DataGeneral

Customer Documentation

Office Copy

Interactive COBOL Utilities (AOS, AOS/VS)

Interactive COBOL Utilities (AOS, AOS/VS)

069-705021-01

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

Ordering No. 069-705021

Printed in the United States of America
Revision 01, April 1987

NOTICE

Interactive COBOL Utilities (AOS, AOS/VS)
069-705021-01

Revision History:
Original Release — June 1983
Addendum 086-000072-00 — August 1984
First Revision — April 1987

Effective with:
(Interactive COBOL, Rev. 1.30)

The content and change indicators in this revision are unchanged from 069-705021-00. This revision incorporates addendum pages only.

Changes to Interactive COBOL

For AOS and AOS/VS users, Revision 1.10 of Interactive COBOL contains the following changes in utilities from Revision 1.00:

An ISAM reliability package has been added, which helps insure the logical structure of ISAM files. It prevents the utilities ANALYZE, COLLAPSE, CSSORT, ICEDIT, certain REORG functions, and the runtime system from accessing ISAM files that are flagged as corrupt. The utility ISAMVERIFY diagnoses where the corruption occurs. The utility REBUILD or REORG can fix the file.

A new utility, REBUILD, fixes an ISAM file if one of its portions is bad. It also allows the user to correct the file size so that it is on a 2KB boundary, which is required under AOS and AOS/VS.

REORG has increased functionality in two main areas. First, it enables the user to declare the .XD header information as invalid (/X switch) and to supply the correct information in the command line. Second, the /Q switch enables the user to operate the utility on a relative file without renumbering the records.

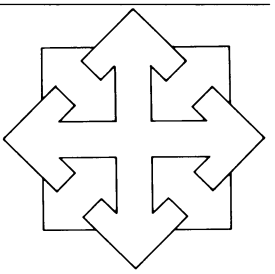
A new utility, SETFORMS, allows users with Data Royal printers to set variable forms lengths and vertical tabbing for special forms.

Contents

Appendix A
Switch Combinations for REORG

Related Documents

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.

Three system-dependent *User's Guides* explain the features of the user's particular operating system — MP/OS, 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 also includes two special COBOL source editors. *IC/EDIT: Interactive COBOL Editor* describes an editor specifically designed for writing programs. *SCREEN: Screen Format Editor* describes the special-purpose editor for designing and automatically coding screen display formats.

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

Scope

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

Organization

The manual is divided into two chapters: an introduction and an alphabetized reference. The introduction provides a summary of the Interactive COBOL utilities and explains the contexts in which they are used. The reference chapter provides detailed operating instructions for the programs ANALYZE, CALC, COLLAPSE, CREV, CSSORT, FILESTATS, ICINQUIRE, ISAMVERIFY, REBUILD, REORG, and SETFORMS. Numerous examples supplement this material. An index provides convenient access to topics.

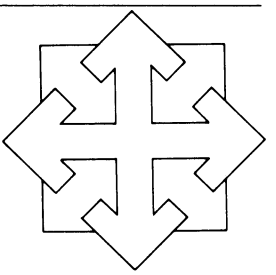
Notational Conventions

The following conventions are used in indicating the syntax of commands and user entries:

UPPERCASE	Commands and switches appear in uppercase.
lowercase	User-supplied arguments, such as filenames, appear in lowercase.
[ ]	Optional arguments are set in brackets; do not enter the brackets themselves.
{ }	One of the arguments in braces must be selected; do not enter the braces.
|	Vertical bars in the margin highlight technical changes made since the last revision of this document.
NL	Press the NEW LINE key.
CR	Press the carriage return key, CR.

In general, NL and CR may be used interchangeably to terminate commands and enter responses to prompts. However, under AOS and AOS/VS, CR deletes all characters from the point you enter it to the end of the line.

Chapter 1
An Overview of the Interactive COBOL Utilities



This chapter describes each utility briefly. The utilities are grouped by function. General instructions for executing the utilities appear after the summary descriptions. Comparisons of Commercial Systems (CS) utilities, AOS utilities, and Interactive COBOL utilities on AOS and AOS/VS appear at the end of this chapter. Complete descriptions of the utilities appear in chapter 2.

Designing and Processing Files

Interactive COBOL provides seven utilities to aid in designing and maintaining system files that meet data storage requirements. These utilities are ANALYZE, COLLAPSE, CSSORT, FILESTATS, ISAMVERIFY, REBUILD, and REORG.

ANALYZE

ANALYZE helps you monitor the growth of existing indexed and relative files. ANALYZE outputs current statistics for a given file, such as record length, the number of data records, and the number of index blocks. On the basis of these and other statistics, you may decide to streamline the file by repacking its index structure and/or physically removing logically deleted records, using the COLLAPSE or REORG utility. ANALYZE checks file integrity before operating on the file.

COLLAPSE

COLLAPSE enables you to improve indexed file access and the use of disk space, based upon FILESTATS and/or ANALYZE statistics. COLLAPSE physically removes logically deleted records from the data portion of the file and repacks the index portion to a specified density, or repacks the index only. No temporary disk space is required to perform these functions. COLLAPSE provides statistics for any changes made to a file.

CSSORT

CSSORT performs off-line sorting and merging of files. CSSORT accepts any file created by Interactive COBOL and produces a sequential file as output. CSSORT can sort by ascending values, descending values, or by a user-defined collating sequence. A sort operation can include reformatting of records. For a merge operation, CSSORT can accept up to six sequential files as input.

FILESTATS

FILESTATS calculates the size of any sequential, indexed, or relative file on disk. It calculates the total number of blocks a file would occupy on disk based on the input of

such file variables as number of records, record length, and key length. Use FILESTATS statistics to determine the optimal key or record length for an indexed file, or to ensure that file sizes can be maintained within the limits of a disk's storage capacity. Based upon FILESTATS output, you may decide to use COLLAPSE to adjust the density of the index structure of a file.

ISAMVERIFY

ISAMVERIFY tests the integrity of ISAM files. If a structural error occurs in a file, the utility notifies you via a screen display or, if specified, an audit file. ISAMVERIFY does not attempt to fix any errors, nor does it change the files, except for setting and clearing the reliability flags. If an error is found in either the .NX or the .XD portion of the file, run REBUILD on the file or substitute a backup file. If both portions of the file are corrupt, run REORG on the file or use a backup file.

REBUILD

REBUILD is used on corrupted ISAM files. It can rebuild either the .NX or .XD portion if the other portion is intact. REBUILD completely restores the integrity of the logical file structure of indexed and relative files, assuming that the AOS or AOS/VS file structure is intact.

REORG

REORG converts files from one type to another. REORG accepts as input any file created by an Interactive COBOL program and outputs files in formats you specify. For example, it can convert a file from indexed to sequential format or rearrange data in records to produce a different print format. It may also be used to physically delete records that have been logically deleted.

REORG is also used to reconstruct an ISAM file when both the .NX and the .XD portions of the file are corrupt.

Performing Calculations: CALC

CALC is a general purpose calculating aid that performs the arithmetic functions, including exponentiation and derivation of roots. It enables you to define variables. CALC automatically displays the five most recent results of your calculations.

Displaying Revision Levels: CREV

CREV displays the revision level of the compiler used on an Interactive COBOL source program, the operating system on which it was compiled, and the code revision.

Editing Files: ICEDIT

ICEDIT is a line-oriented source program editor designed for COBOL program development in the Interactive COBOL runtime environment. It allows developers to write, edit, and run Interactive COBOL programs. ICEDIT is documented in *ICEDIT: Interactive COBOL Editor*.

Processing Data Files: ICINQUIRE

ICINQUIRE is used to review, update, or create data files. It accepts a record descriptor as input and produces a tailored COBOL program (to be executed under the runtime system) that processes any ISAM file matching the record description. ICINQUIRE operations include forward and backward sequential reading, data record display, and updating and deleting records.

Formatting Screens: SCREEN

SCREEN is a tool for designing, coding, and documenting Interactive COBOL display screen formats. With SCREEN, you compose the literal and data fields exactly as you want them to appear to the program operator. Then you instruct SCREEN to write source code for the image. SCREEN-written source files may be used in any Interactive COBOL program. SCREEN is documented in *SCREEN: Screen Format Editor.*

Setting Vertical Forms: SETFORMS

This utility is for those who have the Data Royal printer, model IPS-5000-A, with option 190168. It allows the user to set variable forms length and vertical tabbing positions for special printing forms.

Executing the Utilities

An Interactive COBOL utility executes either under the runtime system or under the command line interpreter (CLI). Runtime utilities are ANALYZE, CALC, and FILESTATS. CLI utilities are COLLAPSE, CREV, CSSORT, ICINQUIRE, ISAMVERIFY, REBUILD, and REORG.

CLI Utilities

The CLI utilities are executed directly from the CLI. Enter the complete command line at the CLI prompt, as described in chapter 2. When the utility finishes executing, you are returned to the CLI.

Invoking Runtime Utilities

The runtime system is called from the CLI by the following command:

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

If a runtime utility is given as the program name in the command, the utility is executed directly from the CLI, without going through the Logon Menu. When execution stops, control returns to the CLI.

If you do not specify a program name in the command, the runtime system executes the Interactive COBOL Logon program, and the menu in Figure 1-1 is displayed.

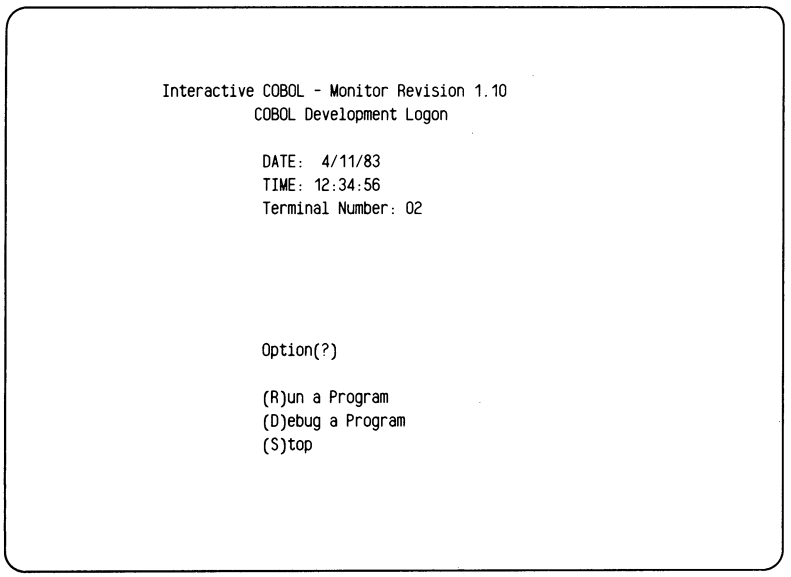


***Figure 1-1** AOS and AOS/VS Logon*

The runtime system displays an appropriate message if you run a program through Logon and it is interrupted by one of the following: a STOP RUN statement, a STOP command from the debugger, a console interrupt, or a program error. When you press NL or CR, the runtime system again executes the Logon program. You can stop the runtime system by issuing a CALL PROGRAM "#S", thus issuing the STOP command to Logon, or by using the CTRL-C CTRL-A sequence.

ISAM Reliability Package

Interactive COBOL's file reliability system helps insure the logical structure of ISAM files. The purpose of this reliability system is to detect possible file corruption and to prevent access of files that may be corrupt.

The data portion (.XD) of each ISAM file contains two reliability flags — one for the index portion (.NX) of the file, and one for the data portion. These flags can be set by the runtime system, COLLAPSE, ISAMVERIFY, and ANALYZE. When the flags in a file are set, only the diagnostic utility ISAMVERIFY, the restructuring utility REBUILD, and portions of the reorganization utility REORG can be run on the file.

If a power failure occurs while a file is open, the reliability flags remain set. Since the flags are set, the utilities ANALYZE, COLLAPSE, CSSORT, ICEDIT, certain functions of REORG, and the runtime system cannot access the file. If you attempt to access a file flagged as corrupt, a message appears warning that the logical file structure may be corrupt.

If this happens, first run ISAMVERIFY to determine what is wrong with the file. If it is corrupt, ISAMVERIFY reports the information necessary to run REBUILD or REORG. If the file is sound, ISAMVERIFY clears the reliability flags, which allows the file to be accessed.

Accessing Files

AOS and AOS/VS use a searchlist for file access. A *searchlist* is a list of directories that the system automatically searches when it cannot find a file in the current working directory and a pathname is not specified. Display or change your searchlist with the SEARCHLIST command.

The searchlist can contain up to eight directories. Since the Interactive COBOL utilities, compiler, and runtime system all add ":UTIL:ICOBOL" to the searchlist, you should specify no more than seven directories in the searchlist.

The link is a file access facility under AOS and AOS/VS. A *link* is a file that contains a pathname or a pathname segment. When a link name appears in a pathname, the system replaces the link name with the contents of the link file. You can create links with the command

```
CREATE/LINK linkname pathname
```

AOS and AOS/VS provide file security by means of an *access control list* (ACL). An ACL contains a list of users who have access to a given file and the type of access each user is permitted. There are five types of access: owner, write, append, read, and execute. The ACL command displays or resets a file's access control list. To execute a utility that accesses files, the file's ACLs must permit appropriate access.

For example, the utility ISAMVERIFY reads files and clears the reliability flags if the file is not corrupt. Therefore, to run ISAMVERIFY on a file, you need read access to the file so that ISAMVERIFY can read its contents, and write access so that ISAMVERIFY can set the reliability flags. However, you cannot change a file's ACL unless you have owner access or write access to the directory. For further information about file access concepts and procedures, see the *Command Line Interpreter User's Manual (AOS and AOS/VS)*.

Comparison of Utilities for AOS and AOS/VS

For the new user of AOS or AOS/VS who is familiar with CS utilities, Table 1-1 lists the CS utilities and identifies the comparable Interactive COBOL utilities and comparable AOS and AOS/VS utilities or CLI commands.

CS	Interactive COBOL	AOS, AOS/VS
ABORT	—	TERMINATE
ANALYZE	ANALYZE	—
BOOT	—	—
BURST	—	—
CALC	CALC	—
COLLAPSE	COLLAPSE	—
COPY	—	PCOPY
CREV	CREV	—
CSINIT	—	—
CSGEN	—	AOSGEN, VSGEN
DDUMP/DLOAD	—	—
DKINIT	—	DFMTR
DO	—	—
ENPAT/PATCH	—	PATCH
FDUMP/FLOAD	—	DUMP/LOAD
FILESTATS	FILESTATS	—
HASP II	—	HASP II
ICEDIT	ICEDIT	AOS editors
ICRJE80	—	RJE80
ICSCREEN	—	SCREEN
	ISAMVERIFY	—
INQUIRE	ICINQUIRE	—
JOB	—	—
LJE	—	QBATCH
LOGON	LOGON	—
MESSAGE	—	SEND
MESSAGES	—	—
NOTES	—	—
PASS	—	AOS batch utilities
	REBUILD	—
REORG	REORG	—
SCREEN	SCREEN	—
SORT/MERGE	CSSORT	—
STOP	STOP	—
TERMINAL STATUS	—	CLI macro
TYPE	—	TYPE
VINIT	—	LABEL
VSTAT	—	LOAD

Table 1-1 Utility Equivalency Guide

Chapter 2
Utility Reference

This chapter describes the functions of the Interactive COBOL utilities on AOS and AOS/VS, and gives instructions for their execution. The utilities are listed in alphabetical order.

ANALYZE *Analyzing Files*

ANALYZE is an Interactive COBOL program that measures the parameters of existing ISAM files. ANALYZE accepts as input any Interactive COBOL ISAM files, including files with alternate keys. It generates file statistics that help the programmer evaluate file structures in order to achieve optimum storage and retrieval. ANALYZE checks certain aspects of file integrity before it begins operating on the file.

ANALYZE runs under the control of the Interactive COBOL runtime system. The program is compatible with all other runtime system uses and program testing. Therefore, it can be executed at any terminal in an Interactive COBOL system and at several terminals at the same time. It allows several files to be analyzed simultaneously.

ANALYZE enables you to monitor file growth and structure. It gives feedback on file design, such as record and key sizes and the number of index levels for each record key. ANALYZE can indicate the need to run the COLLAPSE utility on a file. (COLLAPSE physically removes logically deleted records and/or packs the index structure up to 99% full. Each logically deleted record removed by COLLAPSE may be replaced by a new record.) ANALYZE reports the number of records removed by a previous COLLAPSE as `FREE-SLOTS`.

Output from ANALYZE, which can be sent to the display screen or the system printer, includes the following:

- The filename and the revision of ISAM under which the file was created
- Record length in bytes
- Number of data records, including those logically deleted
- Number of index blocks
- Percentage of index size to total file size
- Number of record slots freed by the most recent run of COLLAPSE
- Number of logically deleted data records (optional)
- An analysis of each primary key and alternate key including: key length in bytes, starting position of the key within the data record, number of keys per index block, number of index blocks, and structure tables (optional)

The index structure tables show for each record key (1) the number of used blocks in the index file; (2) the minimum, maximum, and average number of keys per block; and (3) the percentage of the index level that is full.

Procedure

To invoke ANALYZE at the CLI, enter **ANALYZE**. To invoke ANALYZE at the Logon menu, enter **ANALYZE** in response to the **RUN PROGRAM** prompt.

ANALYZE begins by displaying a series of prompts, one at a time, for which you specify options and filenames. Terminate each response with CR, which transmits the response and displays the next prompt. Pressing ESC in response to any of the prompts stops the program.

You are first prompted:

```
OUTPUT TO PRINTER (P) OR DISPLAY (D)?
```

Enter P to send the output to the system printer; in this case the results are not displayed. Output is queued to the printer. Enter D to display the results on the screen. If you select D, you are given the option of printing the results. ANALYZE then prompts:

```
INDEX TABLES - (Y) OR (N)?
```

The Y response creates a table for each key, reflecting the distribution of record keys through the index structure. The number of deleted keys is tallied. The next prompt appears:

```
RECORD DELETIONS - (Y) OR (N)?
```

The Y response produces a tally of deleted records. If ANALYZE is run on a file after a large number of records are deleted, it may report a number for deleted keys that is greater than the total number of records (including deletions) currently in the file. This is because when a REWRITE statement changes an alternate key, the old index entry is flagged for deletion and a new index entry is written (the data portion is updated in place).

Finally ANALYZE prompts for a file to be analyzed. Enter the name of a file, without the .NX or .XD extension. The name may be a simple filename or a pathname up to 28 characters. It may not include underscores. ANALYZE follows your searchlist if it does not find the file in the current directory. ANALYZE continues to accept filenames until you have entered 15 filenames or pressed CR alone at the prompt. The utility then checks the integrity of the files.

Reliability Checking

Before ANALYZE begins to operate on a file, it checks that the reliability flags are not set. It then checks the following boundary conditions in the data file:

- The number of alternate keys, which must be between 0 and 4
- The record length, which cannot exceed 4096 bytes
- The key length, which cannot exceed 100 bytes
- The key position, which must be 1 or greater
- The index depth, which must be 6 or less

If any of the above parameters is out of bounds, ANALYZE sets the reliability flags and displays the following message:

```
ERROR DETECTED DURING ANALYSIS:
BOUNDARY LIMIT EXCEEDED (NX): filename
```

When the reliability flags are set, ANALYZE displays only the following information on the file:

- Record length
- Number of keys (including alternates)
- Key length of each key
- Relative position of each key within the record

If the file is corrupt, the above information may be incorrect. Run ISAMVERIFY on the file, which will give you further information. ISAMVERIFY clears the reliability flag if it determines that the file is not corrupt. If the file is corrupt, ISAMVERIFY supplies the information necessary to determine whether you should run REBUILD or REORG, or use a backup file.

Example

In the following example, ANALYZE reports statistics for the indexed file PARTFILE. Assume that PARTFILE is not corrupt, that is, it meets ANALYZE's boundary condition checking and the reliability flags are clear. This example is continued in the FILESTATS and COLLAPSE sections to show the interrelationships of these utilities in data file maintenance and planning.

