VM (runtime system executed in the background)
FICX*nn*.VM (runtime system executed in the foreground)

Customizing the Runtime System

The runtime system allows you to decide how to best manage extended memory and other system resources. By specifying the maximum program size, maximum number of terminals, and maximum number of files on the CLI command line, you can customize any particular invocation of the runtime system. You can change the command line parameters according to the configuration of the application software to make the best use of available memory, thereby optimizing the application program's performance.

During initialization, the runtime system allocates and presets all internal data structures related to your specifications, and reserves as much extended memory as possible for use by the file buffering mechanism. It is important to specify the smallest maximum program size necessary so that unused extended memory may be utilized for file buffering, because the performance of the system in an indexed file I/O environment depends heavily on the number of buffers.

Before Starting the Runtime System

Before starting the runtime system, determine the following:

1. The maximum number of program tasks to be active simultaneously (1-33). This includes terminals and detached jobs.
2. The maximum program size (3-31 KB; must be an odd number)
3. The maximum number of indexed and relative files to be open at one time (4-64)
4. The maximum number of additional channels needed (0-236). This includes one channel per sequential file, one channel per task using the COBOL CALL facility, and any channels needed by assembly language subroutines (1-64).

FLEXSTATS

FLEXSTATS is a program that helps you tailor the runtime system to your specific requirements. You must supply FLEXSTATS with the information listed above. FLEXSTATS also asks for the amount of memory available for program space and, if you are using assembly language subroutines, the size of the routines and the number of channels needed. FLEXSTATS checks your information for validity, determines the necessary memory requirements for the runtime system, and displays the command line that tailors the runtime system to your specifications. If the runtime system cannot be tailored to your specifications, FLEXSTATS displays an error message.

You can run FLEXSTATS a number of times, write the results to a logfile, and then determine which configuration of the runtime system best suits your needs. To invoke FLEXSTATS, enter `ICX/C FLEXSTATS` at the command line or `FLEXSTATS` at the `RUN PROGRAM` prompt. The *Utilities* manual gives specific information on FLEXSTATS.

Terminal Line Characteristics

In addition to specifying the total number of I/O channels, the system generation process defines the line characteristics of all terminals connected multiplexor lines. The file ALMSPD.SR is an assembly language source program which stores default line characteristics. To change line characteristics, edit and assemble ALMSPD.SR before system generation.

To set line characteristics after system generation, use the DEFLINES utility. This utility creates the file ICX.LD, which is read at runtime system startup. Line characteristics are adjusted for that invocation of the runtime system. See the *Utilities* manual for instructions on how to use DEFLINES.

Channel Limits

A sequential file uses one operating system channel for each program that opens it. An indexed or a relative file uses two channels, no matter how many programs access it. Each user uses one channel for his terminal; the master console uses two channels, one for TTO and one for TTI. Each terminal that is using the CALL facility uses one channel (that is, one channel for each .VM file). The runtime system can require up to eight additional channels for system files.

The maximum open-file limitations are somewhat flexible. While the maximum number of concurrently open indexed or relative files is fixed, the maximum number of open sequential files is not. A sequential file can be opened whenever a channel is available, even if the sequential file maximum is exceeded. The runtime system also "borrows" against the channels needed to open a display when a user logs on. This borrowing effectively reduces the indexed or relative file maximum and could prevent a user from logging on.

Memory Restrictions

Because both program space and buffer space are allocated in extended memory, the amount of space available for one depends on the amount allocated to the other. By tailoring a runtime system to your software, you can trade off space allocated to one for the other in order to achieve the desired performance characteristics.

There are restrictions on certain combinations of the parameters described above. In particular, increasing the program size not only decreases the amount of extended memory available for buffers, but also decreases the amount of logical memory available for the data structures that describe those same buffers. The number of files (and thus locks) has a similar direct effect on logical memory.

In addition, each program task requires a minimum of two buffers allocated in extended memory. If this is not possible, you must redefine all input parameters to generate a smaller runtime system.

To optimize system performance, the number of buffers should be at least 2 * (*tasks* + *ISAM-files*). If it is not, redefining the input parameters to generate a smaller runtime system system would be beneficial.

You can increase performance by specifying the maximum memory. However, memory exceeding the maximum amount is not used; therefore it should be allocated to the other ground. The FLEXSTATS utility allows you to calculate the maximum amount of memory that a specific choice of parameters is able to utilize.

Table 3-1 shows the minimum and maximum memory the runtime system requires with a maximum program size of 27 KB for various combinations of tasks and ISAM files open.

Tasks	ISAM Files	Min. Memory	Max. Memory
1	16	74	408
13	52	428	636
25	64	778	904
33	64	1012	1094

Table 3-1 Memory Required (KB) with 27-KB Maximum Program Size

Starting the Runtime System

If the runtime system is to execute in the foreground:

1. Link the runtime system error message file with the command `LINK FICX.ER ICX.ER`
2. Start the foreground by issuing either of the following CLI commands at the background master terminal:

   EXFG[/E] CLI (starts foreground CLI on foreground console)
   EXFG[/E] ICX[switches] [arguments]
   (invokes runtime directly in the foreground)

   The optional /E switch indicates equal background and foreground priorities. If the switch is omitted, the foreground has higher priority.

The runtime system can execute in CLI mode, which is used with one-terminal systems only, or in Logon mode. With CLI mode, you specify the name of the Interactive COBOL program to run directly at the CLI. In Logon mode, you run Interactive COBOL programs through the Logon menu.

CLI Mode

If you have a single-terminal system, you can execute Interactive COBOL programs directly from the CLI. The format is as follows:

ICX { **/D** | **/C** } **[/I][/B] prog[/x] [1/T] [size/S] [ISAM-files/F] [channels/N]**

/D	CLI mode; invoke the debugger.
/C	CLI mode; do not invoke the debugger.
/I	Enable interrupts from Interactive COBOL programs.
/B	Enable RDOS or DG/RDOS interrupts.
prog[/x]	The name of the program to be run. The */x* indicates one or more logical switches defined in an Interactive COBOL program. See chapter 1 of the *Interactive COBOL Programmers Reference* for information on logical switches.
1/T	The number of tasks. If this argument is present, one task must be specified or an error is returned.
size/S	Specifies the maximum program size in KB. The value must be an odd number from 3 to 31. The default is 27 KB.
ISAM-files/F	The maximum number of Interactive COBOL ISAM files that may be open concurrently. The value specified must be from 4 to 64. The default is 16 (3 * *tasks* + 13).

channels/N — Specifies the maximum number of additional channels that may be opened. Channels are used for sequential files, .VM files, and files that are opened by an assembly language subroutine. You can specify any number from 0 to 236. The default is 16 (3 * *tasks* + 13).

