Data General

Interactive COBOL Utilities (RDOS)

Interactive COBOL Utilities (RDOS)

DataGeneral

069-705020-01

DATA GENERAL CORPORATION, Westboro, Massachusetts 01580

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS DOCUMENT OR THE INFORMATION CONTAINED IN IT, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

Ordering Number 069-705020
Revision 01, June 1983
(Interactive COBOL, Rev. 1.10)

Original release: July 1982 (Interactive COBOL, Rev.1.00)
Original title: *Interactive COBOL Utilities (RDOS and AOS)*

Changes to Interactive COBOL

For RDOS 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.

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, DEFLINES, enables the user to set line speeds for QTY lines and to restrict access to these lines without bringing down the runtime system.

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

Interactive COBOL Documents
RDOS 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 utilties 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 (RDOS)*. 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 Real Time Disk Operating System (RDOS) should first consult the documentation related to this system (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, DEFLINES, DO, FILESTATS, INQUIRE, ISAMVERIFY, LJE, MESSAGES, NOTES, SETFORMS, REBUILD, and REORG. Numerous examples supplement this material. An appendix summarizes the legal switch combinations for the REORG utility. An index provides convenient access to topics.

Notational Conventions

UPPERCASE	Commands and switches appear in uppercase.
lowercase	User-supplied arguments, such as filenames, appear in lowercase.
[ ]	Optional arguments; 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.

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. 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 a file.

COLLAPSE

COLLAPSE enables you to improve indexed file access and 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 on the changes made to a file.

CSSORT

CSSORT performs off-line sorting and merging of files. CSSORT accepts any file created by Interactive COBOL as input and produces a sequential file as output. CSSORT sorts 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 accepts 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 for an indexed file, you may decide to use COLLAPSE to adjust the density of the index structure of such a file.

ISAMVERIFY

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

REBUILD

REBUILD reconstructs corrupted ISAM files to the point where they may be used by other utilities and programs without "hanging" or terminating the system. REBUILD works only on indexed and relative files. It can rebuild either the .NX or .XD portion if the other portion has passed ISAMVERIFY's tests.

REORG

REORG converts files from one type to another. It 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. With the associated utilities VINIT and VSTAT, REORG can prepare magnetic tapes for export and process tapes from other installations.

REORG also reconstructs ISAM files when both the .NX and .XD portions have been corrupted.

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.

Setting Line Parameters: DEFLINES

DEFLINES is an Interactive COBOL utility that allows you to set line parameters dynamically. With this utility, you do not have to bring down the runtime system to define line characteristics for QTY lines. Each line can be set to a specific speed.

CLI Command File Utility: DO

DO executes a macro file containing a series of CLI commands, containing up to 512 dummy arguments. When you invoke DO, arguments in the command line are substituted in the file's CLI commands before they are executed by the CLI.

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.*

Formatting Screens: ICSCREEN

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

Processing Data Files: INQUIRE

INQUIRE is used to review, update, or create data files. INQUIRE 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. INQUIRE operations include forward and backward sequential reading, data record display, and updating and deleting records.

Processing Batches: LJE

The Local Job Entry monitor (LJE) executes files containing CLI commands. In concurrent systems, LJE effectively gives users in the runtime environment access to the CLI. In systems without concurrency, LJE serves as a batch processor. Jobs that are submitted during runtime system operation are executed by LJE when the runtime system is stopped.

Editing Error Messages: MESSAGES

MESSAGES modifies the contents of the runtime system's error message file. Messages from this file are displayed on the terminals when operator errors (e.g., entry of illegal characters or omission of required data) or runtime errors occur.

Communicating Interactively: NOTES

NOTES provides an interactive facility for messages, announcements, or interoffice communication. NOTES stores the user's name with each note and associates a password with each name. A note may be changed only by its originator. However, when someone writes a response, the note cannot be changed.

Setting Vertical Forms: SETFORMS

This utility is for those who have 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 printing forms.

Executing the Utilities

An Interactive COBOL utility executes either under the runtime system or under the command line interpreter (CLI).

CLI Utilities

The description of each utility in chapter 2 indicates if the utility can be 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 description of each utility in chapter 2 indicates if the utility can be executed from the runtime system. To execute the runtime utilities, first start the runtime system by entering the following at the CLI prompt:

```
ICX[/I/B] [terminals[/T]] [size/S] [files/F]
```

The /I switch enables operators to interrupt COBOL programs at all terminals other than the master terminal. The /B switch enables master terminal operators to interrupt execution of the runtime system. *Terminals* is the maximum number of terminals that will run COBOL programs; the default is 1. *Size* is the maximum COBOL program size, which is an odd number of kilobytes from 3 to 31; the default is 27 kilobytes. *Files* is the maximum number of COBOL ISAM files to be open, which can be from 4 to 64. If you do not enter this argument, the default is (3 * *terminals*) + 13. See the *Interactive COBOL User's Guide (RDOS)* for detailed information on starting the runtime system.