Assume that PARTFILE was initially planned to include 6000 records and occupy 1.5 MB of disk storage. However, it has grown larger than its initial size and contains many logically deleted records. The CLI command FILESTATUS shows that PARTFILE now occupies about 2 MB of disk space. To monitor the growth and current file structure of PARTFILE, invoke ANALYZE and enter PARTFILE at the **FILENAME** prompt. ANALYZE provides the information shown in Figure 2-1.

```
ANALYSIS OF PARTFILE

CREATED BY ISAM REV 5.09



RECORD LENGTH IN BYTES                          100
NUMBER OF DATA RECORDS (INCLUDING DELETIONS)   9747
TOTAL NO. OF INDEX BLOCKS                      1892
% OF INDEX TO TOTAL FILE SIZE                    47
FREE-SLOTS                                        0
NO. OF DELETED RECORDS                         2319

   - Strike Any Key to Continue -
```

```
                        ANALYSIS OF PARTFILE

                            PRIMARY KEY
                            -----------

KEY LENGTH IN BYTES                              15
RELATIVE POSITION OF KEY WITHIN RECORD            1
NO. OF KEYS PER INDEX BLOCK                      25
INDEX DEPTH                                       4

LEVEL          BLOCKS         MIN     MAX     AVE     % FULL
-----          ------         ---     ---     ---     ------

1              749            13      24      13      52
2               57            13      21      13      53
3                4            13      18      14      57
4                1             4       4       4      16

NO. OF INDEX BLOCKS                                   811

                - Strike Any Key to Continue -
```

```
ANALYSIS OF PARTFILE

ALT KEY         1
-----------------

KEY LENGTH IN BYTES                          15
RELATIVE POSITION OF KEY WITHIN RECORD       16
NO. OF KEYS PER INDEX BLOCK                  21
INDEX DEPTH                                   4

LEVEL          BLOCKS         MIN     MAX     AVE     % FULL
-----          ------         ---     ---     ---     ------

1              974            10      18      10      48
2               97            10      14      10      48
3                9            10      17      10      51
4                1             9       9       9      43

NO. OF INDEX BLOCKS                                  1081
NO. OF DELETED KEYS                                     0

   Print - (Y) or (N)? _
```

Figure 2-1 *ANALYZE Display*

ANALYZE reveals that PARTFILE now requires four index levels for each key, that the file index structure requires 1892 blocks of disk, and that no logically deleted records have been physically removed from the file. Because the file has grown without apparent maintenance, assume the index packing density is about 50% for each level.

COLLAPSE may be used to remove logically deleted records. The size of the data portion of the file will not decrease; however, the space freed by deleted records may be reused to add new records to the file. The index portion of the file may be smaller because unused keys have been removed and because fewer index levels may be required.

Error Messages

Under exceptional conditions ANALYZE may display the following error messages:

```
ERROR DETECTED DURING ANALYSIS:
BOUNDARY LIMIT EXCEEDED (NX): filename
```

ANALYZE has determined that the file is possibly corrupt and has set the reliability flags. Run ISAMVERIFY, which will provide information for reconstructing the file with REBUILD or REORG, or will reset the reliability flags to enable you to access the file.

```
FILE NOT FOUND, ALREADY OPEN OR READ PROTECTED: filename
```

The specified file does not exist, is already opened, or your ACL for the file does not include permission to read the file. ANALYZE returns you to the `FILENAME` prompt, where you can enter another filename.

```
WARNING: LOGICAL FILE STRUCTURE MAY BE CORRUPT filename
RUN 'ISAMVERIFY' TO CHECK FILE INTEGRITY
```

ANALYZE has checked file integrity and has determined that one or both reliability flags are set. Run the utility ISAMVERIFY on the file.

The program CALC is a general purpose calculating aid. It provides ten variables that may be assigned values, both built-in and user-defined functions, and displays the five most recent results of calculations.

Procedure

To invoke CALC from the CLI, enter **CALC**. The following information appears:

```
E -- Electronic Desk Calculator....CALC
L -- Library Maintenance.....CALCLIB

Please enter your selection:
```

Enter **E** to invoke CALC. To invoke CALC from the Logon Menu, enter **CALC** in response to the **RUN PROGRAM** prompt. The CALC screen appears, which is shown in Figure 2-2.

```
Date:  4/11/84      CALC Revision 1.30              ^F8 to exit
Time:  12:34:56                                    Library:


                                          0.0000 = A (       )
                                          0.0000 = B (       )
   Results:                               0.0000 = C (       )
                                          0.0000 = D (       )
              0.0000 = K                  0.0000 = E (       )
              0.0000 = L                  0.0000 = F (       )
              0.0000 = M                  0.0000 = G (       )
              0.0000 = N                  0.0000 = H (       )
              0.0000 = O                  0.0000 = I (       )
                                          0.0000 = J (       )

Enter Expression:



ORDER OF    (1) !         (4) * / \ |     SAVE:  f6   LABEL:  f8
OPERATIONS: (2) +Q -Q FUN (5) + - % <>    SEE:   F6   LIB:    f7
            (3) ^         (6) =           KILL: ^F6
```

***Figure 2-2** CALC Screen*

Function keys f6, f7, and f8 alone or combined with the SHIFT and CTRL keys may be used with CALC. In Figure 2-2 F6 is SHIFT-f6, ^F6 is CTRL-SHIFT-f6, and ^F8 is CTRL-SHIFT-f8. The definitions of the function keys appear in Table 2-1.

Key	Function
f6	SAVE key. Define or modify a function.
F6	SEE key (SHIFT-f6). Examine the definition of a function in the scratch or personal libraries.
^F6	KILL key (CTRL-SHIFT-f6). Remove a function from the scratch library.
f7	LIB key. Enter or change your personal library or enter the master library.
f8	LABEL key. Assign a variable name.
^F8	CTRL-SHIFT-f8. Exit from CALC.

Table 2-1 CALC Function Keys

CALC provides ten variable names, A through J. The value of each variable is displayed on the screen and updated whenever it is changed. You may assign names to these variables by using the LABEL key (f8). Press LABEL before entering an expression, or end an expression by pressing LABEL. In either case, CALC prompts for a new name. The name you have assigned to the variable appears in the parentheses to the right of the variable name. However, you must use the one-letter name in the actual expressions.

To facilitate entering large numbers, commas and leading dollar signs are accepted and ignored as part of a numeric string.

The equal sign operator assigns a value to a variable. For example,

F = (B-1)/25

subtracts 1 from B, divides the result by 25, and stores that answer in F.

The five most recent results are automatically displayed on the screen and are named K through O, with K being the most recent result. These values may be used in expressions, but may not be the object of an assignment (i.e., K=3 is illegal).

Arithmetic Operators

CALC recognizes the arithmetic operators listed in Table 2-2.

Operator	Definition
=	Value assignment
+	Addition
−	Subtraction
*	Multiplication
/	Division
%	Percent difference. A%B gives the percent change from A to B.
\	Quotient. A\B gives the whole number part of A/B.
|	Remainder. A|B gives the remainder of A/B.
<	Minimum. A<B gives the lesser of A and B.
>	Maximum. A>B gives the greater of A and B.
^	Exponentiation. A^B is A raised to the power B.
!	Factorial. A! is A * (A-1) * (A-2) * ... * 2 * 1.

Table 2-2 Arithmetic Operators in CALC

If an expression includes several operations, CALC executes them in their order of precedence. Operations at the same level of precedence are executed from left to right, except that multiple exponentiation operators are executed from right to left. Parentheses may alter the normal order of execution; elements in parentheses are executed first. Table 2-3 shows the levels of precedence from the highest level (1) to the lowest level (6).

Precedence	Operator
Level 1	!
Level 2	Functions, leading + and −
Level 3	^
Level 4	* / \ |
Level 5	+ − % < >
Level 6	=

Table 2-3 Levels of Precedence of Operators

CALC displays results truncated to four decimal places. Table 2-4 lists examples of expressions and their results.

Expression	Result	Remarks
1* 2/3	0.6666	/ and * are of equal priority; they are executed from left to right.
5* 2^3	40.0000	^ is executed first.
150%100	−33.3333	150 to 100 is a 33.3% decrease.
SQT 2* 2	2.8284	SQT is resolved before multiplication.
SQT (2* 2)	2.0000	Parentheses force * to be executed first.
2^2^3	256.0000	The rightmost ^ is resolved first.
A=1+B=1	2.0000	B=1 is executed first, then A=1+B.

Table 2-4 Examples of Calculations

The order of operations may differ in expressions containing more than one equal sign. In general, CALC resolves expressions containing more than one equal sign according to the following procedure:

1. Evaluate the expression following the rightmost equal sign and assign the value to the letter preceding that equal sign.
2. Substitute that variable's value in the expression.
3. Repeat step 1 if there are other variable assignments. Otherwise, evaluate the rest of the expression as usual.

The following example shows the right-to-left-processing order:

$$15 + A = 4 * 3 - 2 * B = 22 / C = 4 + \underbrace{D = 1 + 2 * 3}_{D = 7}$$

$$\underbrace{C = 4 + 7}_{C = 11}$$

$$\underbrace{B = 22 / 11}_{B = 2}$$

$$\underbrace{A = 4 * 3 - 2 * 2}_{A = 8}$$

$$\underbrace{15 + 8}_{23}$$

The value of the expression is 23, which is not assigned to a variable.

Functions

CALC function names consist of three letters. Invoke a function by entering its name and a single trailing argument, e.g., ZAH 3. CALC recognizes two kinds of functions: built-in and user-defined. The *built-in functions* are:

- SQT: square root
- LOG: common logarithm (log to base 10)
- EXP: *e* to the given power (*e* is approximately 2.718282)

Names of *user-defined functions* are three letters long and are stored in scratch or personal libraries. A *scratch library* is created each time you enter CALC and is destroyed each time you leave CALC. A new function is stored in the scratch library and in the personal library if specified. Whenever you reference a user-defined function, CALC gets a copy of it from the scratch library.

A *personal library* is saved from session to session. To open a personal library after entering CALC, press the LIB key (f7). CALC prompts you for the name of the library. If the library does not exist, CALC asks if it should be created. To clear the connection without opening or creating a library, enter a blank name. To destroy a personal library, issue the following commands from the CLI:

```
DELETE[/V] CALC$library-name.(NX,XD)
```

The *master library* may not be modified. This library contains the following functions, which are supplied automatically with CALC:

- ABS: absolute value of the argument
- COS: cosine of the argument, which is in radians
- DEG: converts the argument from degrees to radians
- LGE: natural logarithm (base *e*) of the argument
- SIN: sine of the argument, which is in radians

CALC searches for functions in the following order: the built-in functions, the scratch library, all personal libraries that have been opened during that session, and the master library. Thus if you give a personal library function the same name as a built-in function, the built-in function is used. When CALC finds a function, it copies it into the scratch library. CALC displays a message if it fails to find a function. Although you cannot alter master library functions, you can override them by giving a personal or scratch library function the same name as a master library function.

To define or change a function in the scratch or personal library, type an expression that includes the simple variable X. Instead of pressing CR to enter the expression, press the SAVE key (f6), which displays a prompt asking for the name of the function. For example, to define a function TAX, which computes the 5% sales tax on an item, type .05 * X at the entry line and press SAVE (f6). When CALC asks for the function name, enter TAX. Assuming no function with that name exists in the scratch library, CALC stores it and returns the normal calculator line. To test the new function, type TAX $100. CALC shows a result of 5.0000.

CALC next attempts to place the function definition in the scratch library. If the name is in use, CALC asks if the old definition should be deleted in favor of the new one. If you respond N, the definition process is cancelled. If you respond Y, the definition is put into the scratch library. If a personal library is open, CALC then asks if it should enter the function into the personal library.

To examine the definition of a function in the scratch or personal library, press SEE (SHIFT-f6). CALC prompts for the name of the function and searches for it, first in the scratch library, and then, if necessary, in the currently open personal library. With a successful search, CALC displays the definition.

Expressions can be used with functions. For example, entering TAX (17+3) gives a result of 1.0000, which is 5% of 20. Note that the parentheses are important; without them, CALC evaluates the expression as (TAX 17) + 3 and returns the result 3.8500.

To remove a function from the scratch library, press KILL (CTRL-SHIFT-f6). CALC asks for the function name to be deleted.

CALCLIB

CALCLIB is a program for maintaining the function libraries. CALCLIB has four options:

- Display the definition of a function.
- Display a list of all the functions in the library.
- Remove the definition of a function.
- Copy the contents of the library to the printer.

To run CALCLIB, do one of the following:

- Enter `ICX CALCLIB` at the CLI
- Enter `CALCLIB` in response to the `RUN PROGRAM` prompt on the Logon Menu.
- Enter `CALC` at the CLI, then select L (Library Maintenance).

CALCLIB asks for a CALC library name. Enter the name or blanks for the master library. The screen that CALCLIB displays is shown in Figure 2-3.

```
                    Library xxxxxx

What do you want to do?

     1  Inspect a function
     2  See list of functions
     3  Remove a function
     4  Copy library to printer

(ESC to exit)
```

***Figure 2-3** CALCIB Screen*

Inspect a function. CALC prompts for the function name. If the function exists in the specified library, CALCLIB displays its definition. If the function does not exist, CALCLIB displays a message to that effect.

See list of functions. CALCLIB displays an alphabetical list of all the functions in the library, 64 names to a page. If there are too many to fit on a page, CALCLIB pauses; press CR to continue.

Remove a function. CALCLIB prompts for the name of the function, deletes it, and displays a confirmation message. To delete another function, enter its name.

Copy library to printer. CALCLIB copies the contents of the library to a printfile.

If no functions are in the library, CALCLIB displays a message that the library is empty. Otherwise, the program confirms that it is copying the definitions to the printer or the print file. The name of the print file is CALC followed by the time, a period, and the three-digit line number of the user's terminal. Print the print file using the CLI command QPRINT.

Error Messages

Under exceptional conditions CALC or CALCLIB may display the following error messages:

EXPRESSION TOO COMPLICATED

While interpreting an expression, CALC ran out of temporary storage. This is typically caused by too many levels of parentheses or too many assignments in conjunction with other operations. A standard remedy is to break the expression into two or more expressions.

EXPRESSION TOO LONG

While evaluating an expression, CALC ran out of room to store intermediate values. Split the expression into smaller segments.

ERROR IN EXPRESSION SYNTAX

While scanning an expression, CALC identified an illegal sequence, such as A//B or 3(6/7).

FACTORIAL NEEDS A POSITIVE INTEGER

The factorial operator only works for numbers that are 0 or greater and do not have any fractional part.

FRACTIONAL EXPONENT NEEDS POSITIVE BASE

In A^B, if B has a fractional part, A must not be less than zero.

ILLEGAL VALUE SUPPLIED TO FUNCTION xxx

The value given to the named function was not in the acceptable range of values for that function. For SQT, the number must be no less than 0. For LOG, the number must be greater than 0.

INVALID CHARACTER IN EXPRESSION

A character was encountered that was not a capital letter, a numeric character (digit, period, comma, or dollar sign), or an operator. The illegal character is shown after the message.

INVALID NUMBER ENTERED

An invalid number may be a $ at the end of the line, a number with more than one decimal point, or a number with commas after the decimal point.

INVALID USE OF =

The assignment operator may only assign to variables A through J. Examples of invalid use are 3=1/5 and N=3.

INVALID VARIABLE IN EXPRESSION

An invalid variable name was found. Only A through J are valid. The invalid variable is shown after the message.

MASTER LIBRARY NOT FOUND

CALC$$LIB.NX or CALC$$LIB.XD are not in :UTIL:ICOBOL, your current directory, or your searchlist.

NAME NOT AVAILABLE

You tried to create a new library with an invalid name.

NO SUCH FUNCTION

You asked to see the definition of a function that does not exist in the current directory, is not in your searchlist, or is not linked to the current directory.

NO SUCH LIBRARY

You tried to access a library that does not exist, that was not in your searchlist, or that you were not linked to.

NUMBER HAS TOO MANY DIGITS

The number entered is too large for CALC to handle.

RESULT TOO BIG TO HANDLE

While performing an operation, CALC encountered a number too big to process without generating an incorrect result.

THIS LIBRARY IS EMPTY; NO PRINT WILL BE DONE
THIS LIBRARY IS EMPTY, SO THERE IS NOTHING TO LIST

You tried to print or list a library that contains no functions.

UNKNOWN FUNCTION

Two or more letters in a row were found that do not make up a valid function name. This illegal name is displayed after the message.

UNRECOVERABLE DISK ERROR IN PERSONAL LIBRARY
UNRECOVERABLE DISK ERROR IN SCRATCH LIBRARY

An error occurred while altering the scratch or personal library. Run ISAMVERIFY to check the integrity of the library file, following the procedures under ISAMVERIFY. If ISAMVERIFY does not detect an error, you may have a disk or system problem.

YOU MAY NOT ALTER THE MASTER LIBRARY

You cannot alter the functions in the master library. However, you may overlay a master library function by giving a personal or scratch library function the same name as a master library function.

COLLAPSE tailors the structure of indexed files to optimize storage and access time. The utility physically removes logically deleted records from the data portion of an indexed file and repacks the index portion of the file from 50% to 99% full, or repacks the index only.

COLLAPSE requires no temporary work files to streamline indexed files. All disk space freed by streamlining is returned to the collapsed file or to the operating system. Streamlining statistics on collapsed files are reported in a log file.

To restore file space to a disk that is nearly full, use the utility's record deletion facility. This physically deletes records that are logically deleted.

To improve access time for frequently read files, repack the index at a high density. High-density packing results in a smaller index file, which uses less disk space.

If the main operations on indexed files are writing and rewriting, it may be desirable to maintain a low packing density because high-density index packing increases write time. However, a low packing density also results in a larger index file, which uses more disk space.

If indexed files are to be archived on tape or disk, COLLAPSE may be run on these files to physically delete logically deleted records and to pack indexes up to 99% full. When the files are retrieved for rewriting, COLLAPSE may be rerun on them to repack indexes less tightly for maximum writing facility.

COLLAPSE does not work on relative files. Use REORG to remove logically deleted records from a relative file (see "Relative to Relative" under the REORG section). To repack the index portion of a relative file, use the REBUILD utility.

COLLAPSE does not operate on a file if the reliability flags are set. COLLAPSE sets the reliability flags while it is working on a file and clears them when it has finished. Therefore, if COLLAPSE is interrupted in any way, the reliability flags remain set. Further, because COLLAPSE changes the file while operating on it, the file should be considered corrupt.

Removing Logically Deleted Records

A record deleted by ISAM is marked as logically deleted, but it is not physically removed from the file. COLLAPSE can physically remove these records, thus freeing disk space, which may be reused by the file. COLLAPSE does not make the data portion of a file smaller. Rather, it increases the data portion's capacity within the current file size by collecting the "free slots" at the end of the data portion, where new records can be written.

The index portion of the file, however, probably will be smaller. For each deleted record that is removed, the corresponding key entry in the index portion is also removed. This space is returned to the operating system.

Use ANALYZE to determine the number of new records that can be added to a collapsed file without increasing overall file size. ANALYZE reports the number of records removed by the last COLLAPSE as FREE-SLOTS. Run ANALYZE on a file before and after a COLLAPSE session to confirm the results of removing logically deleted records.

Packing the Index Structure

COLLAPSE packs index structures from 50% to 99% full by deleting the old index file and building a new packed index file. To build a packed index file, COLLAPSE reads the data file in keyed sequential order, starting with the primary key and rereading for each alternate key in the file.

The utility FILESTATS helps in determining optimum packing density for an index structure before running COLLAPSE on the file. FILESTATS accepts index packing density as a parameter to its predictions of file storage requirements.

The Command Line

The command line tells COLLAPSE (1) whether to physically delete logically deleted records from a file and (2) the extent to which each level of the index structure is to be packed. The command line has the following format:

```
COLLAPSE[/D] filename[xx xx xx xx xx]
```

/D Physically delete records that are logically deleted. If the /D switch is omitted, logically deleted records are not physically deleted.

xx Pack the index to *xx*% full. The default value is 99; each entry may range from 50 to 99. A value for each index key may be specified, starting with the primary key and ending with the last alternate key.

CLI Macro Command Files

You can submit commands to COLLAPSE via a CLI macro command file, which can contain any number of COLLAPSE command lines. Any number of files on the disk can be collapsed with a single macro command file.

CLI macro command files can be prepared with a system editor or by the CLI command CREATE. For example, the contents of a macro command file STREAMLINE.CLI may be:

```
COLLAPSE/D ACME 80 50
COLLAPSE ABC 75 50 50
COLLAPSE/D REGIONAL 99 75
```

To execute the commands in the macro command file, enter `STREAMLINE` at the CLI.

Log File

After a session, COLLAPSE appends file streamlining statistics to a file called COLLAPSE.LG. This file is created if it does not exist. It may be displayed using a type or print command from the CLI. Any errors encountered during the pass are entered into this log file.

Example

In the following example, COLLAPSE is run on the file PARTFILE, using information from the ANALYZE and FILESTATS utilities. The example shows the interrelationships of these utilities in data file maintenance and planning.