If you are running in CLI mode, PASS is automatically disabled.

Logon Mode

If Interactive COBOL programs are to be invoked through Logon, enter the following CLI command in the appropriate ground:

ICX[/I][/B] { /S | /P } [tasks[/T]] [size/S] [ISAM-files/F] [channels/N] [QTY-line/M]

/I — Lets you interrupt COBOL programs at all terminals other than the master terminal (optional).

/B — Lets the master terminal operator interrupt execution of the runtime system (optional).

/S — Enables spooling while the runtime system is executing. Spooling continues to be an option when printing a file using PASS.

/P — Disables PASS. Files assigned to PRINTER are not queued; instead, they are written to disk. No printer tasks are available. (Use the CLI XFER command to print files when PASS is disabled.)

tasks[/T] — The maximum number of terminals and detached jobs (#O system call) that can be activated during the execution of the runtime system. The value specified must be from 1 to 33. The default is 1 active terminal. (Note that the /T switch is optional.)

size/S — The maximum Interactive COBOL program size in KB. The value specified must be an odd number from 3 to 31. This is because the total extended memory allocated for each program must be aligned on 2 KB page boundaries, and the runtime system allocates 1 KB for its own use. The default program size is 27 KB.

ISAM-files/F — Specifies the maximum number of Interactive COBOL ISAM files that may be open concurrently during runtime system execution. The value specified must be from 4 to 64, and each requires that two channels be available in the ground. The default number of files is 3 * *tasks* + 13. The maximum number of simultaneously locked records is implicitly derived from this parameter by doubling its value (e.g., 16 files implies 32 locks).

channels/N — Specifies the maximum number of additional channels that may be opened. Channels are used for sequential files, .VM files, and files that are opened by an assembly language subroutine. You can specify any number from 0 to 236. The default is 3 * *tasks* + 13.

QTY-line/M — Assign the master console to the QTY line specified. QTY-line can range from 0 to 63.

Executing COBOL Programs from Logon

The runtime system does not have an elaborate sign-on procedure. This task is left to the applications system. Before you log on, press either of the following keys to begin COBOL processing:

B places the terminal under the control of the background runtime system

F places the terminal under the control of the foreground runtime system

The runtime system automatically runs the Logon program in the working directory.

The Logon Program

The runtime system uses the Interactive COBOL program Logon as its default program. When a terminal is first activated, Logon is automatically executed. When program execution stops, either because it encounters a STOP RUN statement or because a fatal error occurs, control returns to Logon.

If the runtime system cannot locate Logon in the working directory when it is started from the CLI at the master terminal, it displays the error message **LOGON PROGRAM WAS NOT FOUND**, followed by the prompt **RUN PROGRAM**. At this prompt the user may enter any valid CALL PROGRAM argument: a program name or a system call. For example, the user might enter #S to stop the runtime system in order to install Logon in the current directory.

For terminals other than the master terminal, if Logon is not found the terminal is logged off.

Data General supplies a standard Logon program that lets you select a program to run. This Logon lets you select a prewritten CALL PROGRAM statement from a menu or make an independent program call by entering a program name at the **RUN PROGRAM** prompt. For example, if the user enters PROG$423, Logon executes the statement:

```
CALL PROGRAM "PROG$423".
```

The specification of the program can include partition and/or subdirectory names and can take advantage of link and equivalence facilities. It also can include one or more program switches. (Filenaming facilities are described in chapter 1. See the *Interactive COBOL Programmer's Reference* for a discussion of program switches.)

Logon is supplied in both source and object format, to allow developers to modify it for their own purposes. A completely different Logon program can be substituted for the standard one, but it should not open ISAM files directly.

Ending Program Execution

When a program terminates because it executes a STOP RUN statement, the runtime system displays the message **STOP RUN**. If a COBOL program commits a fatal error, the runtime system displays an error message at the bottom of the terminal screen and halts program execution. Programs running at other terminals continue to execute normally.

In either case, the terminal is placed in inactive status (see the next section). Press CR, NEW LINE, or ESC to return to or restart Logon.

The runtime error messages have the form:

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

Relative-procedure-address is a five-digit number that indicates the location in the Procedure Division code of the statement that caused the error. The Interactive COBOL compiler lists the relative procedure address for each paragraph and section in its compilation listing, in place of the procedure's line number. To obtain more precise locations, examine a decompilation listing of the program, which is obtained with the /U compiler switch. Appendix A lists the runtime error messages.

Status of Terminals

A terminal is inactive if:

- It is in Logon. (CALL PROGRAM "LOGON" does not deactivate the terminal; CALL PROGRAM "#L" does.)
- It has returned to Logon after a STOP RUN has been executed, or a fatal error has occurred and a terminator has been pressed.
- No programs have been executed on it since it was aborted by a #A system call or deactivated by a #L system call.

Use the terminal status utility (#T system call) to display the status of each terminal under control of the runtime system. A terminal can be inactive, executing, or pended.

Master Terminal

The runtime system can be executed in either the foreground or background. In each ground, one terminal is the master terminal. The following utility functions, invoked through system calls (see chapter 2), can be performed only at the foreground or background master terminal:

- Operating the PASS queue (print spooler) that serves all terminals in the ground (#P system call)
- Aborting a COBOL program running at another terminal in the ground (#A system call)
- Terminating runtime system execution in the ground (#S system call).
- Running a COBOL program detached from a terminal (#O system call).

COBOL Program Interrupts

The /I global switch in the ICX command that starts the runtime system enables COBOL program interrupts.

In Logon mode at a master terminal:

- CTRL-D (CTRL-C CTRL-D under DG/RDOS) terminates the COBOL program. Pressing CR, NEW LINE, or ESC returns control to Logon. This type of interrupt is similar to aborting by a #A at the master terminal.

In Logon mode at a user terminal:

- CTRL-C (CTRL-C CTRL-C under DG/RDOS) terminates the COBOL program running at that terminal. Pressing CR, NEW LINE, or ESC returns control to Logon.
- CTRL-A (CTRL-C CTRL-A under DG/RDOS) terminates the COBOL program running at that terminal. Pressing CR, NEW LINE, or ESC returns control to Logon.