Logon Menu

The Logon Menu is displayed when the runtime system is activated. Figure 1-1 shows the Logon Menu that appears on the master terminal. A similar Logon Menu appears on the other terminals but does not include the PASS, STOP, or COBOL program selections. However, it does include "Hangup," which allows the user to deactivate a terminal without affecting other terminals.

```
Interactive Cobol - Monitor Revision 1.10
          COBOL Development Logon

           DATE:   4/11/83
           TIME:   12:34:56
           Terminal Number: 0




           Utility:(?)

(P).A.S.S. Control          (C)obol Program
(M)essage                   (I)CEDIT Editor
(A)bort                     (O)ptional Utilities
(T)erminal Status           (D)ebug a Program
(S)top                      (J)ob Entry
```

Figure 1-1 *Logon Menu — Master Terminal*

The actions performed by the Logon options include the following:

P Invokes the printer access scheduling system. Spooled output is selected for printing with PASS, which is documented in chapter 3 of the *Interactive COBOL User's Guide (RDOS).*

M Is equivalent to the #M system call. It enables you to send a message to all terminals logged onto the runtime system.

A Is equivalent to the #A system call. It allows you to abort a COBOL program running at any other terminal on the same ground.

T Is equivalent to the #T system call. It lists the status of all terminals logged onto the system, in split-screen format. Press ESC to return to the Logon Menu or CR to redisplay the screen with updated information.

S Is equivalent to the #S system call. It returns you to the CLI prompt. Before you enter S, make sure that all other terminals are inactive.

C Allows you to run a COBOL program from the Logon Menu. You are prompted for the program name and any additional required information. When the utility has finished executing, you are returned to the Logon menu.

You may also run user-created Interactive COBOL programs from the Logon Menu, if the programs have been compiled without errors and are within, or linked to, the working directory. If the program uses switches, enter them along with the name of the program when prompted from Logon.

I Invokes the ICEDIT utility, a text editor created especially for writing COBOL programs. For more information on this editor, consult *ICEDIT: Interactive COBOL Editor.*

O Displays the Optional Utilities Menu, shown in Figure 1-2. The options Help and Type are functions of ICEDIT. The remaining options are utilities that are described above and in chapter 2. (E invokes CALC.)

D Is an interactive facility for program debugging. Any program to be debugged must have been compiled with the command ICOBOL/D. The program must also be in, or linked to, the current directory. The debug function is explained further in Chapter 5 of the *Interactive COBOL User's Guide (RDOS).*

J Places a job in the LJE execution queue. See chapter 2 for an explanation of the LJE utility.

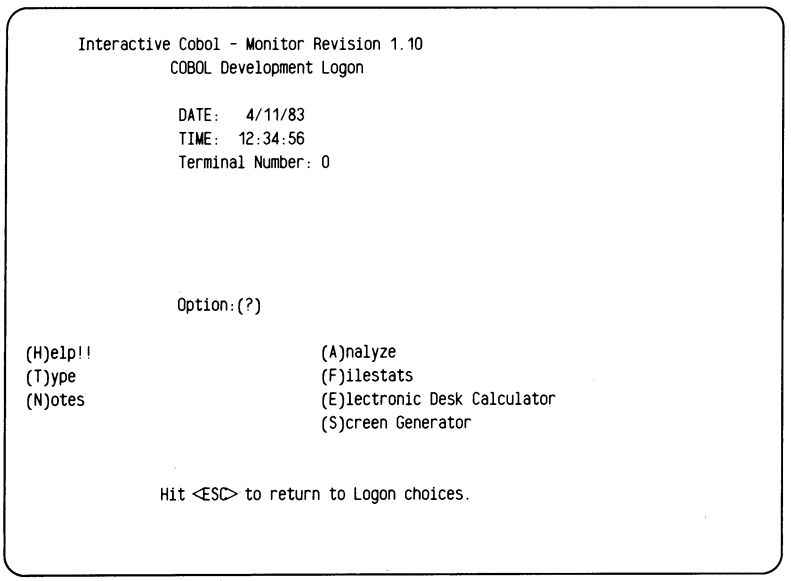


***Figure 1-2** Optional Utilities Menu*

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 (.XD) portion of each ISAM file contains two reliability flags — one for the index (.NX) portion 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 an unusual hardware event such as a power failure occurs while a file is open, the reliability flags remain set. Since the flags are set, the files cannot be accessed by the utilities COLLAPSE, CSSORT, and ICEDIT, nor by certain functions of REORG and ANALYZE, nor by the runtime system. If you attempt to access a file flagged as corrupt, a message is displayed that warns 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