PARTFILE has outgrown its planned size. FILESTATS reveals that the file could be returned to its proper size by packing the file's primary key index 95% full, packing the alternate key indexes 70% full, and physically removing logically deleted records. Refer to the PARTFILE statistics reported by ANALYZE and FILESTATS to compare PARTFILE storage requirements and index structure before and after the COLLAPSE session. The COLLAPSE.LG statistics for PARTFILE appear in Figure 2-4.

```
COLLAPSE ISAM FILE: PARTFILE
----------------

COMMAND:       PARTFILE/D/95/70          4/11/83 12:30:3
--------

DATA FILE: PARTFILE.XD
---------

       STARTING NUMBER OF DATA RECORDS:        9751
       NUMBER OF DATA RECORDS FREED:           2319
       CURRENT NUMBER OF DATA RECORDS:         7432
       DATA FILE SIZE:                      1072682


INDEX FILE: PARTFILE.NX
----------

       STARTING NUMBER OF INDEX BLOCKS:        1893
       NUMBER OF INDEX BLOCKS FREED:            980
       CURRENT NUMBER OF INDEX BLOCKS:          913
       INDEX LEVELS

              PRIMARY KEY
                     BEFORE:        4
                     AFTER:         3

              ALTERNATE KEY 1
                     BEFORE:        4
                     AFTER:         4
```

Figure 2-4 *COLLAPSE Log File Statistics*

The statistics in Figure 2-4 show that COLLAPSE physically removed 2319 logically deleted data records from the data file and freed 980 index blocks from the index file. If ANALYZE were run, it would show 2319 free slots.

Error Messages

COLLAPSE error messages indicate either a problem in the command line or a problem in the file being collapsed. Error messages that are caused by problems in the command line have explanations listed below. Error messages that indicate a problem in the file itself are simply listed. These errors are fatal. If you get a fatal error, run ISAMVERIFY on the backup file — not the collapsed one. Then run REBUILD or REORG if necessary.

CAN NOT GET PATH

CAN NOT RELINK RECORD

CAN NOT REMOVE RECORD

ILLEGAL .NX REVISION

Fatal error. The ISAM revision number is less than 5.00. The current COLLAPSE cannot process the file.

INDEX DEPTH EXCEEDED

LOGICAL FILE STRUCTURE MAY BE CORRUPT

The ISAM reliability flags are set. Run ISAMVERIFY, which will determine whether your file is corrupt. If it is not corrupt, ISAMVERIFY will clear the flags. If the file is corrupt, follow ISAMVERIFY with REBUILD or REORG or use a backup file.

NO FILE NAME

You have entered the utility name with no arguments. You must supply the name of a file to be collapsed.

.NX FILE ERROR

.NX FILE INACCESSABLE

The .NX portion may not be in your directory or searchlist; it may not exist; or someone else has opened the file.

UNABLE TO DELETE RECORD

.XD FILE ERROR

.XD FILE INACCESSABLE

The .XD portion may not be in your directory or searchlist; it may not exist; or someone else has opened the file.

CREV checks the compatibility of a program's object code with the runtime system. CREV displays the name of the operating system under which the Interactive COBOL source program was compiled, the revision levels of the compiler, and the code revision; it also indicates whether extended features were used.

Revision levels are displayed as a major level number, followed by a period and a minor level number. The major and minor levels can range from 0 to 99. The compiler produces as output two object files: *source-filename*.DD and *source-filename*.PD. The revision level of the compiler is written in the .PD file.

For example, if PAYROLL is the source-filename, CREV might supply the information displayed in Figure 2-5.

```
PAYROLL
        Compiled Under AOS/VS    Rev.  1.30
                       Code Rev.  6     Using Extended Features
```

Figure 2-5 *Sample CREV Screen*

Figure 2-5 indicates that the program was compiled under AOS/VS. The compiler that produced the object code for the source file PAYROLL had a major revision level of 1 and a minor revision level of 3. The code revision was 6.00, and extended features were used.

The code revision is keyed to the runtime system. The runtime system runs any program with a code revision of 4 or greater, regardless of the major or minor revision.

Extended features include use of a second system printer, alternate keyed search, and CALL PROGRAM USING.

Procedure

Execute the utility from the CLI with the following command:

```
CREV [source-filename]
```

Source-filename cannot include underscores. You can enter or omit the .PD extension; that is, CREV treats *source-filename*.PD and *source-filename* the same.

If you enter the source-filename at the command line, CREV displays the information, then returns you to the CLI.

If you do not enter the source-filename, you are prompted for it. CREV displays the information and prompts you for another program name. Enter it, or press ESC to terminate CREV.

CREV can also be run under the runtime system through Logon or with a CALL PROGRAM statement. CREV prompts you for a COBOL program name. It displays the program's revision level information and prompts you for another program name. To continue, enter another program name; to terminate CREV, press ESC.

CSSORT is a general sort and merge package. The sort operation accepts as input any Interactive COBOL file type and sorts on keys containing any data type. It produces a sequential file as output. The merge operation accepts as input up to six sorted sequential files of the same type and merges them into a single sequential file.

CSSORT can produce a variety of different output files from a given input file. By selecting only certain portions of the input file, CSSORT can reformat the records for the output file. This provides a tool for developing tailored reports from a master file. Records can be sorted on any data type in ascending or descending order or according to a user-defined collating sequence.

You can define a collating sequence by creating a file that contains an alternate sequence and specifying this sequence's filename in the command line.

CSSORT sorts an input file as specified by the user and outputs a new sorted file. Files that have been sorted on the same key can be merged in a separate operation into one file. However, CSSORT does not run on a file if a file's reliability flags are set.

A statistical report file can be produced for either the sort or merge options by specifying an audit filename in the command line. Since this file is a line sequential file, it can be printed.

CSSORT uses program-generated work files for the sort operation. Normally, these files are named by the CSSORT program. The command line accepts as optional arguments user-specified work filenames.

Sort Procedure

The minimal command line for the sort operations includes an input file (with the input record size if the file is fixed sequential), an output file, and one key specifier. The basic command line is:

```
CSSORT[/N] { in-file[/S] in-rec-size/N } { out-file/O/t                          }
           { in-file/t [in-rec-size/N] } { out-file/O[/S] [sequence-file/C] }

{key-specifier}... [field-specifier]... [workfile/W]... [auditfile/A]
```

Global Switch

/N Suppresses terminal display of audit information. This switch must be used with batch jobs or the system returns the error `LINE TOO LONG`.

Local Switches

/t Defines the file type for *in-file* and *out-file*. The default is a fixed sequential file. *t* may have the following values:

Input-file Type		Output-file Type	
/S	Fixed sequential	/S	Fixed sequential
/L	Line sequential	/L	Line sequential
/V	Variable sequential	/V	Variable sequential
/V/A	AOS-format variable sequential	/V/A	AOS-format variable sequential
/I	Indexed		
/R	Relative		

Variable sequential files store the size of the record in the record. With a variable sequential file created under AOS or AOS/VS COBOL, use the /V/A switches. With a variable sequential file created under Interactive COBOL on AOS or AOS/VS, use the /V switch. This difference arises because AOS and AOS/VS COBOL use a 4-byte header to store record length, while Interactive COBOL uses a 2-byte header. Use CSSORT to convert variable sequential files created under AOS or AOS/VS COBOL to a form compatible with Interactive COBOL.

Required Arguments

in-file/t

The name of the input file. For an indexed or relative file, the filename should not include the extensions .NX and .XD. The maximum filename length is 31 characters. If a value for the /t switch is omitted, the input file is assumed to be fixed sequential, and the input record size is required.

in-rec-size/N

The number of bytes in the input record. The input record size is required only if the input file is fixed sequential. The record length in a line sequential file is limited to 132 bytes plus the terminator. Specify the maximum size for variable sequential input whenever it is known. Otherwise the default, 4096 bytes, may diminish CSSORT's efficiency.

out-file/O/t

The name of the output file. The name cannot currently exist. If a value for the /t switch is omitted, fixed sequential is assumed.

key-specifier

The position and length of a key in bytes followed by /K. From one to eight key specifiers can be given; their order is important. A key specifier does not have to correspond to a primary or alternate record key in an indexed file; rather, it is the portion of a record to be used for sorting. The first key specifier is the major key used for sorting. The other keys are the minor keys, weighted with decreasing significance according to their order in the command line.

Key specifiers have the following form:

key-pos { . / : } key-length/K[/D][/d]

Key-pos is the position of the first byte of the key in the input record. (The position of the first byte of the record is byte number 1.) *Key-length* is the byte length of the key in the input record. The byte length cannot exceed the record size nor run past the end of the record from the key position specified.

/K indicates a key specifier. Keys can be specified for overlapping areas of the record.

/D indicates descending order. If omitted, the sort is ascending. Ascending

and descending keys can be intermixed among the specified keys.

/d indicates data type. ASCII characters are the default. The data type is one of the following:

Switch	Data Type
/A	ASCII characters
/N	Numeric display, unsigned
/N/L/S	Numeric display, leading separate sign
/N/T/S	Numeric display, trailing separate sign
/N/L	Numeric display, leading sign
/N/T	Numeric display, trailing sign
/C	Computational, unsigned
/C/S	Computational, signed

See the "USAGE clause" and "SIGN clause" sections in the *Interactive COBOL Programmer's Reference* for further information on data types.

Optional Arguments

field-specifier

The position and length of an input field to be moved to the output record. Field specifiers tailor the output file to include only selected portions of the input file. There is a limit of eight field specifiers, each with the following form:

field-pos  field-length/F

Field-pos is the position of the first byte of a field in the input record to be moved to the output record. (The position of the first byte of the record is byte 1.) When field specifiers are present, the fields appear in the output record in the order specified in the command line. Fields may be overlapping or duplicated. *Field-length* is the byte length of a field.

work-filename/W

The name of a temporary sorting file. CSSORT uses up to six work files. If you do not explicitly name them, the program creates work files named SORTW1.TP through SORTW6.TP. If these files exist when the SORT is executed, they are deleted and re-created. The most active work files are SORTW1.TP and SORTW4.TP. If there is insufficient disk space or a large number of records to sort, you can specify work files located on different physical devices to increase the efficiency of the program. For example, :DPZ1:DIR1:WORK1.TP/W indicates that WORK1.TP in directory DIR1 on DPZ1 is a work file. The work files' names are assigned to the user-specified names in the order that they appear in the command line. Thus, to specify work file 4 in the command line, the first three must be included.

auditfile/A

The name of the file to which processing information is written. The auditfile contains information such as the operation, input and output filenames, file types, record size, key specifiers, output field specifiers, listing filename, sequence filename, work filenames, time elapsed, and number of records input and output. Do not specify @LPT as both the audit and the output files.

sequence-file/C
: The name of the file containing a user-specified collating sequence. The alternate collating sequence applies only to keys having the ASCII data type. The alternate collating sequence has no effect on keys having numeric or computational data types.

The position of the arguments in the command line is not fixed. However, the order of the key specifiers, field specifiers, and work files is important. For example, the input, output, and audit filenames can appear anywhere in the command line, but *key-specifier-1* must precede *key-specifier-2, field-specifier-1* must precede *field-specifier-2,* and *work-filename-1* must precede *work-filename-2.*

Alternate Collating Sequences

The order in a sort is usually determined by the straight ASCII sequence. The ASCII characters are represented by internal codes consisting of decimal integers from 0 through 127 in the following sequence: digits (0-9), uppercase letters (A-Z), and lowercase letters (a-z). This integer representation permits the characters to be compared for precedence. By temporarily reassigning the integer corresponding to any character, the precedence of the character with respect to the rest of the characters can be altered. This is done using an alternate collating sequence file. The characters in the file are assigned the first (lowest) precedence. This means that new integer codes are assigned to them, starting with 0. The remaining ASCII characters, which are not in the file, are assigned the remaining codes in their usual order with respect to each other.

Be careful when creating alternate collating sequences, or the results may be unexpected. For example, one might want simply to switch uppercase A and B in the alphabetic sequence by creating the collating sequence file SWITCHAB:

```
CREATE/I SWITCHAB<NL>
BA <NL>
)<NL>
```

However, the resulting alternate collating sequence would be incorrect: B, A, digits, remaining uppercase letters, and lowercase letters. Instead of merely switching positions, B and A now precede the digits. The correct sequence is created by specifying the collating sequence as 0123456789BA. The resulting sequence is: digits, B, A, remaining uppercase letters, and lowercase letters in their normal order.

Observe these rules when creating alternate collating sequences:

1. The file can contain only ASCII characters.
2. No character can be repeated.
3. The file cannot be longer than 128 characters.
4. 8-bit characters cannot be used to define an alternate collating sequence.

Using an Alternate Collating Sequence File

In this example, the user wants to generate a report that indicates by salesperson the products being sold. The following information is needed to develop the command line:

Input Filename: SALESREC
File Type: Indexed

Record Position	Contents	Byte Pos : Byte Length
1-6	Item stock number	1 : 6
7-10	Salesperson code	7 : 4
11-16	Transaction date	11 : 6
17-22	Customer number	17 : 6
23-36	Item description	23 : 14
37-40	Item class	37 : 4
41-43	Units sold	41 : 3
44-45	Discount code	44 : 2

Output Filename: SALESITEM
File Type: Fixed sequential

The output file is to be formatted to include the salesperson code, the item sold, number of units sold, and the transaction date:

Record Position	Contents	Byte Pos : Byte Length
1-4	Salesperson code	1 : 4
5-10	Item stock number	5 : 6
11-13	Units sold	11 : 3
14-19	Transaction date	14 : 6

The salesperson codes are numeric. Several years ago the stock number formats were changed. The old stock item numbers are six characters long. The first two characters are upper-case alphabetic and the last four are numeric. New stock item numbers are six digits. When sorting the file, the user wants the old stock items first, then the new ones.

This sort requires an alternate collating sequence in which uppercase must precede digits. The alternate collating sequence would be created as follows:

```
CREATE/I UPPERCASE <NL>
ABCDEFGHIJKLMNOPQRSTUVWXYZ <NL>
)<NL>
```

Since the upper-case letters are in the alternate collating sequence, they are assigned first precedence. The remaining characters follow in their usual order.

The command line to generate the sales report is:

```
CSSORT SALESREC/I SALESITEM/O 7:4/K/N 1:6/K 11:6/K/N 7:4/F 1:6/F 41:3/F 11:6/F @LPT/A UPPERCASE/C
```

The following is an analysis of the command line:

`CSSORT`	With /N omitted, audit information is displayed at the console.
`SALESREC/I`	Input filename. The /I switch indicates an indexed file.
`SALESITEM/O`	Output filename. The output file type switch is omitted; therefore, the file is to be fixed sequential.
`7:4/K/N`	Major key specifier: salesperson. Data type is unsigned numeric. This key is not affected by the alternate collating file.

1:6/K	Minor key specifier: item stock number. With the data type switch omitted, the default is ASCII. This key is sorted by an alternate collating sequence.
11:6/K/N	Minor key specifier: transaction date. Data type is unsigned numeric. This key is not affected by an alternate collating file.
7:4/F	Field specifier: salesperson code
1:6/F	Field specifier: item stock number
41:3/F	Field specifier: units sold
11:6/F	Field specifier: transaction date
@LPT/A	The audit information is to be printed at the system printer.
UPPERCASE/C	The alternate collating sequence filename

The following table illustrates the reformatting done by the field specifiers. The order of the field specifiers in the command line determines the order of fields in the output file. The input file format is listed in the first column, the output file format in the last.

Byte Pos : Byte Length	Contents	Byte Pos : Byte Length
7 : 4	Salesperson code	1 : 4
1 : 6	Item stock number	5 : 6
41 : 3	Units sold	11 : 3
11 : 6	Transaction date	14 : 6

The following audit information is displayed on the screen and printed at the system printer:

```
CSSORT/MERGE PROGRAM REV 1.10      4/11/83    ***SORT OPERATION***

            FILENAME              FILE TYPE      MAXIMUM RECORD SIZE

INPUT:    SALESREC              INDEXED                          45
OUTPUT:   SALESITEM           FIXED-RECORD SEQUENTIAL            19
KEYS:  START BYTE * LENGTH * ASC-DEC * DATA TYPE
           7         4        A      NUMERIC, UNSIGNED
           1         6        A      ASCII CHARACTERS
          11         6        A      NUMERIC, UNSIGNED

OUTPUT FIELD SPECIFIERS :  (START BYTE , LENGTH)
        7,  4          1,  6        41,  3          11,   6
LISTING FILENAME :     @LPT     SEQUENCE FILENAME:   UPPERCASE
WORK FILENAMES   :
      NONE SPECIFIED

PRESORT      11:49:28              11:50:09   NO. RECORDS IN : 673
LAST PASS    11:50:09     DONE     11:50:58   NO. RECORDS OUT: 673
FIXED SEQUENTIAL OUTPUT RECORD SIZE IS 19 BYTES.
```

Using More Than One Key-Specifier

In this example, a user wants to create a mailing list from the employee record file EMPREC. Because this list is to be used for bulk mailing, the sort is by zip code. For internal use, the user wants the sort refined by street, then by employee name.

The following information is needed to develop the command line:

Input Filename: EMPREC
File Type: Fixed sequential

Record Position	Contents	Byte Pos : Byte Length
1-5	Employee number	1 : 5
6-35	Employee name	6 : 30
36-55	Street	36 : 20
56-75	City	56 : 20
76-77	State	76 : 2
78-82	Zip	78 : 5
86-89	Extension	86 : 4
90-95	Date hired	90 : 6

Output Filename: MAILIST
File Type: Fixed sequential

The format of the output file is determined by the field specifiers in the command line. For this example, the employee name, street, city, state, and zip from the input file are to be included in the output file. These items are included in record positions 6-82 (i.e., byte position 6, byte length 77). The output record format is:

Record Position	Contents	Byte Pos : Byte Length
1-30	Employee name	1 : 30
31-50	Street	31 : 20
51-70	City	51 : 20
71-72	State	71 : 2
73-77	Zip	73 : 5

The command line to create the mailing list is:

```
CSSORT EMPREC/S 95/N MAILIST/O 78:5/K/N 36:20/K 6:30/K 6:77/F @LPT/A
```

The following is an analysis of the command line:

`CSSORT`	With /N omitted, audit information is displayed at the console.
`EMPREC/S`	Input filename. The /S indicates that file is fixed sequential.
`95/N`	The input record size. This is required because the file is fixed sequential.
`MAILIST/O`	Output filename. The output file type switch is omitted; therefore, the file is to be fixed sequential.
`78:5/K/N`	Major key specifier. Data type is unsigned numeric. An ascending sort on zip code is specified.
`36:20/K`	Minor key specifier 1. Duplicate zip codes are to be sorted by street address.
`6:30/K`	Minor key specifier 2. Duplicate street addresses are to be sorted by employee name.
`6:77/F`	Minor key specifier 3. The output file is to contain employee name, street, city, state, and zip.
`@LPT/A`	The audit information is to be printed on the system printer.

The following audit information is displayed on the screen and printed at the system printer:

```
CSSORT/MERGE PROGRAM REV 1.10      4/11/83    ***SORT OPERATION***
          FILENAME              FILE TYPE     MAXIMUM RECORD SIZE

INPUT:     EMPREC               FIXED-RECORD SEQUENTIAL       95
OUTPUT:    MAILIST              FIXED-RECORD SEQUENTIAL       77
KEYS: START  BYTE * LENGTH *  ASC-DEC * DATA TYPE
         78          5         A      NUMERIC, UNSIGNED
         36          20        A      ASCII CHARACTERS
          6          30        A      ASCII CHARACTERS

OUTPUT FIELD  SPECIFIERS :   (START BYTE , LENGTH)
                                6,        77
LISTING FILENAME  :   @LPT     SEQUENCE FILENAME:   NONE SPECIFIED
WORK FILENAMES    :
      NONE SPECIFIED

PRESORT      08:09:45           08:10:05   NO. RECORDS IN :    345
LAST PASS    08:10:05    DONE   08:10:35   NO. RECORDS OUT:    345
FIXED SEQUENTIAL OUTPUT RECORD SIZE IS 77 BYTES.
```

Sorting Indexed Files with Alternate Keys

In this example, the user wants to prepare a file that lists account balance, customer name, address, and account number in descending order by balance. If there are duplicate balances, the sort will be on account number in ascending order. The input file is ACCT_REC. It is indexed by account number as the primary key and customer name as the alternate key. The following information is necessary to develop the command line:

Input Filename: ACCT_REC
File Type: Indexed, with alternate keys

Record Position	Contents	Byte Pos : Byte Length
1-10	Account number	1 : 10
11-35	Customer name	11 : 25
36-55	Street	36 : 20
56-75	City	56 : 20
76-77	State	76 : 2
78-82	Zip	78 : 5
83-91	Account balance	82 : 9

Output Filename: ACCT_BAL
File Type: Line sequential

The output file is to be formatted to include the account balance, customer name, address, and account number. The output record format is:

Record Position	Contents	Byte Pos : Byte Length
1-9	Account balance	1 : 9
10-34	Customer name	9 : 25
35-54	Street	35 : 20
55-74	City	55 : 20
75-76	State	75 : 2
77-81	Zip	77 : 5
82-91	Account number	82 : 10

The command line to create the file in account balance sequence is:

```
CSSORT ACCT_REC/I ACCT_BAL/O/L 82:9/K/D/N/L 1:10/K/N 82:9/F 11:72/F 1:10/F
```

The following is an analysis of the command line:

CSSORT	With /N omitted, audit information is displayed at the console.
ACCT_REC/I	Input filename. The /I indicates that the file is indexed. No special accommodations need to be made for indexed files with alternate keys.
ACCT_BAL/O/L	Output filename. The /O indicates that the file is an output file. The /L indicates that the file type is line sequential.
82:9/K/D/N/L	The field position and length of the account balance field, the major key specifier. The /K indicates that the argument is a key specifier. The /D indicates that the final order is descending. The /N/L indicate that the data type is numeric with a leading sign.
1:10/K/N	Minor key specifier. The /N indicates that the data is unsigned numeric. Duplicate balances are to be sorted on account number, in ascending order. No other minor key specifiers are included since the account number is unique.
82:9/F	Field specifier: account balance. This is to appear at the beginning of each record in the output file.
11:72/F	Field specifier. This is the order in which the fields are to appear on the output file. The fields are customer name, street, city, state, and zip. Since the order of these fields will stay the same, they can be listed under one field specifier.
1:10/F	Field specifier: account number. This is to appear at the end of each record in the output file.