Terminating the Runtime System

As a safety feature, the runtime system cannot be stopped unless all terminals under its control are in inactive status. The standard method of terminating the runtime system is with the Stop system call (#S), which can be executed only at the master terminal (see chapter 2 for details). The runtime system also can be terminated using program interrupts.

The /B global switch in the ICX command that starts the runtime system enables the operating system to interrupt the runtime system. If you are running in Logon mode, the operating system can interrupt the runtime system only at the master terminal.

Press CTRL-A under RDOS or CTRL-C CTRL-A under DG/RDOS at the master terminal to terminate the runtime system. This terminates all COBOL programs running in the ground, deactivates all terminals, and returns the master terminal to the CLI. Pressing CTRL-C (RDOS) or CTRL-C CTRL-C (DG/RDOS) at the master terminal has the same effect but also creates a *break file:* BREAK.SV for the background, FBREAK.SV for the foreground.

When either interrupt is used at the master terminal, the integrity of data files may be compromised. The situation is analogous to a system crash or power failure except that files need not be cleared with the CLI command CLEAR.

Runtime System Failure

In certain cases, the runtime system itself fails, causing control to return to the CLI. The runtime system automatically issues a .BREAK system call, which creates a break file, and displays a message. Before doing so, it places some diagnostic information in a certain section of main memory. The nature of this information and its location within the break file are described in the ECLIPSE RDOS Interactive COBOL Release Notice.

Operation of the PASS Queue

The Interactive COBOL runtime system contains its own spooling system called PASS. Spooled output is not printed automatically; you use PASS to select the files for printing. PASS is implemented as part of the runtime system (unless you use the /P switch when bringing up the runtime system) and is called either through Logon or with the system call:

```
CALL PROGRAM "#P".
```

If spooling has been disabled during runtime system operation, it can be reenabled during the printing of a PASS queue file with PASS's S option. To reenable spooling after the runtime system is terminated, use the CLI command SPEBL for each printer.

Assigning Files to the PASS Queue

If a program's SELECT clause assigns a file to PRINTER or PRINTER-1 along with the name of a disk file, the filename is entered in an unordered list called the PASS queue. In this list, each filename is assigned a number, to save you the effort of typing filenames when using PASS. You may enter up to 46 filenames in the PASS queue. If a program tries to add a filename to a full PASS queue, an I/O error occurs: FILE STATUS = 99. Each ground has its own PASS queue:

Background: ICX.LP
Foreground: FICX.LP

Option	PASS Function
Delete file (D)	Delete the file from disk storage and remove its name from the PASS queue.
Print file (P)	Print the file on its assigned system printer. You are prompted to enter the number of copies (default = 1). At this additional prompt, pressing ESC cancels the Print choice.
Remove file (R)	Remove the filename from the PASS queue, but retain the file in disk storage.
Select print options (S)	You are prompted to enter: 1) Starting page number (default = 1) 2) Ending page number (default = 9999) 3) RDOS spooling during print (Yes or No; default = No) 4) Number of copies (default = 1) 5) Printer (0 or 1; default = assigned printer) At these additional prompts, pressing ESC cancels the Select choice. The initial spooling state is reset after completion of each print job.
Terminate printing (T)	Interrupts the printing of the file. The runtime system displays the message JOB ABORTED BY OPERATOR.

Table 3-2 PASS Functions

PASS Control

When you start PASS execution at the master terminal, the PASS display appears on the screen with the following information displayed for each file in the queue:

- The printer to which the file has been assigned ($LPT or $LPT1). The SELECT PRINT OPTIONS command allows printout to be routed to a different printer; the default remains the same. The device name blinks when printing is in progress.
- The number that PASS has assigned to the file. An asterisk before this number indicates that the file is in use and may not be printed at this time.
- The RDOS or DG/RDOS filename.
- The line number of the terminal at which the file was created.

The PASS display is not automatically updated when a program enters a new file in the queue or when a file finishes printing. The display is updated whenever you press CR or NEW LINE while using PASS.

PASS provides print, select, delete, remove, and terminate functions, which you select by means of a screen prompt. Table 3-2 describes each of these functions.

Pressing ESC at the function prompt terminates PASS execution and returns control to Logon. File(s) currently being spooled to the system printer(s) finish printing, even though they are unattended.

To select the file for processing, enter the file's number at the ENTER FILE NUMBER prompt.

Foreground Termination

The runtime system does not disable the foreground-interrupt facility. At the system console, pressing CTRL-F (RDOS) or CTRL-C CTRL-F (DG/RDOS) at any time terminates the foreground program. If the runtime system is running in the foreground, this has the same effect as pressing CTRL-A (RDOS) or CTRL-C CTRL-A (DG/RDOS) at the foreground master terminal.

Key	Action
DEL	Replaces the character preceding the cursor with a space and moves the cursor back to that space. The tone sounds if the cursor is at the beginning of the field.
Left-arrow	Moves the cursor backward one character. The tone sounds if the cursor is at the beginning of the field.
Right-arrow	Moves the cursor forward one character. The tone sounds if the cursor moves beyond the last character in the field.
ERASE EOL	Fills the field with underscores and places the cursor at the beginning of the field.
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. The cursor cannot be moved past an underscore, although a character can be typed over it. In effect, an underscore erases to the end of the input field.

Table 3-3 Editing Keys

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 ends the ACCEPT statement.

Note: The term *input field* refers to TO fields, USING fields, and FROM/TO 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.

During execution of an ACCEPT statement, 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. These codes are stored in a data-item named ESCAPE KEY.
- It validates the contents of input fields against corresponding PICTURE statements.

Input Field Editing

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

The Up-Arrow Key

This key moves the cursor to the previous input field. The contents of the field are moved to screen storage, and the ACCEPT statement remains in effect. When used at the first input field, the up-arrow key returns the cursor to the beginning of this field and sounds the tone.