RDOS uses a system of links for accessing files. A *link* is a filename in one directory that points to a file (called the *resolution file*) in the same directory, another directory or to another link. Links save disk file space by allowing users in different directories to access a single commonly used disk file. With a link, you can access any disk file as if it were in your own directory (assuming that the file does not have the N attribute, which forbids linking). Create a link with the LINK command:

```
LINK link-name resolution-filename
```

RDOS creates the link in the current directory, unless you specify another directory. The directory that the file resides in must be initialized before you can use a linked file. For more information on links, see *RDOS/DOS Command Line Interpreter*.

Chapter 2
Utility Reference

This chapter describes the functions of the Interactive COBOL utilities on RDOS 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 statistics about files that help the programmer evaluate file structures in order to achieve optimum storage and retrieval. ANALYZE checks file integrity before it begins operating on a file. Use the utility frequently on contiguous files to see if the file is becoming full. When it is full, the INDEX FULL message appears.

ANALYZE runs under the control of the Interactive COBOL runtime system. The program is compatible with all other runtime system uses. Therefore, it can be executed at any terminal in an Interactive COBOL system and at several terminals at the same time. It is easy to run and 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 the 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 key within the data record, number of keys per index block, number of index blocks, and structure tables (optional)

For each record key the index structure tables show (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, select the Optional Utilities from the Logon Menu. Type A at the Optional Utilities Menu. 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. If the printer is busy, a file named ANALYZE*xx*.LP (where *xx* is the console or line number) is created and can be printed at a later time. Delete files that have been printed in this way, or future analyses will be appended to the existing file. Enter D to display the results on the screen. After each analysis is displayed, 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. The utility then prompts for a file to be analyzed:

```
FILENAME :
```

Enter the name of a file to be analyzed. Filenames should be entered without the .NX or .XD extensions. The name may include a directory specifier of up to 4 characters, and the name of an ISAM file up to 10 characters. ANALYZE continues to accept filenames until you press CR alone or until you enter 15 filenames. 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 levels 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, REORG, or use a backup file.

Example

In the following example, ANALYZE reports statistics for the indexed file PARTFILE. 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 is not corrupt; that is, PARTFILE's reliability flags are not set, and it meets ANALYZE's boundary condition checks. PARTFILE was designed 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 LIST 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 or because fewer index levels are 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 may be corrupt and has set the reliability flags. Run ISAMVERIFY, which will provide the information necessary to run REBUILD or REORG, or will clear the reliability flags.

```
FILE NOT FOUND OR ALREADY OPENED
```

The specified file does not exist, is already open, or, if the file is a link file, its directory is not initialized. 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 determined that one or both reliability flags are set in the file to be analyzed. Run the utility ISAMVERIFY on the file, which will provide the information for you to reconstruct the file with REBUILD or REORG, or will clear the reliability flags.

The program CALC is a general purpose calculating aid. It features ten variables that may be assigned values by the user, display of the five most recent results, arithmetic operators and functions, entry of algebraic expressions, and user-defined functions.

Procedure

To invoke CALC, choose the Optional Utilities from the Logon Menu. Then choose Electronic Desk Calculator (E). The screen that appears is shown in Figure 2-2.

```
Date:  4/11/83      CALC Revision 1.10              ^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
<	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 any 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 the 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$library-name.NX
DELETE[/V] CALC$library-name.XD
```

The *master library* may not be modified. This library contains the following functions, which are supplied along 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 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 the functions in the master library cannot be altered, 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. 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 places 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 tells where the definition is located and displays it on the entry line.

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, enter CALCLIB in response to the **RUN PROGRAM** prompt on the Logon Menu. 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** CALCLIB 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 the function, and displays a confirmation message. To delete another function, enter its name.

Copy library to printer. CALCLIB uses the PASS queue to copy the contents of the library to the printer. The PASS queue may be accessed from the Logon program at the master console.

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 printfile is CALC followed by the time, a period, and the two-digit line number of the user's terminal. For example, if the contents of the library are printed at 9:25:07 A.M. from terminal 3, the name of the printfile is CALC092507.03.