Merge Procedure

The merge procedure accepts up to six sequential files and merges them into a single sequential file. The sort procedure produces sequential files, the only file type the merge operation accepts. However, if the files to be merged are sequential and sorted on the field you wish to merge on, the sort procedure can be omitted.

The command line to merge variable or line sequential files is:

CSSORT/M/i[/N] in-file-1 in-file-2 [...in-file-6] out-file/O/t key-specifier-1 [...key-specifier-8] [field-specifier-1...field-specifier-8] [auditfile/A]

The input record size must be included if the input file type is fixed sequential. The command line to merge fixed sequential files is:

CSSORT/M/[/S][/N] in-file-1 in-file-2 [...in-file-6] out-file/O/t key-specifier-1 [...key-specifier-8] in-rec-size-1/N in-rec-size-2/N [...in-rec-size-6/N] [field-specifier-1...field-specifier-8] [auditfile/A]

Global Switches

/M Indicates the merge operation

/i Indicates the file type for the input files. Since the file type is indicated by a global switch, the input files must be the same type. The switches permitted are /S for fixed sequential files, /V for variable sequential files, and /L for line sequential files. The default is /S.

/N Suppresses console display of the audit information

Local Switches

The local switches are the same as in the sort operation.

Command Line Arguments

in-file
The name of a sorted input file. The maximum filename length is 31 characters. At least two input files are required; a maximum of six can be specified. The files must be the same type and already sorted on the keys to be used by the merge.

in-rec-size/N
The input record size in decimal. If the input files are fixed sequential, the record size is required. If the files are variable sequential, indicate the maximum record size.

out-file/O/t
The name of the output file. The output file type (/t) switches are /S for fixed sequential, /V for variable sequential, /L for line sequential files, and /V/A for AOS-format variable sequential files. If no type is specified, the default is fixed sequential.

key-specifier
At least one key specifier must be given; up to eight can be specified. Order is important. When sorting and merging on more than one key, the keys must be specified in the same order for both operations. Otherwise, the merged file is unsorted.

Key specifiers have the same form as in the sort operation. The switches have the same functions as in the sort operation. The form is:

key-pos { . : } key-length/K[/D][/d]

Optional Arguments

Work files cannot be specified for the merge option. Therefore the /W switch is not needed. The other optional arguments are the same as those used in the sort operation and can be included as needed:

field-specifier
To reformat the output file, up to eight field specifiers can be used. Their order in the command line is important. The switches have the same functions and form as in the sort operation.

field-pos { . : } field-length/F

auditfile/A
The name of the file to which processing information is written. The auditfile

contains audit information, including the operation, input and output filenames, file types, record size, key specifiers, output field specifiers, audit filename, sequence filename, time elapsed, number of records input for each file, total number of records input, and total number of records output. Do not name @LPT as both the audit and output file.

sequence-file/C
The name of the file containing the alternate collating sequence used in the sort operation. If the input files used in the merge were sorted using an alternate collating sequence, this collating sequence filename must be used.

Using the Merge Operation

In the following example, a company does a heavy volume of business. Each quarter, the new customer list is combined with the existing customer list to produce the quarterly update list. The new customer list is an indexed file named NEWCUST. The existing customer list is also an indexed file named OLDCUST. The customer number is the index key. Both files have the following record format:

Record Position	Contents	Byte Pos : Byte Length
1-6	Customer number	1 : 6
7-36	Customer name	7 : 30
37-81	Address	37 : 45
82-91	Phone number	82 : 10
92-116	Buyer	92 : 25
117-120	Credit limit	117 : 4

The updated customer list is a fixed sequential file named UPDATE. It is sorted alphabetically by customer name and retains the same record format.

Since input files in the merge command line must be sequential and sorted on the key to be used by the merge operation, NEWCUST and OLDCUST must be sorted on key position 7:30. Duplicates will be sorted on customer number, key position 1:6. The command lines for the sort are:

```
CSSORT NEWCUST/I ALPHANEW/O 7:30/K 1:6/K/N @LPT/A
CSSORT OLDCUST/I ALPHAOLD/O 7:30/K 1:6/K/N @LPT/A
```

The record size of these two new files can be obtained from the auditfile. This number is needed in the command line for the merge operation.

The two fixed sequential files, ALPHANEW and ALPHAOLD, can be merged alphabetically into the new UPDATE file. The command line is:

```
CSSORT/M ALPHANEW ALPHAOLD 120/N UPDATE/O 7:30/K 82:10/K/N @LPT/A
```

The following is an analysis of the command line:

CSSORT/M	The /M indicates the merge operation. With /N omitted, audit information is displayed at the console. With the global file type switch omitted, the input files are fixed sequential.
ALPHANEW	Input filename. File type is fixed sequential.
ALPHAOLD	Input filename. File type is fixed sequential.
120/N	Input record size. Because the input files are fixed sequential, they must all have the same record length.
UPDATE/O	Output filename. The output file type switch is omitted; therefore, the file is fixed sequential by default.

7:30/K	Major key specifier: customer name. With the data type switch omitted, the default is ASCII.
1:6/K/N	Minor key specifier: phone number. /N indicates unsigned numeric data.
@LPT/A	Audit information is to be printed on the system printer.

Since no field specifiers are given in the command line, the output file has the same record format as the input files.

The following audit information is displayed on the screen and printed on the system printer:

```
CSSORT/MERGE PROGRAM REV 1.10     4/11/83     ***MERGE OPERATION***

INPUT FILES: TYPE--FIXED-RECORD SEQUENTIAL   MAX RECORD SIZE:  120
NAMES:     1 - ALPHANEW      2 - ALPHAOLD
OUTPUT FILE: TYPE--FIXED-RECORD SEQUENTIAL   MAX RECORD SIZE:  120
NAME:  UPDATE
KEYS:  START BYTE * LENGTH * ASC-DESC * DATA TYPE
           7         30        A       ASCII CHARACTERS
          82         10        A       NUMERIC, UNSIGNED

OUTPUT FIELD SPECIFIERS :  (START BYTE , LENGTH)
       NONE SPECIFIED
AUDIT FILENAME :  @LPT     SEQUENCE FILENAME: NONE SPECIFIED
MERGE  16:35:51            DONE  16:39:05
INPUT RECORD COUNTS :
   1 -         59            2 -         948
TOTAL RECORDS INPUT  :     1,007    TOTAL RECORDS OUTPUT :    1,007
FIXED SEQUENTIAL OUTPUT RECORD SIZE :     120 BYTES
```

Error Messages

The full text of each CSSORT error message indicates whether the error is a user error or a program error. User errors are errors in the command line. Program errors are deficiencies uncovered by the operating system. They include such errors as uninitialized directories, unlocated files, and exhausted file space.

If you encounter a user error, correct the command line. If you encounter a program error, respond as with a CLI error message.

ALL INPUT FILES ARE EMPTY

You cannot merge empty files.

COLLATING SEQUENCE FILE IS TOO LARGE

The collating sequence file cannot be longer than 128 characters.

DUPLICATE KEY FIELDS

Key fields cannot be repeated.

DUP CHARACTERS WERE FOUND IN SEQUENCE FILE

No character can be repeated in an alternate collating sequence file.

FALSE EOF

ILLEGAL COMMAND SWITCH COMBINATION

You may have specified, for example, an indexed output file in a sort.

ILLEGAL GLOBAL SWITCH COMBINATION

You have specified more than one file type for the merge operation, for example, /L/S.

ILLEGAL RECORD SIZE

The record size is limited to 133 bytes for line sequential files and 4096 bytes for fixed or variable sequential files.

IMPROPER DATA FOUND IN KEY

IMPROPER DATA WAS FOUND IN SEQUENCE FILE

Only ASCII characters are allowed in the sequence file.

INPUT RECORD SIZE TOO SMALL FOR KEYS OR FIELDS

The record size must be able to accommodate the position and length of the key and field specifiers.

INSUFFICIENT MEMORY FOR PRESORT

INVALID OUTPUT FIELD SPECIFIER

The field specifiers must be in the form *field-pos.field-length*/F or *field-pos:field-length*/F.

INVALID KEY SPECIFIER

Key specifiers must be in the form *key-pos.key-length*/K or *key-pos:key-length*/K.

KEY RANGE ERROR

KEY SIZE EXCEEDS ITS DATA TYPE LIMIT

MORE THAN ONE SEQUENCE FILE

Only one alternate collating sequence file is permitted in a sort. You most likely specified more than one filename with the /C switch.

NO KEY SPECIFIERS

You must include at least one key specifier; indicate the specifier with the /K switch.

NO INPUT FILE SPECIFIED

You must specify at least one input file when sorting, or two input files when merging.

NO KEY OR OUTPUT FILE FOR SORT

One output file and one key specifier is required for each sort and merge operation.

NO OUTPUT FILE SPECIFIED

An output file is required and must be specified with the /O switch.

NO RECORD SIZE SPECIFIER FOR FIXED SEQUENTIAL

The number of bytes in the input record is required when the input file is fixed sequential. It is specified with the /N switch.

OUTPUT SPECIFIER RANGE ERROR

OUTPUT FILE MUST BE LINE SEQUENTIAL

When sorting files, the output file must be line sequential. No switch indicating file type is allowed.

OUTPUT RECORD IS TOO LONG FOR LINE SEQUENTIAL

The maximum record size for line sequential files is 133 bytes, including the terminator.

PROGRAM ERROR IN GTYPE

PROGRAM ERROR # 1

RECORD TOO SMALL FOR KEY OR FIELD

The specified record size must be able to accommodate the position and length of the key and field specifiers.

REVISION ERROR

TOO FEW INPUT FILES

You must specify at least two input files with the merge operation.

TOO MANY AUDIT FILES SPECIFIED

Only one audit file can be specified.

TOO MANY INPUT FILES SPECIFIED

Only one input file is permitted when sorting. Up to six files may be specified in the merge procedure. You may have forgotten to specify the output file with the /O switch.

TOO MANY KEY SPECIFIERS

Up to eight key specifiers may be included in the command line.

TOO MANY OUTPUT FIELD SPECIFIERS

Up to eight field specifiers may be included in the command line.

TOO MANY OUTPUT FILES SPECIFIED

Only one output file can be specified. You may have specified the /O switch for more than one file.

TOO MANY RECORD SIZE SPECIFIERS

Each input record size must be preceded by the corresponding input filename.

TOO MANY WORK FILES SPECIFIED

Up to six work files can be specified. Specify each work file with /W.

UNIDENTIFIED ERROR CODE

UNIDENTIFIED LOCAL SWITCH

An illegal local switch was specified.

UNRECOGNIZABLE ITEM IN COMMAND LINE

FILESTATS is an Interactive COBOL program that calculates file storage requirements based on record and key specifications. It can provide a comparison of alternative record formats, showing maximum disk space requirements for any given file.

FILESTATS accepts a COBOL file type (indexed, relative, or sequential) and appropriate parameters: record size, number of records, and, for indexed files, index packing density, number of keys, and their lengths.

FILESTATS accepts index packing density as a parameter to its calculations of ISAM file storage requirements; therefore, it can be used to determine the desired packing density for an index structure. The density of an indexed file can be changed by running the COLLAPSE utility.

FILESTATS displays as output the size of the data file in bytes and blocks (sectors). For relative and indexed files, the size of the index file is given for random file allocation. Storage requirements for a contiguous index are also displayed, which is useful if the file is to be used under RDOS or DG/RDOS. For indexed files, FILESTATS also calculates the number of keys per block, the number of index levels, and the number of nodes (occupied blocks) per index level for each alternate key.

Procedure

To invoke FILESTATS at the command line, enter `FILESTATS`. To invoke FILESTATS at the Logon Menu, enter `FILESTATS` in response to the `RUN PROGRAM` prompt.

The program begins by displaying:

```
FILESTATS OUTPUT TO PRINTER (P) OR DISPLAY (D)?
FILE TYPE--SEQUENTIAL (S), RELATIVE (R), OR INDEXED (I)?
```

At the first prompt, enter `P` to output the results on the line printer, or `D` to display them on the console. At the second prompt, enter the file type to be examined. Pressing ESC in response to either of these questions stops the program.

FILESTATS asks for record specifications, and key specifications if required. If you do not know the precise packing density of an index file, type 50% packing for all index structures to get worst-case statistics. Input limits appear in the prompts.

The file data is requested with the limits stated in the prompts. Pressing ESC when entering the data restarts the program. When the appropriate data has been entered, the program outputs the results of its calculations and displays the message `STRIKE ANY KEY TO PROCEED.`

Pressing any key resumes the display of output or returns the initial display. The program can be repeated any number of times to test as many file specifications as desired.

While FILESTATS computes file sizes in bytes, AOS and AOS/VS round relative and indexed files up to the next 2-KB multiple. For example, in the relative file example, FILESTATS shows the data file size as 837,512 bytes and the index file size as 56,832 bytes—a total of 894,344 bytes. However, the FILESTATUS/LENGTH command will show a total length of 894,976 bytes for the file. This is because rounding up to a 2-KB multiple yields 837,632 KB for the .XD portion and 57,344 KB for the .NX portion.

Sequential Files

Enter S at the FILE TYPE prompt to calculate storage results for a sequential file. FILESTATS requests the following information (sample responses are in angle brackets):

```
RECORD SIZE IN BYTES (1 - 32768):          <100>
NUMBER OF RECORDS:                         <1500>
```

FILESTATS then displays the information in Figure 2-6.

```
FILESTATS FOR SEQUENTIAL FILES:
1500 RECORDS:     100 BYTES

DATA FILE SIZE:   150,000 BYTES          293 DISK SECTORS
```

***Figure 2-6** FILESTATS Statistics for a Sequential File*

Relative Files

Enter R at the FILE TYPE prompt to calculate storage requirements for a relative file. The following information is requested (sample responses are in angle brackets):

```
RECORD SIZE IN BYTES (1 - 4096):           <180>
NUMBER OF RECORDS:                         <4500>
```

FILESTATS then displays the information in Figure 2-7.

```
FILESTATS FOR RELATIVE FILES:
4500 RECORDS:     180 BYTES


DATA FILE SIZE:             837,512 BYTES      1,636 DISK SECTORS
RANDOM INDEX FILE SIZE:      56,832 BYTES        111 DISK SECTORS
CONTIGUOUS INDEX FILE SIZE:  59,904 BYTES        117 DISK SECTORS
```

***Figure 2-7** FILESTATS Statistics for a Relative File*

Indexed Files

In this example, FILESTATS helps determine the extent to which the file PARTFILE should be streamlined. (In the ANALYZE section, this file had outgrown its planned size of 1.5 MB.)

To calculate the storage requirements for PARTFILE, enter I in response to the FILE TYPE prompt. The following information is requested; sample responses are listed in angle brackets:

```
RECORD SIZE IN BYTES  (1 - 4096):                 < 100 >
NUMBER OF RECORDS:                                < 7500 >
NUMBER OF ALTERNATE KEYS (1 - 4):                 < 1 >
PRIMARY KEY LENGTH IN BYTES (1 - 100):            < 15 >
PACKING-DENSITY (50% - 99%):                      < 95 >
ALTERNATE KEY 1 LENGTH, IN BYTES (1 - 100):       < 15 >
PACKING-DENSITY (50% - 99%):                      < 70 >
```

After you enter the requested information, FILESTATS displays the calculations that indicate the requirements for both the data and the index portions of PARTFILE (see Figure 2-8).

```
FILESTATS FOR INDEXED FILES:
7500 RECORDS:  100 BYTES
      2 KEYS:       15 BYTES, ROUNDED TO: 16 BYTES
                    15 BYTES, ROUNDED TO: 16 BYTES

DATA FILE SIZE:             825,512 BYTES      1,613 DISK SECTORS
RANDOM INDEX FILE SIZE:     472,064 BYTES        922 DISK SECTORS
CONTIGUOUS INDEX FILE SIZE: 477,696 BYTES        933 DISK SECTORS

FOR PRIMARY KEY, 16 BYTES:
-------------------------
   25 KEYS PER SECTOR
   95% PACKING-DENSITY
   3 INDEX LEVELS

AT INDEX LEVEL:        3      2      1
NUMBER OF BLOCKS:      1     15    327


FOR ALTERNATE KEY 1,  16 BYTES:
------------------------------
   21 KEYS PER SECTOR
   70% PACKING-DENSITY
   4 INDEX LEVELS

AT INDEX LEVEL:        4      3      2      1
NUMBER OF BLOCKS:      1      3     38    536
```

Figure 2-8 *FILESTATS Statistics for an Indexed File*

Note: In Figure 2-8 ANALYZE reports that the data file size for PARTFILE is 975,212 bytes (record length of 100 bytes * 9747 data records + header block of 512 bytes). This discrepancy is due to the fact that the logically deleted space in PARTFILE is not returned to the operating system by COLLAPSE.

Using packing densities of 95% for the primary key and 70% for the alternate key, FILESTATS confirms that the index structure of PARTFILE can be streamlined to only three index levels for the primary key and that the number of disk sectors (blocks) required by the file's index structure can be significantly reduced at these packing densities. To reduce the size of PARTFILE below 1.5 MB, FILESTATS indicates that COLLAPSE should be run on the file with the packing densities indicated above. See the COLLAPSE description for the results of this procedure.

Warning Messages

The following warning messages appear when you define a file that is larger than the system can support.

```
WARNING: DATA FILE EXCEEDS MAXIMUM FILE SIZE OF 33,554,432

WARNING: INDEX FILE EXCEEDS MAXIMUM FILE SIZE OF 33,554,432

WARNING: INDEX DEPTH WILL EXCEED 6 LEVELS
```

ICINQUIRE enables you to review and update data files. It generates an Interactive COBOL inquiry program, eliminating the need to design different COBOL programs for each data file inquiry. Updated transactions can be saved in a disk file for printing and listing. File and record-locking options make this a safe multiuser data-access facility.

The utility generates an Interactive COBOL inquiry program by inserting a COBOL record description into a skeletal COBOL program (MODEL is the smaller version, XMODEL the extended version) and then compiling the COBOL inquiry program. The inquiry program is created via the CLI. However, it runs either from the CLI or from Logon, and can process or create any ISAM file with records and key structures matching its own.

Procedure

To create the inquiry program, type the following at the CLI:

```
ICINQUIRE[/X] [descriptor-filename/I] inquiry-program-name/O [listfile/L]
```

/X Calls the extended utility, which supports full ISAM capability, including alternate keyed search, CALL PROGRAM USING, and use of a second system printer.

descriptor-filename/I
The optional input filename. The descriptor file contains a user-created FD section (see Figure 2-9), which ICINQUIRE accepts as a COBOL file descriptor.

inquiry-program-name/O
The name of the COBOL Inquiry program that you want ICINQUIRE to create.

listfile/L
The optional name of a file to receive the compilation listing. If you do not specify a listing file, compilation statistics are displayed on the screen.

After you enter the command line, ICINQUIRE asks if the data file is indexed or relative and then prompts for the record or relative key. If you are running the extended version for an indexed file, you are asked to specify alternate keys. If no descriptor file is specified, ICINQUIRE prompts you to enter the FD section. The FD section must be in line file format with no source line numbers. Type a record descriptor directly, starting at the first 01 level. After a line is entered, it cannot be modified. No REDEFINES clauses should be used, although implicit REDEFINES are acceptable. Special care should be given to insure that the key names are not ambiguous (see Figure 2-9).

```
Complete File Descriptor              Record Descriptor

FD CUSTOMER-FILE
   LABEL RECORDS OMITTED              01 CUST-RECORD.
   DATA RECORD IS CUST-RECORD.           03 CUST-AREA PIC XX.
                                         03 CUST-KEY PIC X99.
01 CUST-RECORD.                          03 NAME PIC X(30).
   03 CUST-AREA PIC XX.
   03 CUST-KEY PIC X99.                01 SALES-RECORD.
   03 NAME PIC X(30).                     03 ORDERS PIC 9(5)V99.
                                         03 PAYMENTS PIC 9(5)V99.
01 SALES-RECORD.                          03 FILLER PIC X(21).
   03 ORDERS PIC 9(5)V99.
   03 PAYMENTS PIC 9(5)V99.
   03 FILLER PIC X(21).
```

Figure 2-9 *Sample INCINQUIRE File and Record Descriptors*

When all keys have been specified, ICINQUIRE inserts the file or record descriptor into the skeletal COBOL program. It then compiles the COBOL inquiry program and displays compilation statistics, or writes them to the listfile if one is specified.