Input Field Termination

The runtime system recognizes the following field terminator keys: NEW LINE, CR, TAB, down-arrow, ESC, function keys alone (see exceptions below), function keys in combination with CTRL, and function keys in combination with SHIFT. 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. When the runtime system executes an ACCEPT *screen-name* statement, it places the cursor at the beginning of the first field defined in the screen-name. When an input field is terminated:

1. The runtime system validates the field. If ESC was pressed, the runtime system does not validate the field.
2. If the contents of the input field correspond to the PICTURE clause, the runtime system moves the contents of the field to screen storage. If ESC was pressed, the contents of the input field are not moved to screen storage (however, the contents of screen storage are moved to Working-Storage upon completion of the ACCEPT).
3. The runtime system places a two-digit code in the data-item ESCAPE KEY to indicate which terminator key was pressed.
4. If ESC or a function key was pressed, the ACCEPT statement terminates immediately, even if some of the input fields have not been processed. If the ACCEPT statement has been coded with ON ESCAPE *imperative-statement,* the imperative statement is executed. Moves from screen storage to Working-Storage still occur, even for fields that did not receive input.
5. If NEW LINE, CR, TAB, or down-arrow is pressed, the runtime system places the cursor at the beginning of the next input field.
6. After a terminator is pressed in the final input field, the contents of screen storage are moved to the corresponding File, Working-Storage, or Linkage section data-item.
7. The program continues execution with the next statement.

The ESC key

ESC does not cause the contents of the current input field to be moved to screen storage. However, contents of previous input fields are still moved to screen storage. The ACCEPT statement terminates immediately, and contents of all data-items referenced by the ACCEPT are moved from screen storage to the corresponding TO or USING items. The ON ESCAPE statement, if any, is invoked. Although the contents of the current input field are not moved to screen storage, moves from screen storage to File, Working-Storage, or Linkage data-items are still made.

Function Keys

The function keys do not cancel the current input field. They move the contents of the current field to screen storage, terminate the ACCEPT statement, and invoke the ON ESCAPE statement, if it is present.

Single-Field Screens

If a screen has a single input field, the terminator keys treat it as both the first and final input field.

Implementing the Function Keys

As the runtime system executes an ACCEPT statement, it records the function key pressed as a two-digit code in the data item ESCAPE KEY. Use of non-function-key terminators is also recorded; they all generate the same code, 00 (see Table 3-4).

	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

*f9 is not a terminator; it enters a value of minus.

**If your keyboard has the ENTER and ENTER- keys, these keys are field terminators that return the same values as f10 and f11, respectively.

Table 3-4 Terminator ESCAPE KEY Codes

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

1. In the ACCEPT statement that handles input, code an ON ESCAPE *imperative-statement* clause. This clause is executed only if you use a function key or ESC to terminate the ACCEPT.
2. Code the 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, but must first be loaded into a data-item using an ACCEPT id FROM ESCAPE KEY statement; for example:

```
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 ...
```

The ON ESCAPE clause is not essential. The statement following the original ACCEPT can perform the ESCAPE KEY processing.

Table 3-4 lists the two-digit codes loaded into ESCAPE KEY during execution of ACCEPT statements.

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 sections.

If the field does not match the PICTURE, the runtime system displays an error message at the bottom of the screen and positions the cursor 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 on a file, it updates a two-character File Status data-item for the file. This data-item must be defined with PIC XX in the Working-Storage Section and declared in the SELECT entry. A list of File Status codes and their meanings appears in appendix A.

Data-Items Maintained by the Runtime System

The runtime system maintains data-items that record such information as the current date, time, and terminal number. A COBOL program cannot access these system data-items directly. To examine a system data-item, set up a receiving data-item in the Working-Storage Section, pass the value of the system data-item to the programmer-defined data-item in Working-Storage, and examine the programmer-defined data-item.

A statement of the following form moves the contents of a system data-item to a programmer-defined data-item:

```
                  ⎧ DATE             ⎫
                  ⎪ DAY              ⎪
                  ⎪ TIME             ⎪
ACCEPT id FROM    ⎨ LINE NUMBER      ⎬
                  ⎪ USER NAME        ⎪
                  ⎪ ESCAPE KEY       ⎪
                  ⎩ EXCEPTION STATUS ⎭
```

Table 3-5 describes the system data-items and lists the PICTUREs required for the programmer-defined data-items that receive their values.

Data-Item	Required PIC	Description
DATE	9(6) or X(6)	The current date: yymmdd
DAY	9(5) or X(5)	The current day of the year: yyddd (ddd = 1,2,3,...,366)
TIME	9(8) or X(8)	The current time: hhmmss00 (The last two digits are always zeros.)
LINE NUMBER	999 or XXX	The identity of the terminal on which the program is running. Each ground's terminals are numbered 000 (master terminal), 001, 002, etc.
ESCAPE KEY	99	The two-digit code generated by a termination key.
EXCEPTION STATUS	999	A three-digit code that identifies the type of exception condition that has occurred during the execution of a CALL or CALL PROGRAM statement.
USER NAME	X(15)	All blanks. (Under AOS and AOS/VS, the user name associated with the process is returned.)

Table 3-5 Data-Items Maintained by the Runtime System

Exception Status

The runtime system maintains a three-digit data-item named EXCEPTION STATUS that records the result of the most recent CALL or CALL PROGRAM statement.

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, then branches on this 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 statement must immediately follow the CALL or CALL PROGRAM statement, for example:

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

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 the CALL or CALL PROGRAM statement has failed.

See appendix A for a list of Exception Status codes and their meanings.

Chapter 4
The Interactive COBOL Compiler

The Interactive COBOL compiler transforms a COBOL source program into object code that can be processed by the runtime system. This single-pass compiler can process source files in CRT or card format (with or without line numbers) and with indexed or sequential file organization.

The compiler produces up to four output files:

- Two object files constituting the executable program:

  source-file.DD, which contains the Data Division
  source-file.PD, which contains the Procedure Division
- An optional listing file
- An optional error file

The Command Line

The compiler is a program named ICOBOL.SV. It also has an overlay portion, ICOBOL.OL. It is invoked with a CLI command. As with any CLI command, the compilation command can be included in a CLI macro (.MC file). The command line has the following format:

```
ICOBOL[/global-sw] source-file[/I] [listfile/L[/P]] [error-file/E[/P]]
```

Source-file is the name of the RDOS or DG/RDOS file containing the COBOL source code. The compiler looks for the source file using the following algorithm: First it looks for *source-file*. Then it looks for *source-file*.SR. Last, it looks for *source-file*.CO. If you include the local /I switch, do not use an extension with *source-file*.

Listfile names an optional compilation listing file. Any filename extension can be used. The global switch /L may also be used to obtain a listing file having the standard .LS extension. To send a compilation listing directly to a system printer, specify $LPT or $LPT1 as the listing file.

Error-file names an optional error file. Any filename extension may be used. To send an error listing directly to a system printer, specify $LPT or $LPT1 as the error file.

Note that the listing and error files cannot be the same unless they are both sent directly to a system printer.