Error Messages

EXPRESSION TOO COMPLICATED

While interpreting the expression, CALC ran out of temporary storage. This is usually 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 the expression, CALC ran out of room to store intermediate values. Split the expression into smaller segments.

ERROR IN EXPRESSION SYNTAX

While scanning the 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 have a $ at the end of the line, more than one decimal point, or 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 is not in the current directory or is not linked to the current directory.

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 or is not linked to the current directory.

NO SUCH LIBRARY

You tried to access a library that does not exist 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 file. 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 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 file 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 indexed files are mainly written to, high-density index packing slows down write time. In this situation, it may be desirable to maintain a low packing density. A low packing density, however, 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, physically deleting logically deleted records and packing index structures up to 99% full. When the files are retrieved for rewriting, COLLAPSE may be rerun on them, packing index structures less tightly for maximum writing facility.

Note: Do not run COLLAPSE on a relative file. COLLAPSE cannot distinguish between indexed and relative file organization, and it will corrupt relative files. To remove logically deleted records from a relative file, run REORG on the file (see "Relative to Relative" in 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 both corrupt and unsalvageable.

If a file has been corrupted during a COLLAPSE session, restore file integrity to the backup file — not the collapsed file that was corrupted. Run ISAMVERIFY on the backup file; then run REBUILD or REORG if necessary.

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 this statistic 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 you determine 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 indexed file storage requirements.

Procedure

You can invoke COLLAPSE interactively, through a command file, or through a CLI macro command file.

Using COLLAPSE Interactively

When COLLAPSE is used interactively, it prompts for a series of entries that name the files and describe their streamlining requirements. You can store the specifications in a command file to make subsequent COLLAPSE sessions more efficient. To invoke COLLAPSE interactively, enter

```
COLLAPSE
```

The program asks if the disk is backed up. If not, COLLAPSE tells you to proceed no further and returns control to the CLI. Otherwise, COLLAPSE requests the name of the COLLAPSE command file:

```
Type the name of Collapse Command File _____
(If None Strike Return)
```

If you enter the name of a COLLAPSE command file, COLLAPSE displays the command lines in the file and asks if you wish to enter any more files. If you do not, COLLAPSE executes the command file. If you add other files to the COLLAPSE command file, or if you did not specify a command file, COLLAPSE asks:

```
Name of File to be Collapsed _____
(If None Strike Return)
```

After you enter the filename, COLLAPSE asks the following questions about the file:

```
Should Deleted Records be Physically Removed ____
(Y or N)

Primary Key
What Percent Full Should the Index Blocks Be ____
( 50 <= Percent <=99) Default = 99
```

COLLAPSE repeats the last question for each alternate key in the file. Press CR to enter the default value of 99% initially or to echo a value that was entered for a previous key in the same file.

This series of questions creates a command line tailored to the specified indexed file. When all values have been entered, COLLAPSE displays the command line and asks for the name of another file to be collapsed. This procedure is repeated for each specified file until you press CR in response to the request for another filename.

Saving Interactive Specifications

You may save a command file modified during an interactive session. After you enter the additional filenames in the command file, COLLAPSE asks:

```
Should These Instructions Be Saved
(Y or N)
```

If you respond Y, COLLAPSE appends the command lines to the command file that you specified at the beginning of the COLLAPSE dialog. If you respond N, COLLAPSE executes directly. Similarly, if you enter individual filenames, you may save the commands created from your specifications. COLLAPSE asks:

```
Should These Files Be Saved
(Y or N)
```

If you answer Y, COLLAPSE asks:

```
Please Enter Name of New Collapse Instruction File _____
(If None Strike Return)
```

Pressing CR causes COLLAPSE to begin execution. If you enter a new command filename, COLLAPSE creates the .CX command file before beginning execution.

Command Files

You can also execute COLLAPSE commands from a command file. This method is convenient if you have a large number of files to collapse, because you avoid having to repeat the COLLAPSE command for each file. The COLLAPSE command file must be named with the extension .CX and can contain up to 40 lines. Each line of the command file is a specification for streamlining a single file. The command line tells COLLAPSE (1) whether to physically delete logically deleted records from a file and (2) the packing density of each level of the index. The command line has the following format:

```
filename[/D][/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.

A command file is executed by entering at the CLI:

```
COLLAPSE command-filename
```

COLLAPSE displays the file's name on the screen while processing the file and confirms when streamlining is completed. For example, the command file ACME.CX contains the following lines:

```
ACCT$REC/D/90/70/70/70/70
DEBITS/75/50/50/50
WAGE$RPT/99/80/80/80
```

The command COLLAPSE ACME.CX collapses the files ACCT$REC, DEBITS, and WAGE$RPT according to the specifications.

Building a Command File

A command file can be prepared with a system editor or by invoking COLLAPSE interactively (see above). To use the CLI command BUILD to prepare a command file, enter