Executing the Inquiry Program

Invoke the COBOL inquiry program from the CLI. You are asked for the name of the data file to be processed. When the command list is displayed, the data file can be processed. Figure 2-10 shows the command list for the extended utility; the smaller version does not include the Read Previous (RP) and Read Previous with Lock (PL) commands. The inquiry program created by ICINQUIRE cannot access an ISAM file if the reliability flags are set on that file. Instead, the message **LOGICAL FILE STRUCTURE MAY BE CORRUPT** is displayed.

```
                    COMMAND LIST
                    ------------

D I S P L A Y                       P O S I T I O N I N G

DI    Display record field (FN3)    RE   Read exact key
#R    Repeat record                 <CR> Read next (FN1)
#     Read multiple keys            RP   Read previous (FN2)
#L    Repeat keys w/lock            RA   Read approximate
<FN3> Display current record        RL   Read next w/lock
                                    PL   Read previous w/lock
                                    BI   Beginning of file

U P D A T E  R E C O R D            I N P U T / O U T P U T

RW    Rewrite existing record       OP   Open file
WR    Write new record              OU   Change log file
RI    Reinstate record              CL   Close file w/lock
KD    Keyed delete record           CU   Close file
UL    Unlock records                SR   Stop run

Command:__
```

Figure 2-10 *ICINQUIRE Command List Screen*

Two-letter codes control all record and file processing functions. Several data files can be processed during an Inquiry program pass, provided their record length and key positions match those of your COBOL inquiry program.

Issue the OU command (see Figure 2-10) at the start of an inquiry pass to store updated records in a disk log file. Log information can also be written directly to a system printer.

When the inquiry program is ready to accept a command, it displays the `Command` prompt. Type a two-character command code, or press function key 3 (F3) to view the entire current record. The current record can also be viewed by issuing the DI command and typing the record name at the `Enter Field Name` prompt. To display all data field names, press F4 at the `Enter Field Name` prompt.

Display Commands

This section defines the various display commands. Multiple display commands #L, #, and #R use a Repeat Count Menu to display a specified number of records sequentially, forward or backward. Type in the number of records to be reviewed and the number of seconds to pass between each record display.

`Display record field: DI`

Type a field name that is not a key at the `Enter Field Name` prompt. If you enter a group name, all subordinate record fields are displayed. Press CR to display the same field of the next record. ESC terminates the command.

`Display current record: <F3>`

Press F3 at the command prompt to display the entire current record.

`Repeat record: #R`

Displays the Repeat Count Menu. Type in the number of records to review and the number of seconds to pass between each record display. The program sequentially reads forward or backward the specified number of records. To indicate a backward display, type a minus sign before the number of records.

`Read multiple keys: #`

Similar to #R, except that keys are displayed instead of whole records.

`Read keys w/lock: #L`

Similar to #R, except that each record read is locked.

`Display field names: <F4>`

When pressed at the `Enter Field Name` prompt, F4 displays all field names.

```
(0) primary-key
(1) alt-key-1
(2) alt-key-2
(3) alt-key-3
(4) alt-key-4
Key option: _
Enter Search Key:
_____________
```

Figure 2-11 *Key Option Menu*

Positioning Commands

Positioning commands RE and RA use a Key Option Menu to determine which record key and key value to use in locating a record (see Figure 2-11). Key names are displayed. Choose a key option and type in a value for the key. The inquiry program locates the record and displays the values of all its keys.

Positioning command RL displays all key values, while RE and RA display the Key Option Menu, followed by the record contents. If no alternate keys exist, the entire record contents are displayed in every case.

Read exact key: RE

Displays the Key Option Menu. Choose a key and enter a value. If no alternate key exists, all record contents are displayed. If alternate keys exist, only the key values are displayed.

Read next: <CR>, <F1>

Reads the succeeding record, based on the key last accessed. CR displays all fields in the succeeding record if it follows a record level DI command.

Read next w/lock: RL

Reads the next sequential key, based on the key last processed, and locks the record for exclusive use.

Read previous: RP, <F2>

Reads the preceding record, based on the last key accessed, and displays the record fields. Pressing CR resumes processing forward.

Read previous w/lock: PL

Reads the preceding record, based on the last key accessed, and locks the record for exclusive use.

Read approximate: RA

Displays the Key Option Menu. Enter the option number and a key value; the first key that is equal to or greater than the key is displayed. Pressing the F1 or CR keys displays the succeeding key. Pressing F2 displays the preceding key. The RA command is useful for locating a record when its precise key value is unknown.

Beginning of file: BI

Positions pointers to the beginning of the data file by primary key. There is no display associated with this command.

Update Commands

Update commands RI, KD, RW, and WR display the record key names. Type in the key contents to process a record.

Rewrite existing record: RW

Updates field values, with the exception of the primary key. The system responds as in the WR command, except that instead of merely requesting values for alternate keys, it enables you to alter their values. Blanks denote empty fields.

Write new record: WR

Updates record fields, including key values. The inquiry program requests values for the primary key and each alternate key specified in the record descriptor. After you type in key values, the prompt **Enter Field Name** appears at the bottom of the screen with three choices provided:

- Enter a field name from the data record and type in field contents.
- Press ESC to abort the WR attempt.
- Press CR to terminate the pass and update the record.

Duplicate keys and invalid keys are intercepted by the program.

```
Reinstate record: RI
```

Reinstates a logically deleted record. Enter the primary record key. ESC aborts the command.

```
Keyed delete record: KD
```

Logically deletes a record. Enter the primary record key.

```
Unlock records: UL
```

Unlocks all previously locked records. The inquiry program asks you to verify unlocking.

Input/Output Commands

If no ISAM file is currently open, only the ESC key and the input/output commands OP and SR can be executed; all other program commands are disabled.

```
Open file: OP
```

Opens an ISAM file to be processed. Type the ISAM filename to be opened. A locked file cannot be opened.

```
Change log file: OU
```

Changes the inquiry program logging device. The four logging devices are the first and second line printers, a disk file, and the display screen. If a disk file is specified, updated records are appended to the file. The log file is created if one does not already exist.

```
Close file with lock: CL
```

Closes and locks the current ISAM file.

```
Close file: CU
```

Closes but does not lock the current ISAM file.

```
Stop run: SR
```

Terminates the COBOL inquiry program. Press CR to return to Logon.

Preparing a Record Descriptor

When preparing a record descriptor as input to ICINQUIRE, some size considerations should be observed. The skeletal program contains an entire COBOL Procedure Division with directions for processing and creating ISAM files. The program record descriptor and all associated procedure statements are installed by ICINQUIRE, and then the program is compiled. This process produces a tailored ISAM inquiry program, but it also generates excessive code. To reduce the amount of code generated by the program, the record descriptor should be kept as compact and simple as possible. A descriptor with a large number of fields causes the compiler to generate correspondingly greater amounts of code for the Screen Section.

For example, an improperly organized descriptor with 23 fields can generate a program too large to run on smaller systems, whereas the same properly organized record descriptor would generate an executable program.

Some space saving steps are:

- Assign FILLER to fields that will not be modified. ICINQUIRE keeps FILLER fields in the record descriptor to retain record layout compatibility, but these FILLER fields are not displayed during program execution.
- Reduce groups to as few levels as possible to avoid duplicate coding generated by overlapping field names at each level. For example, a group that looks like this:

```
01 FIELD-1.
   02 FIELD-2.
      03 FIELD-3.
         04 FIELD-4 PIC XXXX.
```

can be replaced by this:

```
01 FIELD-1.
   04 FIELD-4 PIC XXXX.
```

The following restrictions govern record descriptors:

- To avoid screen overflow, no more than 23 elementary fields, including keys, are allowed in the record descriptor. The screen contains only 24 lines for data display.
- The COBOL REDEFINES and OCCURS clauses cannot be used in the record descriptor. However, an implicit REDEFINES at the 01 level is acceptable as long as the combined number of fields does not exceed 23.
- No elementary field, except FILLER, can exceed 80 characters in length. Characters that exceed the limit are truncated on the screen.
- If duplicate field names are used in the record descriptor, only the first occurrence of the field name is retrieved by the inquiry program.
- Numeric and alphanumeric edited fields should not be included in the record descriptor. These are not reproduced correctly on the display screen. For example:

Input Field	Display Field
PIC XX/X	PIC XX/ /

- The scaling character P can be used in the record descriptor but is not displayed. Only the displayed characters can be modified. For example:

Input Field	Display Field
PIC 99PPP	PIC 99
PIC PPP99	PIC 99

- Do not assign a PICTURE that contains characters other than X to subfields of a primary or alternate record key. This might cause the COBOL inquiry program to produce misleading displays of the key fields' contents.

Error Messages

ICINQUIRE provides error checking routines related to the preparation and online indexing of the XMODEL record descriptor. The error messages included in the program, with brief explanations, follow.

`ALTERNATE KEY MISSING IN FD FILE`

An alternate key was not specified during the ICINQUIRE dialog.

`FIELD EXCEEDS 80 CHARACTERS`

A COBOL picture clause coding restriction has been violated.

ILLEGAL FILE TYPE

The file type specified must be indexed or relative.

LEVEL NUMBER IS MISSING OR INADEQUATE

A COBOL syntax rule has been violated.

MODEL FILE CANNOT BE OPENED

No skeletal program exists in the current directory at the time ICINQUIRE is invoked from the CLI.

MORE THAN 23 ELEMENTARY FIELDS

An XMODEL coding restriction has been violated.

NO OCCURS IS ALLOWED IN FD FILE

An XMODEL coding restriction has been violated.

OUTPUT FILE MISSING

The /O switch is missing from the ICINQUIRE command line.

PARENTHESES MISSING

Syntax error exists in the record descriptor.

PERIOD MISSING IN FD FILE

A COBOL syntax rule has been violated.

PRIMARY KEY MISSING IN FD FILE

The primary key was not specified during the ICINQUIRE dialog.

RECORD NAME MAY NOT BE KEY-NAME ITSELF

A level 01 item cannot share the name of a field designated as the primary key.

RECORD KEY MUST BE ALPHANUMERIC: data-name

The PICTURE must be alphanumeric (X).

REDEFINES IS NOT ALLOWED IN FD FILE

An XMODEL coding restriction has been violated.

Runtime Error Messages

Each time the runtime system performs an I/O operation, it generates a 2-byte File Status code indicating the outcome of the operation. The inquiry program that ICINQUIRE creates provides appropriate messages for the runtime system's File Status codes. Therefore, if an I/O error occurs, the inquiry program's message, rather than the File Status code, is displayed. A list of the inquiry program error messages and the corresponding runtime system File Status codes appears in Table 2-5. For further information on File Status codes, see the *Interactive COBOL User's Guide (AOS, AOS/VS)*.

Runtime Code	Inquiry Program Error Message
24	BOUNDARY ERROR, INDEX DEPTH EXCEEDED
9B	CORRUPT INDEX STRUCTURE
10	END OF FILE
9A	FILE DESCRIPTOR INCONSISTENCY
94	FILE IN USE OR RECORD LOCKED ON READ
92	FILE NOT OPEN OR OPEN ON FILE WITH LOCK
9F	LOGICAL FILE STRUCTURE MAY BE CORRUPT
91	FILE OPEN ERROR
9C	INDEX (.NX) FILE IS FULL
22	INVALID KEY (DUPLICATE KEY)
21	INVALID KEY, OUT OF SEQUENCE
23	INVALID KEY, RECORD NOT FOUND
96	OPEN ERROR (DIRECTORY NOT INITIALIZED)
30	PERMANENT HARDWARE ERROR
9E	RECORD LOCK LIMIT EXCEEDED
97	SIMULTANEOUS ISAM OPENS EXCEEDED
34	SYSTEM ERROR (DISK SPACE EXHAUSTED)
98	NOT ENOUGH DISK SPACE TO CREATE FILE

Table 2-5 Inquiry Error Messages with Runtime File Status Codes

ISAMVERIFY tests the integrity of ISAM files. It checks the validity of the system information that actually defines the logical structure of a file.

ISAMVERIFY does not alter the file except for setting and clearing the ISAM file reliability flags. It sets the flags when the file's logical structure is bad and clears them if the logical structure is sound. Note that if a file's logical structure is good and the reliability flags are set, ISAMVERIFY clears them. It does not attempt to fix any errors it may find; you must do this by running REBUILD, REORG, or using a backup copy of the file.

ISAMVERIFY opens a file exclusively. Therefore, if ISAMVERIFY is running on a file, the file cannot be accessed; and the utility cannot access an ISAM file while it is being used by another program.

ISAMVERIFY tests four areas: (1) data and index file system information, (2) data records, (3) key access path descriptors, and (4) key entries in B-trees, which are logical storage structures in the .NX portion of the file. ISAMVERIFY reports that the logical structure of a file is corrupt if:

- The values of system words in the file are not within their specified ranges. System words, which are in block 0 of the index (.NX) and data (.XD) portions of a file, describe the file contents.
- The same system words contained in the .NX and .XD portions of the file are not equal.
- The values of system words in the file do not agree with computed values.
- Key strings in the .NX portion and in the associated record are not identical.
- Pointers to records do not address record-aligned boundaries.
- Errors are detected in the link fields of records.
- Errors are detected in the B-trees, the logical structures in the .NX portion that store keys.
- Counts of records, keys, and index blocks reveal inconsistencies with values of system words in the file.

Use ISAMVERIFY regularly to evaluate the structural integrity of files. Run it immediately after any hardware problems or power failures.

Procedure

Invoke ISAMVERIFY at the CLI by the following command:

```
ISAMVERIFY filename-1[/R] [...filename-n[/R]] [auditfile/A]
```

where /R designates a relative file and /A designates an audit file. If no switch appears with the input filename, the program assumes it is indexed. The filename arguments are position-independent. Only one audit file can be specified; if it already exists, it is deleted and a new one is created. Output is displayed on the screen whether or not an audit file is specified. ISAMVERIFY does not accept templates in the filename.

ISAMVERIFY attempts to process both portions of a file before it begins to read the next file. If the utility detects an error in a file, it sends a message to the screen and, if specified, to the audit file. Each message specifies which portion of the file is corrupt. The error message information is needed to determine (1) whether REBUILD or REORG should be used and (2) what information must be supplied in the REORG or REBUILD command line.

When ISAMVERIFY begins testing a file, it displays the message

```
filename IS BEING VERIFIED
```

It checks that the .NX and .XD portions of the file are present. If either portion is missing, ISAMVERIFY stops processing the file. If the .NX portion is missing, the REBUILD utility can reconstruct it. However, if the data portion is gone, you must substitute a backup copy of the file. Run ISAMVERIFY on the backup to be sure it is correct before you use it.

If both portions of the file exist, ISAMVERIFY begins checking block 0 of the .NX and .XD portions and displays the message:

```
Checking .NX and .XD Block 0
```

If no discrepancies are found in checking the headers, it begins checking the data portion of the .XD file. It confirms processing of the data records in the .XD portion, displays the key path, and displays the number of records that have been read (the Record Number Update) at periodic intervals.

If ISAMVERIFY detects an error, it stops processing the .XD portion and begins processing the .NX portion. If these is a problem in the .XD link structure, ISAMVERIFY displays the message `XD PORTION IS CORRUPT`, followed by the specific error message. If no errors are found in the .XD portion, ISAMVERIFY displays `XD PORTION IS VALID`, and writes this message to the auditfile, if one is specifed.

The utility then confirms that it is checking the .NX portion and examines the remaining .NX blocks and the B-tree structure. If it detects a problem in the B-tree structure, it displays the message `NX PORTION IS CORRUPT`, followed by the specific error message. If no errors are found in the .NX portion, ISAMVERIFY displays `NX PORTION IS VALID` and writes the message to the audit file, if one is specifed.

After ISAMVERIFY has finished processing all the specified files, it displays a summary message indicating the number of files processed and the number of files with errors. If it detects errors in a file's logical structure, follow one of these procedures:

- Run the REBUILD utility on the file. Use REBUILD if either the .NX or the .XD portion of the file is corrupt. As an extra precaution, run ISAMVERIFY after rebuilding the file.
- Run the REORG utility on the file. REORG should be used if both the .NX and .XD portions of the file are corrupt. After REORG has run on the file, run ISAMVERIFY again.
- Substitute a backup file for the corrupted file. Run ISAMVERIFY on the backup file before substituting it for the corrupted file.

Abnormal Termination

The *next-record-to-append pointer* tells the system where the next logical record of the file is to be located. The value for this pointer is stored both in the header of the .NX portion and that of the .XD portion of the file. When ISAMVERIFY begins operating on a file, it checks whether both pointer values in the headers agree, and, if they do, continues checking other values.

It is possible for the pointer values in the headers to be incorrect, even though they agree. The runtime system copies the value for the next-record-to-append pointer into memory. Whenever a program writes a new record, the value in memory is updated. However, the values in the .NX and .XD headers on disk are updated only periodically. Therefore, if your system goes down or is abnormally terminated, the values in the headers may agree, but they are probably out of date.

ISAMVERIFY does not discover this discrepancy until it examines the .XD portion. If the link points beyond the next record to append, but it is within the end of file, ISAMVERIFY follows the link and displays this message:

```
INVALID NEXT RECORD TO APPEND POSITION
```

To recover the file with REORG, assume that both portions of the file are invalid and supply the header information, using REORG's /X switch. See "Invalid .XD Header" in the REORG section.

When you supply the header information to REORG, include the total number of records in the form *num-recs*/R. To get this total, add the total number of records and the number of records beyond the next record to append, which ISAMVERIFY displays and writes to the audit file, if specified. If you do not specify *num-recs*/R, the recovered file will most likely include one record that is all nulls.

Error Messages

The following list explains ISAMVERIFY error messages and how to correct the error. For all file errors, ISAMVERIFY specifies whether the .NX or the .XD portion of the file is corrupt. This information is necessary to determine whether to use REBUILD or REORG on the corrupt file. If ISAMVERIFY's error messages indicate that only one portion is corrupt, use REBUILD on the file. If both portions of the file are corrupt, use REORG.

In certain cases, only one portion of the file may be corrupt, but you must assume that both portions are corrupt. These instances involve information stored in block 0 of the .XD and .NX portions. This information includes the number of alternate keys, the record length, the next-record-to-append pointer, and tables of information describing each key. ISAMVERIFY compares the versions in the .NX and the .XD, but it cannot determine which version is correct if they do not agree. In these cases, which are noted below, you must assume that both portions of the file are corrupt and use REORG. Of course, you can always substitute a backup file.

```
AUDIT FILE CANNOT BE DELETED
```

A delete-protected file was specified as the audit file. Either change the file's attributes or select a different audit file.

```
BAD TOP BLOCK NUMBER
```

The .NX portion is corrupt. The top-level index block must be less than the next block to append. ISAMVERIFY stops processing the file.

```
BLOCK NUMBER OR BLOCK POINTER INCONSISTENCY
```

The .NX portion is corrupt. The relative block number in the block header must be the same as the block number pointer used to logically find the block. ISAMVERIFY stops processing the file.

```
BLOCK ZERO OF .NX FILE COULD NOT BE READ
BLOCK ZERO OF .XD FILE COULD NOT BE READ
```

The .NX or .XD file contains fewer than 512 bytes. Since ISAM files must be greater than 512 bytes, this file was not created by MINISAM. Delete the file.

BLOCK 0 SYSTEM INFORMATION HAS NON-FATAL ERROR

ISAMVERIFY continues processing and displays the non-fatal error.

B-TREE SEARCH LEVEL NOT BETWEEN 1 AND 6

The .NX portion is corrupt. The most likely possibility is that a loop was detected when searching the B-tree. ISAMVERIFY stops processing the file.

DEPTH OF B-TREE DISCREPANCY

The .NX portion is corrupt. The examined depth of the B-tree must equal the depth of the B-tree listed in block 0 of the .NX portion. ISAMVERIFY stops processing the file.

DUPLICATE ALTERNATE KEYS INCORRECTLY LINKED

An error exists in the .XD portion. Records with duplicate alternate keys must be in strictly ascending order. This message indicates that the record position of the previous record is greater than the record position of the current record. Most likely, revision 1.00 of Interactive COBOL's COLLAPSE or revision 4.40 or less of CS COLLAPSE has been run on the file. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX portion. If you use a backup file, make sure that it has not been collapsed using the above revisions.

EXTRA INDEX BLOCKS ALLOCATED

To detect whether index blocks are being duplicated, ISAMVERIFY maintains a count of all the blocks examined. This count must be one less than the next available block number, which is kept in block 0 of the index portion. This message indicates an error in the .NX portion. ISAMVERIFY stops, but it has finished processing all B-trees in the .NX portion.