Global Switches

/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. If relative files have more than 9999 records, the data-name for the relative key must be declared PIC 9(5) COMP to PIC 9(18) COMP.

/C Card format source. The compiler ignores all characters beyond column 72 of a card-format source line. With CRT format, the maximum line length is 132 characters. Absence of this switch indicates CRT format.

/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.

/E Compiler statistics and error messages are not sent to the screen. This switch is not required if error messages are directed to an error file (see local /E switch).

/I Allows ICOS cross-development. With this switch, the .PD and .DD files are code revision 4. The current code revision is 6. The data-name for a relative key must be declared PIC 9(4) COMP. You can use only the following new features with this switch: level 88, PERFORM...AFTER, abbreviated combined relation conditions, and the compiler /O and /R switches. Other features that are listed in the "Changes to Interactive COBOL" page in the front of this manual cannot be used.

/L Use *source-file*.LS as the compilation listing file. This switch is overridden by a listing file argument *listfile*/L.

/N Do not produce an object program. In the absence of this switch, the compiler produces two object files: *sourcefile*.DD and *sourcefile*.PD. This switch cannot be used with the /U switch.

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

/R Rounds the .PD file up to a 2-KB boundary.

/S Append the symbol table, compiler statistics, and additional data to the listing file.

/U Append a decompilation of the object program to the listing file. This switch cannot be used with /N.

/W Suppresses warning messages in the listing file.

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

Local Switches

/I The source-file has indexed organization, as produced by ICEDIT (*source-file*.NX and *source-file*.XD). Specify the source-file without any file-name extension. Use the global switch /C in conjunction with this switch to indicate card-format source code.

/E Identifies the preceding filename as the error file.

/L Identifies the preceding filename as the compilation listing file.

/P Purges the existing contents of the listing file or the error file indicated in the command. Without this switch, the listing is appended to the existing file contents.

If the compiler finds any errors in the command line, it terminates execution and displays an error message. Error messages are listed in Appendix A. Other errors that may occur at initialization (e.g., specifying a read-protected file as the source-file) cause termination with the appropriate system error message.

Source Listing

The source program is listed to the file specified with the local /L switch. 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 command 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

Note: The operating system reports the data and procedure sizes and the total program size in blocks; each block is 512 bytes.

If you request a listing file and use the /S switch, the following additional statistics are 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 statements refer to data-items. The maximum number of data references is 2294.
- Procedure names: the number of procedure names in the program
- The symbol name, byte address of .DD storage, length, and data reference table index number for each symbol

Compiler Error Messages

Error messages are written to the error file following a line containing a program statement in which an error is detected. If the error file is the same as the listing file, then 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.

Appendix A lists and defines the compiler error messages.

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. With the debugger you can:

- 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, insert procedure names 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 switch /D:

   ```
   ICOBOL/D [additional switches] program-name [arguments]
   ```

   The /D switch 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 may be set and displayed.
3. Debug the program. To debug the program you can do one of the following:
   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. Invoke the runtime system in CLI mode with the debugger option (/D global switch):

      ```
      ICX/D[global-switches] prog-name[local-switches]
      ```

      This option is for a single user at the master console.

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 (!). If you invoke the debugger without first compiling the program with the global /D switch, the program simply executes.

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. Thus, you can use the standard input-editing keys listed in Table 5-1.

Key	Action
DEL	Delete previous character
Left-arrow	Move left one character
Right-arrow	Move right one character
Up-arrow	Return to beginning of field
ERASE EOL	Erase entire field
Underscore	Erase to end of field when terminator is pressed

Table 5-1 Debugger Editing Keys

Use any of the following keys to enter and execute a debugging command: CR, NEW LINE, TAB, down-arrow, all function keys. Pressing ESC cancels the command and redisplays the debugger's prompt.

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. Appendix A contains a list of debugger error messages and their meanings.

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 when the program commits a fatal error, when it comes to a STOP RUN statement, or when you issue the debugger command STOP. In these cases, control returns to Logon (or the CLI, if the debugger has been invoked from the CLI) after you press CR, NEW LINE, or ESC.

Once a program is debugged, recompile it without the /D switch. There no longer is any need for the symbol table, which adds substantially to the object program size. Also, delete any extra paragraph names that may have been 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. PROGA issues a CALL PROGRAM to PROGB. PROGB (but not PROGA) has been compiled with the global /D switch. Executing the debugger on PROGA executes PROGA and debugs PROGB.

If an #L call (or any other operation that deactivates the terminal) is encountered, the debugger is interrupted.

Debugger Commands

The following 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 does not change the PERFORM stack before beginning execution. Thus, this command may cause unanticipated results if used in the middle of a PERFORM sequence.

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

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

RUN *procedure-name* START
: Runs the program, starting where execution was last suspended and continuing 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, displaying the message `STOP RUN`. Pressing ESC, CR, or NEW LINE returns control to Logon or, if in CLI mode, to the CLI.

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 (if any) at the beginning and end of the specified procedure. UNCLEAR does not recover from 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)
: Pressing NEW LINE alone is equivalent to the DISPLAY command. 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 *id* is omitted, the most recently referenced data-item is used. If *lit* is a nonnumeric literal, it must be enclosed in quotation marks.

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. Use the START statement 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) Attempt to WRITE or REWRITE a record that would create a duplicate primary key. (2) Attempt 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	No disk space is available to write a new record.
91	OPEN error: (1) An OPEN statement referenced a nonexistent file, a file already opened, or a file with an illegal name. (2) A CLOSE statement referenced a file that had not been opened.
92	Access mode error: (1) File not opened. (2) WRITE or DELETE attempted for 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.
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	Partition or subdirectory not initialized.
97	Maximum number of OPEN files exceeded.
98	(1) Not enough disk space is available to create a contiguous file. (2) Attempt to write more than 65,534 records to a relative file. (3) I/O error on reading a link. ISAM file may be corrupt. The file can be checked with the ISAMVERIFY utility.
9A	File description inconsistency. Record length, key length, or key positions specified in program does not agree with data file.
9B	(1) After a successful OPEN of an ISAM file, the runtime system has detected possible corruption in the file. Close the file; this sets the reliability flags and prevent further access to the file. (2) Data (.XD) portion of an indexed or relative file is full. The reliability flags are set.
9C	Index (.NX) portion of an indexed or relative file is full. The reliability flags are not set.

Code	Meaning
9D	Too few blocks specified in the INDEX SIZE or DATA SIZE clauses of the SELECT entry.
9E	The record lock limit has been exceeded.
9F	Possible corruption of an ISAM file has been detected on attempting to OPEN the file, i.e., the reliability flags have been set.

Exception Status Codes

Code	Meaning
000	Successful CALL or CALL PROGRAM execution
001	A "#Cfilename" used an invalid filename.
013	The file in a "#Cfilename" is not open (i.e., it does not need to be physically closed).
048	(1) A "#C" was issued but no files can be closed. (2) A "#Cfilename" was issued for a file in use.
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.
205	Error in processing of the temporary file that passes information to the Linkage Section of a called program.
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, partition, or contiguous space in a system file during a program call. 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.
212	No tasks available on #O.
213	Called program cannot be loaded.
214	Error in processing .VM file.

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.

A MAPPED ECLIPSE RUNNING RDOS IS REQUIRED

Interactive COBOL requires the ECLIPSE processor.

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 PIC 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 on the screen at the end of compiling a program that contains at least one error. This message appears even if you specified the /E switch in the compiler command line.

COMPILER PANIC

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

COMPUTE STATEMENT

Illegal syntax for a COMPUTE statement.

COMPUTER-NAME

The computer name is illegal.

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 DESCRIPTION CLAUSE DUPLICATE

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

DATA DIVISION OVERFLOW

Size of the data 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 entry, but have not been specified in the Procedure Division Using list. These names are all flagged following the Procedure Division Using statement.
3. Data-names are referenced in the Procedure Division and defined in the Linkage Section but are not used in the Procedure Division Using statement.

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 in a SELECT clause is not DISK, PRINTER, PRINTER-1, KEYBOARD, or DISPLAY.

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 LOCAL SWITCH

The /L or /E switch appears with more than one file, or more than one source-file is specified.

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 must be 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 continues to be read and listed.

FD DUPLICATE: file-name

An FD has 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: file-name

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

FILE NAME

A file-name contains an illegal character.

FILE NAME AMBIGUOUS

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

FILE NAME UNDEFINED: file-name

No SELECT clause exists for this file-name.

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 Identification 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

Illegal local switch in the command line. The legal switches are /I, /E, /L, and /P.

ILLEGAL SEPARATOR

A separator was not followed by a space, or an illegal separator was used.

ILLEGAL SWITCH COMBINATION /A/I

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

ILLEGAL SWITCH COMBINATION /N/U

These two switches may not be used together in the 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 was expected but was not found.

INPUT FILE IS NOT ISAM

The specified file is not an indexed file, is not in the disk directory, or was specified with a filename extension.

INSPECT STATEMENT

Illegal syntax for an INSPECT statement.

INSUFFICIENT EXTENDED MEMORY IS AVAILABLE

If more memory is available in the other ground, use the SMEM command to reallocate it.

INSUFFICIENT LOGICAL MEMORY IS AVAILABLE

The maximum amount of logical memory available is 64 KB.

INTEGER

1. The PICTURE is numeric but the value for the data-item contains a nonnumeric item.
2. 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., READ from a printer.

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 AN 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 or 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

Data has been defined past line 255 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 (file-name, 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 ITEMS 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.

NO SOURCE FILE SPECIFIED

A file without switches or a file with the local /I switch must appear in the command line.

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 was not found.

NUMERIC OR NUMERIC EDITED OPERAND REQUIRED

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

OCCURS CLAUSE

Illegal syntax for an OCCURS clause.

OCCURS DEPTH

OCCURS clauses are permitted only to a depth of 3.

OCCURS NOT PERMITTED AT LEVEL 01 OR 77

OCCURS can be specified only from level 02 to level 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 was 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 contains no 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 entry, 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 to 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 (.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 OUTSIDE LIMITS OF CONTAINS CLAUSE

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

RECORD SIZE EXCEEDS DEVICE LIMITATION

Files asigned to PRINTER, PRINTER-1, KEYBOARD, or DISPLAY may have a maximum record size of 132 characters; indexed or relative files assigned to DISK may have a maximum record size of 4096 characters.

RECORDING MODE

The RECORDING MODE may be specified only for disk files; the VARIABLE clause is permitted only 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 will probably 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 cannot be used as the identifier in a statement of the form ACCEPT id FROM. . . .

SECTION MISSING

A section header was expected but was not found.

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.

SOURCE FILE NOT FOUND

The specified source-file is not in the disk directory.

START STATEMENT

Illegal syntax for a START statement.

STOP STATEMENT

Illegal syntax for a STOP statement.

SUBSCRIPT

A syntax error exists in a subscript specification; for example, the closing parenthesis is omitted.

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.

THE CHARACTER INSTRUCTION SET IS REQUIRED

Interactive COBOL requires the ECLIPSE character instruction set.

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 FOR REDEFINES

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

VALUE NOT PERMITTED IN FILE SECTION

A VALUE clause may not be used in the File Section.

VALUE NOT PERMITTED IN LINKAGE SECTION

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

VALUE NOT PERMITTED IN OCCURS

A VALUE clause may not be used in an item with an OCCURS clause or an item subordinate to an OCCURS 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., file-name, 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 syntactual errors have been made.

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 appears:

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 generated by the runtime system. The errors may occur because of problems in starting the runtime system, a failure in the runtime system, or a fatal error in an Interactive COBOL program.

Starting the Runtime System

Channels computed (based on files & terminals) exceed ground maximum

Duplicate switch setting

Extra channels must be from 0 to 236

Failed trying to read ICX.LD file

Delete the file ICX.LD, invoke the runtime system, and run the utility DEFLINES to create a new ICX.LD file.

First argument must be COBOL program name

If you are running in CLI mode, ICX/C or ICX/D must be followed by the Interactive COBOL program-name.

ICX must be run on a mapped Eclipse running RDOS

ICX required the Character Instruction Set

Insufficient extended memory is available

Try a new combination of the parameters in which any or all of *size, files,* and *tasks* are smaller. Alternately, if more memory is available in the other ground, use the SMEM command to reallocate it.

Insufficient logical memory is available

The amount of memory (in bytes) that needs to be allocated to logical address space is calculated by the runtime system. This calculation must be less than or equal to 65,536 bytes (i.e., 64 KB, the logical address space limit). Try a new combination of the parameters in which any or all of *size, files,* and *tasks* are smaller. *Size* has the greatest effect on the required logical address space.

Invalid argument value specified

Not a legal COBOL program file

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

Number of files must be from 4 to 64

Number of program tasks must be from 1 to 33

Program Not Found

No file having the program name has been located.