FILE DOES NOT EXIST

ISAMVERIFY was unable to find the .NX and .XD portions of the specified file. You may have spelled the filename incorrectly, your searchlist may be incorrect, or you may be in the wrong directory.

ILLEGAL SWITCH SPECIFIED

The command line contains an illegal switch. Enter the correct command line.

INCORRECT B-TREE NEXT BLOCK VALUE

The .NX portion is corrupt. In the .NX, the value of the next block to append must be less than or equal to the file size divided by 512. ISAMVERIFY stops processing the file.

INCORRECT NUMBER OF ALTERNATES

The number of alternate keys is not between 0 and 4, or the values in the .NX and .XD portions of the file do not agree. Either or both portions of the file may be corrupt. ISAMVERIFY stops processing the file. When invoking REORG, assume that both portions of the file are corrupt, and supply the header information in the command line. See "Invalid .XD Header" in the REORG section.

INDEX DEPTH NOT BETWEEN 0 AND 5

The .NX portion is corrupt. The index level depth for each key path must be between 0 and 5. ISAMVERIFY stops processing the file.

INVALID FIRST RECORD POINTER

The .XD portion is corrupt. If the first record pointer in the .XD header block is greater than 0, this message indicates that the pointer is not less than the next-record-to-append field or is not record-boundary aligned. If the first record pointer is 0, this message indicates that the .XD portion contains records. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

INVALID INDEX BLOCK KEY COUNT

The .NX portion is corrupt. The number of entries in the key block header of the B-tree index block must be less than or equal to the maximum number of entries allowed in a block. ISAMVERIFY stops processing the file.

INVALID INDEX BLOCK NUMBER

The .NX portion is corrupt. The prefix of each key in a nonterminal node must have a valid index block number in the first word. ISAMVERIFY stops processing the file.

INVALID KEY LENGTH

The key length must be between 1 and 100 bytes and must be less than the record length. The .NX portion is corrupt. The .XD portion may also be corrupt; ISAMVERIFY stops processing the file before it gets to the .XD. If you run REORG, describe the record length and the key position and length for each key.

INVALID KEY OFFSET IN RECORD

The key offset must be less than or equal to the record length minus the key length. ISAMVERIFY reports the key offset as one less than defined in your COBOL program. The .NX portion is corrupt. The .XD portion may be corrupt; ISAMVERIFY stops processing the file before it gets to the .XD. If you run REORG, describe the record length and the key position and length for each key.

INVALID KEY PATH NUMBER IN INDEX BLOCK

The .NX portion is corrupt. The key path number must be the same for all the blocks in a key path B-tree. ISAMVERIFY stops processing the file.

INVALID KEY STRING SEQUENCE

The .XD portion is corrupt. The primary key string value must be greater than, and the alternate key string value must be equal to or greater than, the key string value of the previously examined record. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

INVALID NEXT-RECORD-TO-APPEND POSITION

The next available record position is not greater than 512 bytes, or the values in the .NX and the .XD portions do not agree. ISAMVERIFY stops processing the file. Either portion of the file may be corrupt. However, since it cannot be determined which is corrupt, assume that both are. Use REORG, and supply the header information in the command line. See "Invalid .XD Header" in the REORG section.

If this message appears after ISAMVERIFY begins processing the .XD portion, follow the same procedure. However, if you invoke REORG without specifying the number of records, you are likely to get one record that contains all nulls.

INVALID RECORD COUNT

The .XD portion is corrupt. The number of records examined must not exceed the total number of records, which ISAMVERIFY derives from the values contained in the header block. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

INVALID RECORD LENGTH

The logical record size (excluding record header bytes) is not between 1 and 4096, or the .NX and .XD values in block 0 do not agree. ISAMVERIFY stops processing the file. When invoking REORG, assume that both portions of the file are corrupt, and supply the header information in the command line. See "Invalid .XD Header" in the REORG section.

INVALID RELATIVE WORD STRING SEQUENCE

In a relative file, the present record's relative word must be greater than the previous record's relative word. ISAMVERIFY stops processing the .XD portion and begins processing the .NX.

KEY LENGTH-HALF BLOCK POINTER INCONSISTENCY

The offset of the half block pointer, calculated by using the key length, must agree with the value in the key path descriptor table. Both portions of the file may be corrupt; ISAMVERIFY does not process either portion.

KEY LENGTH-KEYS PER BLOCK INCONSISTENCY

The maximum number of keys in an index block, computed by using the key length value, must agree with the value in the key path descriptor table. Both portions of the file may be corrupt; ISAMVERIFY does not process either portion.

KEY LENGTH-KEYS PER HALF BLOCK INCONSISTENCY

The maximum number of keys in half of an index block, computed by using the key length value, must agree with the value in the key path descriptor table. Both portions of the file may be corrupt; ISAMVERIFY does not process either portion.

KEY LENGTH-WORDS PER HALF BLOCK INCONSISTENCY

The number of words required by the maximum number of keys in half an index block must agree with the value in the key path descriptor table. Both portions of the file may be corrupt; ISAMVERIFY does not process either portion.

KEY LENGTH-WORDS PER KEY INCONSISTENCY

The key entry, as computed by the key length, and the key entry in block 0 must be the same. Both portions of the file may be corrupt; ISAMVERIFY does not process either portion.

KEY STRINGS IN .NX FILE AND ASSOCIATED RECORD DO NOT MATCH

The .NX portion is corrupt. The key string in the record must be identical with the key string in the B-tree that addressed the record. With a relative file, the key string in the index block must be identical to the relative word in the record header. ISAMVERIFY stops processing the file.

KEYS IN B-TREE NOT IN ASCENDING ORDER

The .NX portion is corrupt. Keys in B-trees must be in strict ascending order. In the case of alternate keys, the feedback, which is appended to each alternate key to ensure that no duplicates exist, must be in ascending order as well. ISAMVERIFY stops processing the file. A likely cause of this error is that revision 1.00 of Interactive COBOL's COLLAPSE or revision 4.40 or less of CS COLLAPSE was used on the file. If you use a backup file, be sure that it has not been collapsed by these versions.

LINK BOUNDARY ERROR

The .XD portion is corrupt. Values for link fields are invalid. These invalid values are 1 to 511 and anything beyond the next record to append. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

LINK NOT RECORD ALIGNED

The .XD portion is corrupt. The link field must be record-boundary aligned. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

LINK POINTER BEYOND NEXT RECORD TO APPEND

The .XD portion is corrupt. The link field is equal to or greater than the next record to append; it should be less than the next record to append. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

MAJOR REVISION NUMBER IS NOT CURRENT

The major revision numbers of the .NX and .XD portions must be current. Two possibilities are: (1) the revision levels were current but a system problem caused them to be changed, or (2) the file is out of date. In either case, ISAMVERIFY does not set the reliability flags and continues to process the file.

NEXT-RECORD-TO-APPEND POINTER IS INVALID

The next-record-to-append pointer is greater than the actual file size, or the values in the .NX and .XD portions for the next-record-to-append pointer do not agree. ISAMVERIFY stops processing the file. Assume that both portions of the file are corrupt. Run REORG, and supply the header information in the command line. See "Invalid .XD Header" in the REORG section.

NO FILENAME ARGUMENTS

Reenter the command line, specifying filename arguments.

NO RECORDS IN FILE

The .XD portion of the file does not contain any records. ISAMVERIFY continues to process the .NX portion of the file. If the .NX portion has a correct logical structure, ISAMVERIFY does not count this file as one with errors when it displays its summary message.

NUMBER OF KEYS NOT EQUAL TO NUMBER OF RECORDS

ISAMVERIFY maintains a count of all the keys examined. For each key path, the count must equal the number of physical records. This message indicates that the .NX portion is corrupt. ISAMVERIFY stops processing the file.

.NX PORTION COULD NOT BE OPENED

Most likely, someone has already opened the .NX portion of the file.

.NX PORTION OF FILE IS MISSING

The index portion of the specified file could not be located. ISAMVERIFY stops processing the file. Check the location of the file before running a restructuring utility or using a backup file.

PREMATURE END OF FILE

The .XD portion is corrupt. Links have indicated a logical end of file before the appropriate number of records (based on the next-record-to-append pointer) have been examined. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

PRIMARY KEY HAS REWRITE BIT SET

The primary key cannot be rewritten. If this bit is set, the .NX portion is corrupt. ISAMVERIFY stops processing the file.

RECORD POINTER FROM KEY PREFIX BEYOND LAST RECORD

The record pointer in the key header is pointing beyond the end of the data file, indicating that the .NX portion is corrupt. ISAMVERIFY stops processing the file.

RECORD POINTER IS NOT RECORD BOUNDARY ALIGNED

The .NX portion is corrupt. The record pointer must address a boundary where a record entry, including links and relative word, begins. ISAMVERIFY stops processing the file.

RELATIVE WORD NOT SUBSTRING OF PRIMARY KEY

If you are running ISAMVERIFY on a relative file, check that the /R switch was specified in the command line. Otherwise, the .XD portion is corrupt, because the relative word must be the same as the first two bytes of the primary key. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

SYSTEM BLOCK NUMBER OF .NX PORTION NOT ZERO

The .NX portion has a non-fatal error. The system block number of the .NX portion of the file must be zero. If ISAMVERIFY finds no other errors in the .NX file, assume that all key access paths in the .NX portion are logically correct. ISAMVERIFY continues processing.

WARNING: ONE OR MORE B-TREES ARE FULL

ISAMVERIFY continues to operate. You can read the file but you cannot write or rewrite records without repacking the index structure or purging logically deleted records.

WARNING: .XD PORTION FILE ELEMENT SIZE IS NOT A MULTIPLE OF 4
WARNING: .NX PORTION FILE ELEMENT SIZE IS NOT A MULTIPLE OF 4

All ISAM files under AOS and AOS/VS must have an element size that is a multiple of 4. If these are the only messages that ISAMVERIFY displays, the reliability flags are not set and you do not have to run REBUILD or REORG. Instead, create a temporary file with an element size of 4, copy the .XD portion into this file, delete the original .XD portion, and rename the temporary file. MINISAM_CONVERT.CLI is a macro that performs this procedure. ISAMVERIFY continues operating.

WARNING: .NX PORTION FILE SIZE IS NOT ON A 2KB BOUNDARY
WARNING: .XD PORTION FILE SIZE IS NOT ON A 2KB BOUNDARY

If an ISAM file is to be open for INPUT by an Interactive COBOL program, the .NX and .XD portions must each be on a 2-KB boundary. The REBUILD utility and the MINISAM_CONVERT macro change the file size to a 2-KB boundary. If a file is to be used only for OUTPUT or I-O, however, you may ignore this message. ISAMVERIFY does not set the reliability flags upon detecting this condition.

.XD PORTION COULD NOT BE OPENED?

Most likely, someone has already opened the .XD portion.

.XD PORTION OF FILE IS MISSING

ISAMVERIFY could not locate the data portion of the file. Check the location of the .XD portion. If it cannot be located, use a backup file. ISAMVERIFY stops processing the file after this error is encountered.

REBUILD is a utility that reconstructs corrupted indexed and relative files. It is able to restore the integrity of the logical file structure of the index (.NX) or the data (.XD) portion of the file if the other portion is intact.

REBUILD is able to:

- Reconstruct a corrupted .NX file from an .XD file
- Reconstruct a corrupted .XD file from an .NX file
- Round the .NX and .XD portions to the next 2 KB boundary

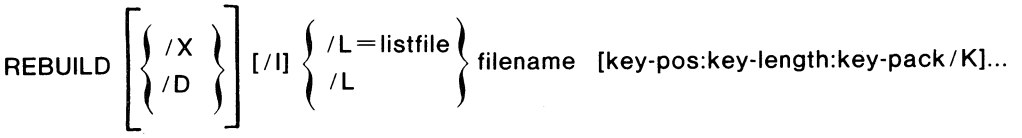
If the .XD portion of the file is sound, REBUILD attempts to restore corrupted links by using the .NX file. REBUILD can reconstruct an ISAM file, but it cannot fix data within a file if the data itself is corrupt.

REBUILD cannot restructure the key or record format of the file or reorganize a file from one type to another. REBUILD cannot be used on a file if both portions of the file are corrupt; if this occurs, use REORG.

REBUILD should be run to align the boundaries of a valid ISAM file or to reconstruct an ISAM file in which either portion is corrupt. ISAMVERIFY reports how the file has been corrupted; you must supply the information necessary for REBUILD to recover the corrupted files.

Procedure

REBUILD is invoked at the CLI. Its general form is as follows:



filename
The name of the file to be rebuilt.

key-pos
The starting byte position of the key in the record. If no key position is specified, it is assumed to be 1.

key-length
The length of the key in bytes.

key-pack
The packing limit percentage for the key specified. The packing density can be between 50% and 99%. If no packing density is specified, it is assumed to be 75%.

The packing density is an arbitrary number. For example, if the file had a packing density of 60% before it was corrupted, you can specify the same or a different density. The only considerations should be the packing limitations (50% to 99%) and the access method for the file. Note that REBUILD packs the .NX portion only.

Global Switches

/X Rebuild the index (.NX) portion of the file. This switch indicates that the .NX does not exist or is corrupt if it does exist. If this switch is used, the .XD portion must have passed all of ISAMVERIFY's tests for file integrity. REBUILD follows the links of the .XD file to rebuild the .NX file. The new .NX has the user-specified packing limit for each key.

/D Rebuild the data (.XD) portion of the file. If this switch is used, the .NX portion of the file must have passed all of ISAMVERIFY's tests for file integrity. REBUILD reconstructs the .XD file and its links using the .NX file.

Note: The /X and the /D switches are mutually exclusive; both cannot be used in one command line. If neither /X nor /D is used, REBUILD assumes that the index or data portion is not on a 2KB boundary.

/L Audit information is appended to the listing file. If a filename argument is used after the /L switch, audit information is appended to *filename.*

/I Invoke REBUILD interactively. However, REBUILD cannot be run via QBATCH if it is used interactively.

Local Switches

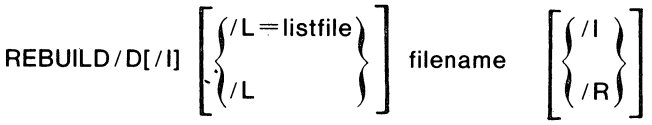
/I Indicates an indexed file. If neither /I nor /R is specified, an indexed file is assumed.

/R Indicates a relative file.

/K Identifies the key information: position in record, length, and packing density. REBUILD interprets the number of alternate keys as the number of times the /K switch is specified minus 1. The utility checks the number of alternate keys specified in the command line against the number of alternate keys stored in block 0 in the .XD portion of the file. If this information is not identical, REBUILD displays an error message.

Rebuilding the .XD Portion of an ISAM File

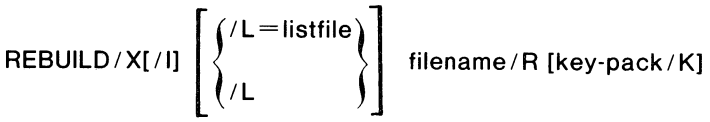
To rebuild the .XD portion of an indexed or relative file, enter:



Because you are using the good .NX to rebuild the .XD, no key information is needed. If you enter any key information, REBUILD displays an error message.

Rebuilding the .NX Portion

To rebuild the .NX portion of a relative file, enter:

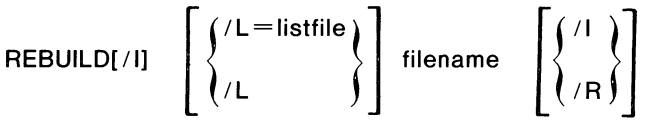


The key in a relative file is always in the same position of the record and is always two bytes long. Therefore, the key position and the key length are not needed; REBUILD displays an error message if you enter them. The packing density, *key-pack*/K, is the only key information to supply.

To rebuild the .NX portion of an indexed file, you must supply the key position, key length, and key packing density for each key.

Correcting Boundary Alignment

To correct the file sizes so that they are on a 2KB boundary, enter



Both the index and data portions must be present and must not be corrupt.

Using REBUILD Interactively

REBUILD can be used in batch or interactive mode. The global /I switch invokes REBUILD interactively.

The differences between batch and interactive mode are as follows:

- Interactive mode allows you to exit from REBUILD before it begins to operate on a file. In batch mode, you cannot exit from REBUILD after entering the command line, except by entering the CTRL-C CTRL-B sequence. If you enter this interrupt sequence after REBUILD begins processing a file, you may have further corrupted your file.
- In interactive mode, audit information is displayed on the screen. There is no screen display if the global /I switch is not used.
- The batch mode allows REBUILD to be run via QBATCH when the system is not being used heavily.

Example

ISAMVERIFY has been run on the file DATA and has specified that the index portion is corrupt. The user enters the following command line:

```
REBUILD/X/I DATA :15:/K 16:10:/K 20:17:85/K 40:5:70/K 46:12:95/K
```

The following is an analysis of the command line:

/X	Tells REBUILD to follow the links of the .XD portion of the file to rebuild the .NX portion. The .XD portion has passed all ISAMVERIFY tests for structural integrity.
/I	Indicates REBUILD's interactive mode.
DATA	The file that REBUILD is to process. The local /I switch is omitted, so an indexed file is assumed.
:15:/K	The primary key specifier. No key position is specified, so it is assumed 1. The key length is 15 bytes. The packing density is not specified, so it is assumed to be 75%. The /K switch indicates that the information pertains to the keys.
16:10:/K	The first alternate key specifier. The key position is 16, the key length 10 bytes, and the packing density 75%.
20:17:85/K	The second alternate key specifier. The key position is 20, the key length 17 bytes, and the packing density 85%.
40:5:70/K	The third alternate key specifier. The key position is 40, the key length 5, and the packing density 70%.
46:12:95/K	The fourth alternate key specifier. The key position is 46, the key length 12, and the packing density 95%.

After the command line is entered, REBUILD displays the starting time and date. This information, and all other information displayed on the screen, is entered in the audit file.

REBUILD next displays a screen similar to the one shown in Figure 2-12. If you are running the utility in interactive mode, it prompts:

```
READY TO REBUILD .NX FILE
CONTINUE? ([YES] OR NO)
```

The default is yes; enter Y, YES, CR, or NL. If you enter NO or N, REBUILD terminates without accessing the corrupted file. This is the last point that you can exit from REBUILD and be sure that the file is not touched. The batch mode version of REBUILD has no such option.

If you respond Y to the **CONTINUE** prompt, REBUILD begins to fix the .NX portion. The utility displays the number of indexed blocks written to the .NX file, and the elapsed time, which is updated periodically. All information displayed on the screen is entered in the audit file, if one is specified. When REBUILD has fixed the corrupted portion, it displays the total elapsed time and returns you to the CLI.

```
FILENAME is                                DATA
Number of records in .XD file is:            6
Maximum # of records in .XD file is:         6
                                      From     From     From
                                      Old .XD  Old .NX  User
                                      -------  -------  ------
Number of ALTERNATE keys is:              4        4
Record size is:                         342      342
PRIMARY KEY - Position in record is:                 1        1
              Length is:                            15       15
ALT KEY # 1 - Position in record is:                16       16
              Length is:                            10       10
ALT KEY # 2 - Position in record is:                20       20
              Length is:                            17       17
ALT KEY # 3 - Position in record is:                40       40
              Length is:                             5        5
ALT KEY # 4 - Position in record is:                46       46
              Length is:                            12       12
```

***Figure 2-12** REBUILD Screen*

Messages

REBUILD messages generally fall into four types:

- Syntax messages, which denote an error in the command line syntax. Reenter the command line correctly if a syntax message is displayed.
- Status messages, which are informative.
- Internal consistency errors, which denote an error in REBUILD's internal checks or a hardware problem. These errors are fatal. Try running REBUILD again. If the same error message occurs, contact your Data General representative. As a stopgap measure, run REORG on the same file or substitute a backup file.
- File errors, which indicate that REBUILD has found an error in the portion that was not declared corrupt. REBUILD stops operating and sets the ISAM reliability flag in that portion. Run ISAMVERIFY on the file, and proceed according to the information that ISAMVERIFY produces.

```
ARITHMETIC ERROR
```

Internal consistency error.

```
ATTEMPT TO REBUILD THE DATA (.XD) FILE HAS FAILED
ATTEMPT TO REBUILD THE INDEX (.NX) FILE AS FAILED
```

Use REORG on the file instead.

```
BLOCK 0 OF THE .NX FILE IS CORRUPT
BLOCK 0 OF THE .XD FILE IS CORRUPT
```

File error. You probably have declared one portion of the file corrupt, and REBUILD has found a possible instance of corruption in block 0 of the other portion. Use REORG on the file instead. See "Invalid .XD Header" in the REORG section.

```
CORRUPT LINKS WERE FOUND IN THE .XD FILE
UNABLE TO RESOLVE BAD LINKS WITH THE .NX FILE
```

The key in the .XD portion cannot be found in the .NX portion, therefore, the link cannot be restored. You may use REORG. The key may still be incorrect after using REORG, but your system will be able to use the records.

```
GLOBAL /X AND /D SWITCHES ARE MUTUALLY EXCLUSIVE
```

Syntax error.

```
INDEX DEPTH OVERFLOW WHILE REBUILDING THE .NX FILE
```

The .XD file may contain too many records; you may have specified too low a packing density.

```
INVALID FILE REVISION IN .NX FILE
INVALID FILE REVISION IN .XD FILE
```

File error. The file revision in the portion you have declared valid is not current.

```
INVALID GLOBAL SWITCH SPECIFIED
```

Syntax error. Permissible global switches are /X, /D, /I, /L, and /L=*filename*.

```
INVALID KEY ARGUMENT SPECIFIED
```

Syntax error.

```
INVALID KEY LENGTH ARGUMENT SPECIFIED
```

Syntax error. You may have specified a nonnumeric key length or one that is greater than 4096, the maximum record size.

```
INVALID KEY POSITION ARGUMENT SPECIFIED
```

Syntax error. You may have specified a nonnumeric key position or one that is greater than 4096, the maximum record size.

```
INVALID LOCAL SWITCH SPECIFIED
```

Syntax error.

```
INVALID NEXT RECORD TO APPEND POINTER IN THE .NX FILE
```

One or more of the following conditions is not met: the pointer must be record-boundary aligned, must be greater than 512 bytes, and must be less than or equal to the file size.