Program Too Large

The specified program exceeds the program size limit.

Revision Incompatibility

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

Program size must be from 3 to 31 kilobytes

Program size must be an odd number of kilobytes

Unrecognizable switch setting

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 break file 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 break file. Otherwise, save the break file, record the error message, and contact your DG representative.

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

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 has occurred, and your program has no USE procedure to handle it.

```
Index Register Overflow. COBOL PC =
```

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

```
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 the interrupt control characters during program 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 =
```

You have attempted 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 (RDOS, DG/RDOS) **069-705020**

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, which is 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.

JOBS User's Guide **055-000042**

Describes the Job Organization Batch Stream utility. JOBS places CLI macros and Interactive COBOL programs on a queue to be executed at the end of the day. The manual describes how to use JOBS and illustrates how it can be applied to typical situations.

RDOS and DG/RDOS Documents

Introduction to RDOS **069-400011**

Introduces RDOS concepts to readers who are unfamiliar with the operating system and its capabilities.

How to Load and Generate RDOS **069-400013**

Guides the reader step by step through the RDOS system generation process. The manual serves the first-time user and the user who wants to generate a system tailored to specific requirements.

RDOS/DOS Command Line Interpreter **069-400015**

Introduces the command line interpreter (CLI) and describes its operations and advanced functions. It also highlights the features and operating procedures of the batch monitor.

Using DG/RDOS on DESKTOP GENERATION Systems **069-000056**

Explains how to install and operate DG/RDOS software on your system.

RDOS/DOS Debugging Utilities **069-400020**

Describes commands and procedures for using the symbolic debugger (DEBUG). DEBUG helps you detect, locate, and remove program errors.

Index

.AX file 1-2
.CO file 1-2
.CU file 1-2
.DD file 1-2, 4-1
.DL file 1-2
.DR file 1-2
.DX file 1-2
.LJ file 1-2
.LS file 1-2
.MC file 1-2
.NX file 1-2, 1-12
.OL file 1-2
.PD file 1-2, 4-1
  rounding up to 2-KB boundary 4-2
.QK file 1-2
.RB file 1-2
.SR file 1-2
.SS file 1-2
.SV file 1-2
.SX file 1-2
.TX file 1-2
.VM file 1-2, 3-3
.XD file 1-2, 1-12

A

Abort
  of COBOL program 2-2, 3-7
ACCEPT
  branching on fuction key 3-12
  editing input field 3-10
  implementing terminators with 3-11
  runtime system functions with 3-10
  system data-item 3-13
  with system call 2-5
ALMSPD.SR (stores line characteristics) 3-3
ANALYZE utility 1-13
ANSI 74 code 4-1
Assembler subroutine
  calling from COBOL program 2-6
  linking into runtime system 2-9

B

Background 3-1, 3-5
Blank line, in debugger command 5-4
Break file 3-8

C

CALL PROGRAM statement 2-1
  and Exception Status 3-14
  recovery from failing 3-14
  under debugger control 5-2
CALL statement 2-1
  and Exception Status 3-14
  calling an assembler subroutine 2-6
  recovery from failing 3-14
CDIR (Create directory command) 1-4
Chaining
  to Logon 2-2, 2-3
Channels 3-3
CHATR 1-8
CHLAT 1-8
CLEAR, debugger command 5-4
CLEAR procedure-name, debugger command 5-4
CLI
  mode of runtime system 3-4
  system call to return control to 2-2, 2-4
Closing a file
  system call to 2-2, 2-5
COBOL program
  branching on function key 3-12
  executing 3-5
  executing system call from 2-5
  interrupting 3-7
  running detached from terminal 3-7
  system call to abort 2-2, 3-7
  terminating 3-6
Commands, debugger 5-3
Compiler 4-1
  command line 4-1
  error messages A-2, 4-4
  files produced by 4-1
  source listing 4-2
  statistics 4-3
  warning messages 4-3
Contiguous files 1-12
Copy files 4-2
Cross-reference table 4-2

D

Data entry
  editing 3-10
  field validation 3-13
  function keys 3-11

Data entry (continued)
interactive 3-10
terminator keys 3-11
DATA SIZE clause 1-12
Data validation, of input fields 3-13
DATE 3-14
DAY 3-14
Debugger 5-1
#D system call 2-2, 2-3, 5-1
assembler 2-10
building runtime system with 2-9
COBOL program under control of 2-2, 2-3, 5-1
command line for 5-1
commands with 5-3
editing keys 5-2
error messages 16
program calls under control of 5-2
DEFLINES utility 3-3
Detached jobs, system call 3-7
Device filenames 1-10
DIR command 1-4
Directories 1-2
accessing files outside of 1-7
changing 1-4
creating 1-4
initializing 1-5
master 1-5
system call to initialize 1-1, 1-6, 2-2, 2-3, 2-5
system call to release 1-1, 1-6, 2-2, 2-4
working directory 1-4
Disk, mnemonic codes for 1-1
DISPLAY, debugger command 5-4
Display terminals, assigning files to 1-11
DKINIT 1-2

E

EQUIV 1-7
Equivalences 1-7
Error files 4-1
Error messages
compiler A-2, 4-4
data validation A-15
debugger A-16
Exception Status codes A-2
File Status codes A-1
runtime 17, 3-6
ESCAPE KEY 3-14
Exception Status 3-14
table of codes A-2

F

File access
outside working directory 1-6
restrictions on 1-7
File characteristic codes 1-3, 1-4
File Status 3-13
table of codes A-1
Filenames
assigning to primary partitions 1-7
conventions for 1-1
default extensions for 1-2
default external names 1-8
external 1-8
internal 1-8
of devices 1-10
Files
allocation of 1-12
assigning default output to 2-2, 2-4
assigning to disk 1-9
attribute-protected 1-8
attributes of 1-7
break 3-8
command 1-2
contiguous 1-12
copy 4-2
created with ICEDIT 1-2
error 4-1
ISAM 1-2, 1-12
listing 1-2, 4-1
macro 1-2
object 1-2, 4-1, 4-2
of runtime system 3-1
opening exclusively 1-13
overlay 1-2
permanent 1-8
preventing link access 1-8
primary partition 2-3
random 1-12
read-protected 1-8
relocatable binary 1-2
renaming 2-2, 2-4
resolution 1-7
save 1-2, 1-8
source (card) 1-2, 4-1
source (CRT) 1-2
system call to close 2-2, 2-5
text 1-2
virtual memory 1-2, 3-3
write-protected 1-8
FILESTATS utility 1-13
FLEXSTATS utility 2-9, 3-2
Foreground 3-1, 3-5
termination of 3-9
Function keys, implementing with ACCEPT 3-11