INVALID NEXT RECORD TO APPEND POINTER IN THE .XD FILE

One or more of the following conditions is not met: the pointer must be record boundary aligned, must be greater than 512 bytes, and must be less than or equal to the file size.

INVALID NUMBER OF ALTERNATE KEYS FOUND IN THE .NX FILE

The number of alternate keys must be between 0 and 4.

INVALID NUMBER OF ALTERNATE KEYS FOUND IN THE .XD FILE

The number of alternate keys must be between 0 and 4.

INVALID PACKING LIMIT ARGUMENT SPECIFIED

Syntax error. A number between 50 and 99 must be entered.

INVALID RECORD ADDRESS ENCOUNTERED

The record address is not record-boundary aligned. You may have entered the /X global switch, which indicates that the .NX is bad, and REBUILD has found an invalid link, which means that the .XD file is bad. Run ISAMVERIFY and take the appropriate action.

INVALID RECORD LENGTH FOUND IN THE .NX FILE
INVALID RECORD LENGTH FOUND IN THE .XD FILE

The record length must be between 1 and 4096. Either of these messages indicates that ISAMVERIFY has not been run before REBUILD. Run ISAMVERIFY to determine where the file is corrupt, then run the appropriate utility.

KEY COLLATING SEQUENCE ERROR

REBUILD has detected a collating error in the file. You can recover the file with REORG.

KEY IS BEYOND END OF RECORD

Syntax error.

LOCAL /K SWITCH IS VALID ONLY WITH THE GLOBAL /X SWITCH

Syntax error. You have tried to specify key information with the global /D switch. This is illegal.

MULTIPLE FILE NAMES SPECIFIED

Syntax error.

NAME OF FILE TO BE REBUILT IS NOT THE FIRST ARGUMENT

Syntax error.

NO FILE NAME SPECIFIED

Syntax error.

NO KEY ARGUMENTS SPECIFIED

Syntax error. If you use the /X switch, you must include the key position, length, and packing density for each key.

RECORD ADDRESS AND FEEDBACK IN KEY ENTRY DO NOT MATCH

The probable cause is that the .NX is corrupt, but you declared it correct.

THE DATA (.XD) FILE MAY BE CORRUPT

File error.

THE INDEX (.NX) FILE WAS FOUND TO BE CORRUPT

File error.

THE LAST KEY ENTRY IN THE .NX FILE IS INCORRECT

File error.

THE .XD FILE IS A NULL FILE.

Your file does not contain any records.

TIME CONVERSION OVERFLOW

Internal consistency error.

TOO MANY ARGUMENTS IN COMMAND LINE

Syntax error.

TOO MANY BLOCKS WRITTEN TO .NX FILE

Internal consistency error. More than 64K blocks were written to the .NX file.

TOO MANY KEY ARGUMENTS SPECIFIED

Syntax error.

WRONG NUMBER OF KEY ARGUMENTS SPECIFIED

Syntax error. The number of keys specified in the command line and the number of keys that exist as determined from the .XD header do not agree.

YOU MAY NOT SPECIFY THE GLOBAL /D SWITCH WHEN THE .NX FILE IS CORRUPT
YOU MAY NOT SPECIFY THE GLOBAL /X SWITCH WHEN THE .XD FILE IS CORRUPT

Syntax error.

YOU MAY NOT USE REBUILD WHEN BOTH FILES (.NX AND .XD) ARE CORRUPT

Both reliability flags are set, indicating that both portions of the file are corrupt. Run ISAMVERIFY and take the appropriate action.

REORG is a file conversion utility that enables you to change the structure of a file to improve processing speed and flexibility. REORG's capabilities include:

- Recovering ISAM files
- Converting among indexed, relative, fixed-length sequential, variable-length sequential, and line sequential file types
- Changing the key access of indexed files and sorting by different keys
- Purging logically deleted records from ISAM data files
- Selecting and rearranging fields in data records and inserting editing characters

Reliability Checking

If an ISAM file's reliability flags are set and you attempt to use it as an input ISAM file without the /D or /E switch, REORG displays the following error message:

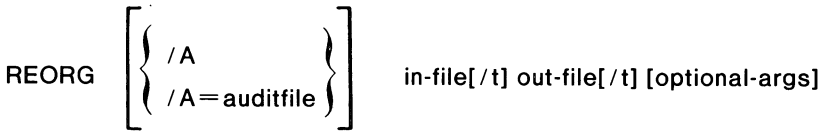
```
LOGICAL FILE STRUCTURE MAY BE CORRUPT
```

However, if a file's reliability flags are set and you are using the .XD portion to recover, REORG does operate. It uses the data portion to read the file sequentially and to recover the index portion.

Procedure

REORG is called through the CLI and runs under operating system control. The original (input) file is preserved, and a converted (output) file is created under a different name. Audit data on REORG's processing can be directed to the display terminal or stored in disk files.

The basic command line is:



The following arguments are required in the command line, and must appear in the order shown above.

in-file
: The name of the input file, which must be the first parameter after the optional audit file. If a value for the /t switch is omitted, an indexed file is assumed (see "Local Switches" below). Do not specify the .NX or .XD extension.

out-file
: The name of the output file, which must follow the input filename. The same filename should not be used for the input and output files if both are sequential. If the output file does not exist, REORG creates it; the packing density is 50%. If the output file is an existing indexed or relative file, REORG inserts the additional records, overwriting duplicates. Do not specify the .NX or .XD extension. If the output file is an existing sequential or line file, REORG appends new records, retaining all original records. If a value for the /t switch is omitted, an indexed file is assumed (see Local Switches below).

Global Switches

/A Directs audit information concerning REORG's processing to the display screen. Audit information consists of the date and time of execution, input and output filenames and types, record and key sizes and formats, and input and output record counts.

/A=auditfile
Stores audit information in the specified file. If the file does not exist, it is created. If the file exists, information is appended to it. This switch cannot be used with the /A switch above. The auditfile contains the date and time of execution, the input and output filenames and types, the record and key sizes and formats, and the input and output record counts.

Note: Do not give the audit and output files the same name.

Local Switches

/t Input or output file type. If no switch is used, REORG assumes an indexed file. The switches for input and output files are the same, except for the /Q switch, which can only be used with an output file.

Switch	File Type
/S	Fixed sequential file (fixed-length records)
/V	Variable sequential file (variable-length records)
/L	Line sequential file (variable-length records terminated with CR or NL)
/R	Relative file
/I	Indexed file
/Q	Relative output file. Records are not renumbered.

When a line sequential file is output from REORG, the record description must not contain any data fields described as computational. The results are unpredictable and can cause a system error. If a line sequential file is used as input, blank lines and form feeds are not written to the output file.

Appendix A contains a matrix that shows the legal combinations of local switches.

Optional Arguments

The syntax of the optional arguments and a brief description of their uses appear in Table 2-6. The optional arguments can occur in any order in the command line. However, all alternate key specifiers, field specifiers, and insertion specifiers must appear in the sequence desired in the output file. Optional arguments are required in certain types of file conversions. Table 2-7 shows which optional arguments are required for the specified conversion.

Argument	Form	When Used
Relative key specifier	start-rel-key/K	Number of first record key when the output file is relative
Record size specifier	in-rec-size/I	Size of input record in bytes when input file is fixed sequential
	out-rec-size/O	Size of output record in bytes. Required when the input record is variable length and no output field specifiers define output record length
Access key specifier	access-key/S	Input file is indexed and will be accessed by an alternate key
Key specifier	key-pos:key-length/K	Specifies key when output file is indexed. Specifies primary key if output file is to include alternates
	alt-key-pos:alt-key-length/A	Specifies alternate key when output file is indexed. Up to 4 can be specified.
Field specifier	field-pos:field-length/F	Field specifiers for rearrangement of data fields within record. Up to 33 field specifiers can be used.
Insertion specifier	character:count/P	Inserts the *character count* times in the output record

Table 2-6 REORG Optional Arguments

Input File	Output File			
	Indexed	Line Sequential	Variable Sequential	Fixed Sequential
Line Sequential	key-specifier out-rec-size			out-rec-size
Variable Sequential	key-specifier out-rec-size			out-rec-size
Fixed Sequential	key-specifier in-rec-size	in-rec-size	in-rec-size	in-rec-size

Table 2-7 When Optional Arguments Are Required

The optional command line arguments are described in more detail below.

start-rel-key/K

The first record number in a relative output file. Subsequent records have key numbers in ascending sequence with an increment of one. When this parameter is not specified, the first record has a relative key of 1. See "Processing Relative Files" below.

in-rec-size/I

Input record size in bytes. This argument must appear only if the input file is fixed sequential.

out-rec-size/O

Output record size in bytes. This argument must be included if no field specifiers are present and the input record size is variable. Field specifiers determine the output record size. If the input record size is fixed and field specifiers are omitted, the output record size is the same as the input record size.

access-key/S

The alternate key by which the input file is to be accessed. The key is identified by its position within the record, i.e., by whether it is the first, second, third, or fourth alternate key. The form of the argument is n/S, where *n* is 1, 2, 3, or 4.

key specifier

The key position and key length arguments when the output file is indexed. If the indexed file includes alternate keys, the argument identifies the primary key. Key specifiers can be omitted when both the input and output files are indexed and data fields in the record are not rearranged.

Key specifiers have the following form:

key-pos { . : } key-length/K

Key-pos is the position of the first byte of the key in the output record. The position of the first byte of the record is byte number 1. *Key-length* is the length of the key in the output record (1 to 100 bytes). The /K switch identifies the argument as a key specifier.

Alternate keys are specified in the same form, but they are followed by the /A switch, which specifies them as alternate keys.

field specifier

The field position and field length arguments. The order and number of field specifiers and insertion characters define the output record. When field specifiers are present, the output record is built by successive moves of data fields from the input file to the output file. Up to 33 field specifiers can be included in a command. Field specifiers have the form:

field-pos { . : } field-length/F

Field-pos is the position of the first byte of a field in the input record to be moved to the output record. The position of the first byte of the record is byte 1. The destination of each move is the position following that of the field from the previous move.

Field-length is the byte length of a field defined by the preceding field position argument. The /F switch identifies the argument as a fields specifier.

insertion specifier

Insertion specifiers enable you to insert specified character strings into the output record. They have the following form:

character { . : } count/P

Character is any ASCII character (except CR, NL, space, parentheses, brackets, angle brackets, slash, and back slash) to be inserted one or more times into the output record. Lowercase characters are converted to uppercase characters. *Count* is the number of times that the specified character is to be inserted at this point in the output record.

The /P switch identifies the argument as an insertion specifier. The position of the insertion is determined by the argument's position in relation to other insertion (/P) and field (/F) specifiers in the command line.

Processing Indexed Files

With an indexed input file, REORG reads the file sequentially, based on ascending order values of the access key. The primary key is the default access key; an alternate key can be specified by including the *access-key*/S argument in the command line.

When REORG creates an indexed output file from a sequential input file or from an indexed file using an alternate key as the access key, it may encounter two or more

records with the same primary key value. Since no two output records can have the same primary key value, REORG resolves the problem by overwriting the previous record with the subsequent record having a duplicate primary key. Thus, only the final record with a duplicate primary key is retained in the output file—all others are lost. In its audit file, REORG counts the subsequent record both as a "duplicate" record and as an output record.

Alternate key values can always be duplicated in Interactive COBOL. Thus, REORG writes a new record when it encounters a duplicate alternate key value—no record is overwritten and lost.

REORG can be used to merge the records of two indexed files by specifying one of the files as the input file and the other as the output file. The record size and key lengths of the two files must agree. If REORG encounters duplicate primary key values, it overwrites records in the output file as described above.

Processing Relative Files

Relative files are like indexed files with one essential difference: the key is not part of the actual data record. Instead, the key indicates the relative record position within the file. If the /Q switch is not specified in the output file, REORG changes the keys. Therefore, if it is important to associate a given key with a record, the /Q switch should be used after the output file argument.

Recovering ISAM Files

REORG can recover ISAM files if the .NX and .XD portions are corrupt. If only one portion of the file is corrupt, you can use REORG, but first attempt recovery with REBUILD. If REBUILD is unsuccessful, use REORG to recover. Only ISAM files created under ISAM revision 5.00 or later can be recovered in this way.

REORG reads the .XD portion of the file sequentially; it does not operate on the .NX portion. If the .NX portion of the file is bad, REORG can recover from the .XD portion. If the .NX portion and the .XD header are corrupt, REORG can still recover the file from the .XD portion and additional information that you insert in the command line. However, if the data itself is corrupt, you should consider substituting a backup file.

Invalid .NX Portion

To recover a file when the .NX portion is bad, first rename *filename*.XD to a name without an extension. Then issue a REORG command line for the renamed file:

1. Specify the renamed data portion with the /D or /E switch:

   renamed-file/D (purge logically deleted records)
   renamed-file/E (retain logically deleted records)
2. Specify the position and length of the file's primary key field with the key specifier (/K) arguments.
3. To recover one or more of the file's alternate key structures, specify alternate key (/A) specifiers.

For example, to recover the indexed file CUSTFILE from its data portion, rename CUSTFILE.XD to a name without an extension, e.g., CUST$DATA. Then issue the following REORG command:

```
REORG/A CUST$DATA/D NEWCUSTFL/I 46:3/K
```

This command produces the indexed file NEWCUSTFL with the primary key in its original position (bytes 46-48); logically deleted records are purged.

REORG displays the audit information listed in Figure 2-13. Any discrepancy between the input and output record sizes is due to system overhead that is discarded when the file is built.

```
DATE:                  04/11/83
START TIME:            09:29:34
INPUT FILENAME:        CUST$DATA
        TYPE:          XD FILE
OUTPUT FILENAME:       NEWCUSTFL
         TYPE:         INDEXED
INPUT RECORD SIZE:         459
OUTPUT RECORD SIZE:        453
OUTPUT RECORD FORMAT:
    1:453   FROM       7:453
INPUT RECORD COUNT:    600
OUTPUT RECORD COUNT:   600
END TIME:              09:38:08   ELAPSED TIME:      00:01:44
```

***Figure 2-13** REORG Audit Information Screen*

Invalid .XD Header

When recovering an ISAM file, REORG takes the following information from the .XD file header: the number of alternate keys, the record length, and the next record to append. However, if the .XD header is invalid, you can supply this information in the command line instead.

To recover an ISAM file when the .XD header or both portions are invalid, rename *filename*.XD to a name without an extension. This renamed file is used as the input file.

The form of the command line is as follows:

REORG [{ /A=auditfile / /A }] in-file { /D / /E } /X out-file [{ /I / /Q / /R }]
key-specifiers num-alt-keys/N rec-length/L [num-recs/R]

/A A global switch that directs REORG's audit information to the screen or to the specified file.

in-file
The name of the renamed .XD portion. Both the /D and the /E switches indicate that the input file is a renamed .XD file. Use the /D switch to purge logically deleted records and the /E switch to retain logically deleted records.

/X A switch that tells REORG that the .XD header information is to be taken from the command line instead of the .XD header.

out-file
The name of the output file. It may be indexed (/I), relative (/R), or relative without renumbering the records (/Q). The default is indexed.

key specifier
The position and length of a key, in the form *key-pos:key-length*/K. Specify alternate keys in the same form but with the /A switch.

num-alt-keys/N
The number of alternate keys in the file. Follow this number with the /N switch. If the file has no alternate keys, enter 0/N.

rec-length/L

The length of the data portion of the .XD record, in bytes. This number should not include the .XD record header. The record length is the total number of characters in the PICs for that record, as defined in the original COBOL program. The record length is followed by the /L switch.

num-recs/R

The number of records in the uncorrupt version of the file. The .XD header contains a pointer to the next record to append. REORG ordinarily calculates the number of records to read from this pointer; however, REORG cannot do this because the .XD header is invalid. Enter the number of records if possible. If you enter the number of records and REORG reaches the logical end of file first, REORG stops at the logical end of file. If you do not enter this argument, REORG proceeds until it reaches the logical end of file.

Note: When recovering a file with an invalid .XD header, be sure to specify the record length and the number of alternate keys that are in the corrupted file. If you change the record length or number of keys, REORG may operate, but it is not able to determine the correct record boundaries. Therefore, the data you recover will be corrupt.

Relative File Recovery

Assume you have determined that the index to a relative file is bad. You want to recover the file, but you do not want the relative keys to be renumbered. Rename the data portion (as explained in "Recovering ISAM Files" above) in this case, CUSTOMERS1. Then enter the following command line:

```
REORG/A=@LPT CUSTOMERS1/D CUSTOMER00/Q
```

Audit information is printed at the system printer. The /D switch indicates that logically deleted records should not be included as the new index is built. The /Q switch preserves the original relative key number.

Converting File Types and Deleting Records

REORG allows you to convert among any of the Interactive COBOL file types. When you convert from an ISAM file to an ISAM file, or from an ISAM file to a sequential file, logically deleted records are not written to the output file.

Indexed to Indexed

The following command copies the indexed file 1981ITEMS to the indexed file CLEANUP, retaining the same record size and key:

```
REORG/A=@LPT 1981ITEMS CLEANUP
```

All logically deleted records of 1981ITEMS are physically deleted in CLEANUP. Audit information is output to the printer.

Indexed to Sequential

Assume you want to transmit the indexed transaction file SALESLOG from a remote site to a central computer system that cannot read Interactive COBOL indexed files. The file must be converted from indexed organization to fixed sequential. To convert the file, issue this command:

```
REORG/A=SALESAUDIT SALESLOG SENDSALES/S
```

The indexed file SALESLOG is copied to the fixed sequential file SENDSALES. At the same time, logically deleted records are physically deleted; this prevents unprocessable records from being transmitted. Audit information is written to the file SALESAUDIT. The file SENDSALES may now be transmitted.

Printing an ISAM FIle

An ISAM file can be printed by using REORG to create a line sequential version of the file. The following command lists the indexed file 1981ITEMS to the system printer:

```
REORG/A 1981ITEMS @LPT/L 11:20/F 38:6/F
```

Output file @LPT is specified as line sequential (/L). The fields printed, specified by the /F switch, are bytes 11-30 and 38-43 from each record. Audit information is displayed on the terminal.

To obtain a permanent copy of 1981ITEMS to print, simply specify an output filename rather than @LPT with the /L switch.

Relative to Relative

You can eliminate logically deleted records from a relative file by REORGing it from relative to relative. This conserves disk storage space and increases the number of available records in the file. To REORG the relative file CUSTOMERS1, enter:

```
REORG/A=@LPT CUSTOMERS1/R CUSTOMER00/R
```

The audit file is printed on the system printer.

In the above command line, the logical (or key) order of the file is maintained, but the first record now has the relative key 1. For example, CUSTOMERS1 contains three records. If the record referred to by key 9 is logically deleted, CUSTOMER00 contains two records after REORG has processed it.

Before REORG		After REORG	
Key	**Data**	**Key**	**Data**
8	C	1	C
9	B		
10	A	2	A

If you do not want to renumber the records in this way, use the *start-rel-key*/K argument. This argument allows you to specify the starting number of the first relative record. However, the rest of the key numbers are still incremented by 1.

To preserve the relative record numbers, follow the output filename with the /Q switch rather than /R.

```
REORG/A=@LPT CUSTOMERS1/R CUSTOMER00/Q
```

After using REORG on the file, you should delete the old version of the file and rename the new version. The following CLI command removes the old version of the file from the system:

```
DELETE CUSTOMERS1.(NX XD)
```

This command gives the new version of the file the proper name:

```
RENAME CUSTOMER00.(NX XD) CUSTOMERS1.(NX XD)
```

Sequential to Indexed with Alternate Keys

A data base includes the sequential file ORDERS, with records of the following format:

Field	Position
Order number	Bytes 1-6
Customer name	Bytes 7-26
Customer number	Bytes 27-30
Customer region	Bytes 31-35
Customer address	Bytes 36-75

To create an indexed file CUSTOMERS that contains the same data fields, has the customer number as the primary key, and the order number and customer region as alternate keys, enter:

```
REORG/A=@LPT ORDERS/S CUSTOMERS 75/I 27:4/K 1:6/A 31:5/A
```

The input record size is required because the input file is fixed sequential. The argument 27:4/K specifies customer number as the primary key; 1:6/A and 31:5/A specify order number and customer region, respectively, as alternate keys.

Changing Access Keys

REORG allows you to change or to add keys to an ISAM file. For example, the indexed file INVENTORY has stock number as the primary key, and the record layout is as follows:

Field	Function	Position
Stock number	Primary key	Bytes 1-5
Stock item	Data field	Bytes 10-29
Unit cost	Data field	Bytes 30-39
Unit price	Data field	Bytes 40-49
Number on hand	Data field	Bytes 50-60

Assume you want to add stock item as an alternate key and to print the file ordered by the number-on-hand field so you can determine which items to reorder. To do this, enter the following command line:

```
REORG/A INVENTORY INVENTORY_OUT 1:5/K 10:20/A 50:11/A
```

This creates the file INVENTORY_OUT, which has two alternate keys (specified by the local /A switches) in addition to the primary key. Audit information is displayed.

To print the file ordered by the second alternate key, enter the following command:

```
REORG/A INVENTORY_OUT INVENTORY_PRINT/L 2/S
```

This command creates the line sequential file INVENTORY_OUT, which can be printed on the system printer. The access key is the second alternate key, as specified by the argument 2/S. Audit information is displayed.

Rearranging Data Fields

A data base includes the indexed file PERSONNEL, with records of the following format:

Field	Function	Position
Social Security number	Primary key	Bytes 1-9
Employee name	Alternate key	Bytes 10-29
Employee address	Data field	Bytes 30-69
Department number	Alternate key	Bytes 70-75

REORG is to create the fixed-length sequential file EMPNAMES with records containing employee names and addresses, and ordered by department number. The command line is:

```
REORG/A PERSONNEL EMPNAMES/S 2/S 10:20/F 30:40/F
```

The access key argument 2/S specifies the access key to be the second alternate key, which is the department number. Note that the access key does not have to be included in the output file.

Inserting Editing Characters

The following command copies the indexed file 1981ITEMS to the sequential file DESC:

```
REORG/A=@LPT 1981ITEMS DESC/S 11:20/F *:3/P 1:5/F
```

The 20-byte field at position 11 of the input record becomes bytes 1 to 20 of the output record. An asterisk is inserted in bytes 21, 22, and 23 of the output record. The 5-byte field at position 1 of the input record becomes bytes 24 to 28 of the output record. The output record size is therefore 28 bytes. Audit information is output to the line printer.

Error Messages

ALTERNATE KEY FALLS OUTSIDE OF OUTPUT RECORD

The byte position specified for the alternate key cannot exceed the output record size, and the byte length cannot run past the end of the record.

BAD REVISION

The major revision number of the input file, which is stored in block 0 of the .NX and .XD portions, is less than 5.00.

CHARACTER INSERTION OVERFLOW (> 33)

You can specify insertion characters up to 33 times in one command line.

CONSOLE INTERRUPT

You have entered the control interrupt characters CTRL-C CTRL-A.

DIR. ANNEX BLOCK ERROR

ISAM error.

DUPLICATE SPECIFICATION

You have specified more than one access key (/S). Only one is permitted.

END OF FILE ERROR

A premature end of file was reached. Run ISAMVERIFY or use a backup file. You may have a hardware problem.

FIELD FALLS OUTSIDE OF INPUT RECORD

The field position or length (/F) cannot exceed the record size and cannot run past the end of the input record.

FIELD FALLS OUTSIDE OF OUTPUT RECORD

The field position or length (/F) cannot exceed the record size and cannot run past the end of the output record.

FIELD SPECIFIER OVERFLOW (> 33)

You cannot specify more than 33 field specifiers.

INDEX DEPTH EXCEEDED

ISAM error.

KEY NOT FOUND

ISAM error.

KEY SIZE ERROR

ISAM error.

IF /X, MUST GIVE /L AND /N

If you use the /X switch, indicating that the .XD header is bad, you must supply the number of alternate keys and the record length, using the /L and /N switches, respectively.

ILLEGAL ACCESS KEY

You have specified an access key that is greater than 4 (4/S).

ILLEGAL ACCESS KEY FOR RELATIVE FILE

You cannot specify an access key for a relative file.

ILLEGAL FILE TYPE

You specified a file type switch that is not permitted. Legal file type switches are /I, /L, /R, /S, /V, /Q (output only), and /D and /E (input only).

ILLEGAL INTEGER

You may have specified nonnumeric character where a numeric character is required.

ILLEGAL LOCAL SWITCH

You have specified an incorrect local switch.

INPUT FILE DOES NOT EXIST

The input file may be sequential and you have specified it as indexed. If this is the case, reenter the command line, using the /V, /L, or /S switch after the file type. Other possibilities are that the file is not in your directory or searchlist, or you are attempting to REORG a directory.

INPUT FILE NOT FOUND

You have tried to REORG an indexed file and have specified it as sequential.

INPUT FILE NOT INDEXED

You may have specified a key for a sequential input file.

INPUT FILE TYPE MUST BE /D, /E, OR /R

If the /Q switch is specified with an output file, the input file must have the /D, /E, or /R switch.

INPUT RECORD SIZE NOT SPECIFIED

If your input file type is fixed sequential (/S) you must specify the input record size.

INPUT RECORD SIZE SPECIFIED AS 0

You cannot specify a record size as 0.

KEY LENGTH INCOMPATIBLE WITH EXISTING FILE

The output key length for a relative file must equal 2.

KEY SPECIFIED FOR NON-INDEXED OUTPUT FILE

You cannot specify a key when the output file is sequential.

MLS IS NOT ACTIVE

The ISAM server process is not active.

MOVES SPECIFIED

You cannot move the data in a file when recovering an ISAM file.

NO INPUT FILE SPECIFIED

You have not specified any files.

NO KEY SPECIFIED

You must specify a key when producing an indexed output file. Either you have not specified the key or you have an incorrect switch for the output file type.

NO OUTPUT FILE SPECIFIED

You have specified only one file; you must specify an input and an output file.

OUTPUT FILE IS NOT RELATIVE

You have attempted to use an existing output file for REORG and have specified it as relative when it is not.

OUTPUT FILE MUST BE ISAM

When you are recovering an ISAM file, your output file must be ISAM and must be followed by the /I or /R switches.

OUTPUT LENGTH AND MOVE FIELDS SPECIFIED

You cannot specify output fields (/F switch) and output length (/O switch) on the same command line.

OUTPUT RECORD SIZE NOT SPECIFIED

You must specify the output record size when the input file is line sequential, when converting from variable sequential to fixed sequential, and when converting from variable to indexed.

OUTPUT RECORD SIZE SPECIFIED AS 0

You cannot specify an output record size as 0.

PRIMARY KEY FALLS OUTSIDE OF OUTPUT RECORD

The primary key specified is beyond the length of the output record. For example, you may have a record length of 50 bytes and specified 51:2/K as the key specifier.

RECORD ALREADY EXISTS

ISAM error.

RECORD SIZE > 132 CHARACTERS

The maximum record size in a line sequential file may not exceed 132 characters (plus the terminator).

RECORD SIZE > 4096 CHARACTERS

The maximum record size in a fixed or variable sequential file is 4096 characters.

RECORD SIZE ERROR

ISAM error.

RECORD SIZE INCOMPATIBLE WITH EXISTING FILE

REORG has determined that the record length in the existing file is not the same as the specified record size.

RECORD SIZE INCOMPATIBLE WITH FIELD SPECIFIERS

The field specifiers are too large to be accommodated by the record size.

RELATIVE KEY OVERFLOW

The relative key is larger than 65,534. You may have attempted to REORG a very large sequential file to relative format.

TOO MANY ALTERNATE KEYS SPECIFIED

Up to four alternate keys can be specified.

SETFORMS is a menu-oriented utility for those who use the Data Royal printer, model IPS-5000-A, with option 190168. It allows you to set variable forms length and vertical tabbing positions for special forms. SETFORMS programs the printer for vertical form length and tabbing. You do not have to add code to your Interactive COBOL program.

Procedure

Before invoking SETFORMS, set the serial interface switches on the printer (back panel) as follows:

SW1	*	9600 baud
SW2	*	4800 baud
SW3	*	2400 baud
SW4	*	1200 baud
SW5	*	300 baud
SW6	*	110 baud
SW7	OFF	2 stop bits
SW8	ON	7 data bits
SW9	ON	parity present
SW10	OFF	Even parity

*The first six switches control baud rate.
Only one of the six can be turned ON.

Set the system configuration switches (front panel) as follows:

SW1	OFF	80 character/line
SW2	OFF	not used
SW3	ON	no LF after CR code
SW4	ON	serial interface
SW5	OFF	ignore CR code
SW6	OFF	no perforation skipover
SW7	OFF	not used
SW8	OFF	not used

To invoke SETFORMS through Logon, select (C) and enter SETFORMS at the RUN PROGRAM prompt. To invoke SETFORMS through the CLI, enter SETFORMS at the CLI.

SETFORMS displays a screen with the procedure for preparing the printer for programming. The procedure is as follows:

1. Set the printer off-line.
2. Remove the standard paper and load the forms to be used.
3. Align the paper to the first line on the form.
4. Press the top-of-form switch.
5. Set the printer on-line.

The printer is now ready to be programmed. Press ESC to exit SETFORMS; press any other letter or number key to display the next screen.

The next screen prompts you for programming information. Enter the printer device name. This is the filename in your Interactive COBOL program that denotes the printer, i.e., @LPT or a filename that is linked to @LPT. SETFORMS also prompts for the number of lines per inch (6 or 8), the number of inches per form (1 to 24), and whether or not vertical tabbing is required. You may exit from the program by pressing ESC after the **PRINTER DEVICE FILENAME** prompt.

If you require vertical tabbing, SETFORMS asks for the number of vertical tabs. SETFORMS also displays the maximum number of tabs that you can enter. For example, a two-inch form with six lines per inch cannot have more than 12 tabs. The maximum number of tabs is 16, regardless of the lines per inch and size of the form.

After you enter the number of tabs, SETFORMS prompts for the starting line number for each vertical tab. For example, to tab to line 4, then to line 10, enter 2 for the number of vertical tabs. The prompts **TAB 1** and **TAB 2** are displayed. Enter 4 and 10, respectively. Press ESC to exit from SETFORMS.

After you have entered the required information, SETFORMS displays the prompt **ANY CHANGE**. This enables you to change any of the answers you have given.

- If you enter **Y**, SETFORMS prompts you for the response to be changed. If you change the number of lines per inch or inches per form, SETFORMS recalculates and redisplays the maximum number of vertical tabs. You must then enter the number of vertical tabs and the line number for each tab. You may change your responses by entering Y in response to the the **ANY CHANGE** prompt.
- If you enter **N**, the utility begins to program the printer.
- If you press the ESC key, you exit from SETFORMS and return to the CLI.

SETFORMS displays a message that tells you the printer is being programmed. This programming takes a few minutes; SETFORMS informs you when it has finished. You may now print the special forms via your COBOL program. Figure 2-14 is an example of a COBOL program that you would invoke after running SETFORMS.

```
SELECT PRINTFILE.
ASSIGN TO PRINTER, PRINTFILE-NAME.
FILE STATUS IS...
    ...
FD  PRINTFILE
    ...
01  PRINT-RECORD           PIC X(80).

01  TAB-RECORD.
    05  TAB-CODE           PIC 99 USAGE COMP.
    05  FILLER             PIC 9(18) USAGE COMP.
    ...
WORKING-STORAGE SECTION.
01  VERTICAL-TAB           PIC 99 VALUE 11 USAGE COMP.
    ...
PROCEDURE DIVISION.
    OPEN EXCLUSIVE OUTPUT PRINTFILE.
    WRITE-LINES.
    ...
    MOVE LOW-VALUES TO TAB-RECORD.
    MOVE VERTICAL-TAB TO TAB-RECORD.
    WRITE TAB-RECORD AFTER ADVANCING 0 LINES.
```

Figure 2-14 *Sample Program for Printing to a Special Form*

Once the forms have been printed, you may reload standard paper or run SETFORMS again to print other special forms. In either case, follow this procedure:

1. Press the RESET switch.
2. Reload the printer with the new forms or standard paper.
3. Align the paper to the first line on the form.
4. Press the top-of-form switch twice. (The second time lets you check paper position.)
5. Set the printer on-line.
6. Enter the command line to run SETFORMS again.

Error Messages

`EXCEEDS MINIMUM/MAXIMUM TABS`

Your response to `NUMBER OF TABS` is less than 1 or greater than the maximum: 6, 8, 12, or 16, depending on the length of the form and the number of lines per inch.

`MUST BE 6 OR 8`

Your response to the `LINES/INCH` prompt must be 6 or 8.

`MUST BE BETWEEN 1 AND 24`

You entered a value for `INCHES/FORM` that was out of range.

`MUST BE 'Y' OR 'N'`

Your response to the `TABBING REQUIRED` prompt must be `Y` or `N`.

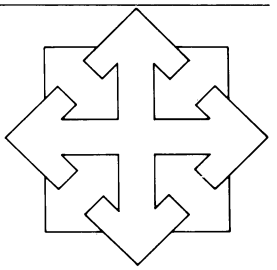
`PRINTER ACCESS FILE ERROR: nn`

An error has occurred on printer access. The runtime system displays a two-digit File Status code after the message. See the *Interactive COBOL User's Guide (AOS, AOS/VS)* for an explanation of the File Status codes.

`TAB POSITION OUT OF RANGE`

You entered a tab position that was 0, greater than the number of lines on the form, or less than or equal to the previously entered tab position.

Appendix A
Switch Combinations for REORG



The matrix below indicates the legal combinations of local switches that may be specified in a REORG command line. The input and output file requirements are designated by their respective switches:

/S Fixed sequential file type
/V Variable sequential file type
/R Relative file type
/I Indexed file type
/L Line sequential file type
/Q Relative output file, without renumbering records
/D Delete logically deleted records
/E Retain logically deleted records
/X File header information in the command line

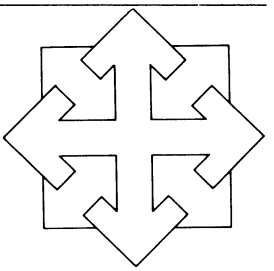
All switches shown are local switches. The switches in the boxes within brackets are optional. Switches outside brackets are required to create the output file from the given input file. If a box is empty or there are no switches outside brackets, no switches are required. X means the input file cannot be reorganized to the given output file type.

For definitions of switches and a complete discussion of reorganizing files, see REORG in chapter 2.

OUTPUT FILES

INPUT FILES	/S	/V	/R	/I	/L	/Q
/S	/I [/O/F/P]	/I [/O/F/P]	/I [/K/O/F]	/I /K [/A/O/F]	/I [/O/F/P]	X
/V	/O [/F/P]	[/O/F/P]	/O [/K/F]	/O /K [/A/F]	[/O/F/P]	X
/R	[O/F/P]	[/O/F/P]	[/K/O/F]	/K [/A/O/F]	[/O/F/P]	
/I	[/S/O/F/P]	[/S/O/F/P]	[/S/K/O/F]	[/A/S/O]	[/S/O/F/P]	X
/L	/O [/F/P]	[/O/F/P]	/O [/K]	/O /K [/A]	[/O/F/P]	X
/D	X	X	[/K]	/K [/A]	X	
/D/X	X	X	/N /L [/K/R]	/N /L /K [/A/R]	X	/N /L
/E	X	X	[/K]	/K [/A]	X	
/E/X	X	X	/N /L [/K/R]	/N /L /K [/A/R]	X	/N /L

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, and a set of COBOL program examples is provided. A syntax summary section provides a quick reference.

Interactive COBOL User's Guide (AOS and AOS/VS) **069-705015**
Supplies the programmer with information relating specifically to Interactive COBOL on the AOS and AOS/VS operating systems. The document describes the file system, system calls, the runtime system, the compiler, and the debugger. Lists of error messages and their meanings are provided.

IC/EDIT: 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 IC/EDIT 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 IC/SCREEN and CLI/SCREEN programs, which are special purpose editors 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.

AOS Documents

Learning to Use Your Advanced Operating System (AOS) **069-000018**
Summarizes AOS system utilities and leads the reader through practice sessions. It explains the steps required to run FORTRAN, COBOL, and assembly language programs.

Introduction to the Advanced Operating System (AOS) **069-000016**
Serves as a technical introduction to AOS, highlighting the features and facilities of this powerful operating system.

Command Line Interpreter User's Manual (AOS and AOS/VS) **093-000122**
Describes the command line interpreter (CLI), which is the primary interface to the system for the system manager, operator, and most users. The manual explains how users can run utilities, execute programs, maintain files, and build command "macros" by means of the CLI.

AOS Operator's Guide **093-000194**
Describes how to start up, run, and shut down an AOS system; how to operate the EXEC, batch, spooler, and other useful utilities; and what to do in the event of hardware and software errors or a system failure.

Managing AOS **093-000217**
Explains how to create an AOS system tailored to the requirements of a specific installation and how to manage the system. It describes techniques for creating optimal performance and security. It presents the salient features of system utilities and parameters, the AOS file access protection scheme, and process management considerations. It also explains the PREDITOR program, which creates user profiles.

AOS Programmer's Manual **093-000120**
Serves as a primary reference for assembly language programmers. It describes in detail the external features of AOS, including the AOS system calls and information required for an in-depth understanding of the operating system. Other topics included are process concepts, memory management, file I/O, and multitasking.

AOS Console User's Handbook **093-000150**
Contains a digest of useful information that is presented in more detail in other AOS manuals. Its compact size and concise format make it a convenient reference document.

AOS Link User's Manual **093-000254**
Describes in detail the LINK utility, which binds assembled or compiled modules into a memory context, resulting in a file of binary information the computer can execute. The manual describes the AOS memory scheme, command line syntax, and object block structure. Appendices explain symbol types, the Resource Handler Table, linking AOS object modules for execution under RDOS, and LINK error messages.

AOS/VS Documents

Learning to Use Your AOS/VS System **069-000031**
Serves as a basic introduction to the Advanced Operating System/Virtual Storage for programmers and nonprogrammers. It summarizes the system utilities and gives an overview of AOS/VS products. It also leads the reader through practice sessions with editors and illustrates the creation and execution of programs written in FORTRAN, COBOL, BASIC, and assembly language.

Command Line Interpreter User's Manual **093-000122**
(AOS and AOS/VS)
Describes the command line interpreter (CLI), which is the primary interface to AOS/VS for the system manager, operator, and most users. The manual explains how users can run utilities, execute programs, maintain files, and build command "macros" by means of the CLI.

AOS/VS Operator's Guide **093-000244**
Describes how to start up, run, and shut down an AOS/VS system. It contains instructions for formating disks, installing and maintaining a system, and installing new systems. The manual is a reference tool, not a step-by-step guide for performing all related procedures.

Managing AOS/VS **093-000243**
Explains how to build an AOS/VS system tailored to a user's hardware and software and how to manage the system. It provides information for managing memory and CPU resources for optimal efficiency and secure software.

AOS/VS Programmer's Manual **093-000241**
Serves as a primary reference for assembly language programmers. It describes the external features of AOS/VS, including system calls and information required for understanding the operating system. Other topics included are virtual memory concepts, interprocess communication, file structure and maintenance, file I/O, multitasking, and binary synchronous communication.

AOS/VS Link and File Editor User's Manual **093-000245**

Describes Link and Library File Editor (LFE), which play related roles in program development. It serves users who want to link or relink programs assembled and compiled under AOS/VS, and users who want to assemble and compile programs under their own language processors. It describes how to use LFE to create, edit, and analyze library files.

Index

P

R

S

V

X

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

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

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

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

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

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

☐ Purchase Order Number:_____

SHIP VIA

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

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

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

Person to contact about this order _____ Phone _____ Extension _____

Data General Corporation
Educational Services/TIPS
MS G214
2400 Computer Drive
Westboro, MA 01580
(617) 366-8911, Extension 1610

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:

   Data General Corporation
   Educational Services/TIPS
   MS G214
   2400 Computer Drive
   Westboro, MA 01580
   (617) 366-8911, Extension 1610
8. We'll take care of the rest!

educational services

Data General Corporation, Westboro, MA 01580

069-705021-01