I

ICEDIT 4-2
files created 1-2
ICOS, cross-development 4-2
INDEX SIZE clause 1-12

INIT 1-5
/F switch 2-3
Initialization
of partitions 2-2, 2-3
Initialize a directory
CLI command to 1-5
system call to 1-1, 1-6, 2-2, 2-3, 2-5
Interrupts 3-7
ISAM files 1-2
contiguous allocation of 1-12
portions of 1-12
runtime system opens exclusively 1-13
source 4-2

L

Line characteristics, setting 3-3
LINE NUMBER 3-14
Link files, restricting access to 1-8
Links 1-7
preventing access through 1-8
LIST, debugger command 5-4
LIST END, debugger command 5-4
Listing files 1-2, 4-1
LJE command file 1-2
Logon
executing system call from 2-5
mode of runtime system 3-5
program 3-6
system call to chain to 2-2, 2-3, 3-7

M

Magnetic tape, assigning files to 1-11
MAP.DR 2-3
Master terminal 2-1, 3-7
interrupts used at 3-8
Memory
range required by runtime system 3-3
restrictions 3-3
Messages
system call to send 2-2, 2-4
warning 4-2
Monitors, assigning files to 1-11

O

Object files 4-1
rounding .PD 4-2
Opening files
concurrently 3-3
exclusively 1-13
Output
system call to assign default output to a file 2-2, 2-4
Overlay files 1-2

P

Partitions
assigning filenames to 1-7
file extensions with 1-2
full initialization of 2-2, 2-3
primary 1-2
secondary 1-3
PASS 1-10, 1-11, 3-8
system call to run 3-7
system call to use 2-2, 2-4
Pause
system call to 2-2, 2-5
Printer
default filename for 1-10
spooling to 1-11
Printer access scheduling system, see PASS

R

Random files 1-12
Release a directory
system call to 1-1, 1-6, 2-2, 2-4
Renaming files
system call to 2-2, 2-4
REORG utility 1-13
RUN, debugger command 5-3
RUN END, debugger command 5-3
RUN procedure-name, debugger command 5-3
RUN procedure-name END, debugger command 5-3
RUN procedure-name START, debugger command 5-3
RUN START, debugger command 5-3
Runtime system 3-1
and opening files 1-13
channel limits 3-3
CLI mode 3-4
data-items maintained by 3-13
debugging 2-10
error messages 17, 3-6
failure 2-3, 3-8
files 3-1
functions performed with ACCEPT 3-10
interrupts 3-8
linking assembler subroutine into 2-9
Logon mode 3-5
master terminal 3-7
memory required 3-3
restrictions 3-3
starting 3-4
status of terminals 3-7
system call to shut down 2-2, 2-4, 3-7
tailoring 3-2

SCREEN, file extensions 1-2
SELECT statement 1-8
Sequential files
  contiguous allocation of 1-12
  not opened exclusively 1-13
SET TO lit, debugger command 5-4
Source files 1-2
  indexed organization 4-2
Source listing, compiler 4-2
SPEBL (reenables spooling) 3-8

S

Save file 1-2, 1-8
Switches (compiler)
  /A (ANSI 74 code) 4-1
  /C (card format) 4-1
  /D (add symbol table) 4-2
  /E (error file) 4-2
  /E (error messages not displayed) 4-2
  /I (ICOS cross-development) 4-2
  /I (indexed source) 4-2
  /L (listing file) 4-2
  /N (no object code) 4-2
  /O (suppress copy file listing) 4-2
  /P (purge listing or error file) 4-2
  /R (round up .PD) 4-2
  /S (add statistics to listing) 4-2
  /U (add decompilation to listing) 4-2
  /W (suppress warnings in listing) 4-2
  /X (add cross-reference to listing) 4-2
Symbol table 4-2
SYS.DR 2-3
System calls 2-1
  #A 2-2, 3-7
  #C 2-2 2-5
  #D 2-2, 2-3, 5-1
  #F 1-1, 2-2, 2-3
  #H 2-2, 2-3
  #I 1-1, 1-6, 2-2, 2-3, 2-5
  #L 2-2, 2-3, 3-7
  #M 2-2, 2-4
  #N 2-2, 2-4
  #O 2-2, 2-4, 3-7
  #P 2-2, 2-4, 3-7
  #R 1-1, 1-6, 2-2, 2-4
  #S 2-2, 2-4, 3-7
  #T 2-2, 2-5, 3-7
  #W 2-2, 2-5
  errors in 2-6

T

Terminal
  line characteristics 3-3
  master 2-1, 3-7, 3-8
  placing under background or foreground control 3-5
  setting line characteristics 3-3
  status 3-7
  system call to deactivate 2-2, 2-3
  system call to display status of 2-2, 2-5, 3-7
  system call to send message to 2-2, 2-4
Terminators
  codes 3-12
  implementing with ACCEPT 3-11
TIME 3-14
TRAP procedure-name, debugger command 5-4
TRAP procedure-name END, debugger command 5-4

U

UNCLEAR, debugger command 5-4
USER NAME 3-14

V

Validation, of input fields 3-13
Virtual memory file 1-2, 3-3

W

Warning messages 4-2
  compiler 4-3

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		
Other (Specify) _____		

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

CUT ALONG DOTTED LINE

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

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.

Data General

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

User Documentation Remarks Form

Your Name _____ Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title _____ Manual No. _____

Who are you?

☐ EDP Manager
☐ Senior Systems Analyst
☐ Analyst/Programmer
☐ Operator
☐ Other _____

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)* _____

___ Introduction to the product
___ Reference
___ Tutorial Text
___ Operating Guide
___ Other _____

About the manual:

	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

134-664

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

User Documentation, M.S. E-111
4400 Computer Drive
Westborough, Massachusetts 01581

User Documentation Remarks Form

Your Name _____ Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title _____ Manual No. _____

Who are you? ☐EDP Manager ☐Analyst/Programmer ☐Other _____
☐Senior Systems Analyst ☐Operator _____

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)* _____

___ Introduction to the product ___ Tutorial Text ___ Other
___ Reference ___ Operating Guide _____

About the manual:	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

User Documentation, M.S. E-111

4400 Computer Drive

Westborough, Massachusetts 01581

Data General Corporation, Westboro, MA 01580

069-705014-02