```
BUILD/N command-filename.CX -.NX
```

This places the names of all indexed files in the current directory into the *command-filename*.CX file. Use a system editor to make the file's contents conform to proper format. Specifically, you must delete all relative files, remove the commas that separate each filename, delete the .NX extensions, and insert COLLAPSE specifications (switches and packing density) for each key. Since the .CX file may contain a maximum of 40 files for processing, excess names must be removed or transferred to another .CX file.

CLI Macro Command Files

You can submit commands to COLLAPSE via a CLI macro command file, which can contain any number of command files. All indexed files on the disk can be collapsed with a single macro command file. Prepare a CLI macro command file with a system editor or by the CLI command XFER. For example, the macro command file STREAMLINE.MC contains the following three command files:

```
COLLAPSE ACME.CX
COLLAPSE ABC.CX
COLLAPSE REGIONAL.CX
```

To execute the commands in the command files, enter STREAMLINE at the CLI.

Log File

After a session, COLLAPSE appends streamlining statistics to a file called COLLAPSE.LG, which may be displayed using the TYPE or PRINT command from the CLI. Any errors encountered during the pass are entered into this log file. If COLLAPSE is running in the foreground, the log filename is FCOLLAPSE.LG.

In the following example, COLLAPSE is run on the file PARTFILE, which has outgrown its planned size. FILESTATS reveals that PARTFILE 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          4/11/82 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. Refer to FILESTATS's prediction of PARTFILE storage requirements, based on index packing densities specified in this COLLAPSE command line.

Error Messages

COLLAPSE error messages indicate either a problem in the command file or a problem in the file being collapsed. Error messages that are caused by problems in the command file have explanations listed. 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 INACCESSIBLE
```

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

```
UNABLE TO DELETE RECORD
```

```
.XD FILE ERROR
```

```
.XD FILE INACCESSIBLE
```

The .XD portion may not be in your directory; 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. It displays revision levels of the compiler that processed an Interactive COBOL source program. 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. CREV also displays the name of the operating system under which the program was compiled and the code revision; it also indicates whether extended features were used.

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.

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

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
```

For example, the entry **CREV PAYROLL** yields the following information:

```
PAYROLL:  COMPILED UNDER RDOS REV. 1.10
                        CODE REV. 5   USING EXTENDED FEATURES
```

This indicates that the program was compiled under RDOS. 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 00. The code revision was 5, and extended features were used.

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. Enter it with or without the .PD extension. CREV 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.

CSSORT sorts an input file and produces a new sorted output file. Files that have been sorted on the same key can be merged in a separate operation into one file. CSSORT uses program-generated work files for the sort operation. Normally, these files are named by the CSSORT program. The command line accepts user-specified work filenames as optional arguments.

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

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 runs under the control of the operating system. The utility can be run from the master terminal in utility mode, or from the concurrent operation console in systems with that option.

Sort Procedure

The minimal command line for the sort operations includes an input file (with input record size if the input file is fixed sequential) an output file, and one key specifier. The basic command line for the sort option 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-1/K [...key-specifier-8/K] [field-specifier-1/F...field-specifier-8/F]
[workfile-1/W...workfile-4/W] [auditfile/A]
```

Global Switch

/N Suppresses the terminal display of the audit information

Local Switches

/t Defines the file type for the input and output files. When no switch is supplied with the in-file or out-file in the command line, a fixed sequential file is assumed by the program. The values that *t* may have are:

	Input-file Type		Output-file Type
/S	Fixed sequential	**/S**	Fixed sequential
/L	Line sequential	**/L**	Line sequential
/V	Variable sequential	**/V**	Variable sequential
/I	Indexed		
/R	Relative		

in-file

The name of the input file. For an indexed or relative file, do not include the .NX or .XD extensions. The maximum filename length is 13 characters. If 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. This argument 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 record size for variable sequential input whenever it is known. Otherwise, the default, 4096 bytes, will diminish CSSORT's efficiency.

out-file/O/t

The name of the output file. The name cannot currently exist. The maximum filename length is 13 characters. If a value for the /t switch is omitted, fixed sequential is assumed.

key-specifier/K

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. Keys can be specified for overlapping areas of the record. 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.

/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 chapter 7 of 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, followed by /F. 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 to be on different devices to increase the efficiency of the program. For example, DP1:DIR1:WORK1.TP/W indicates that WORK1.TP in DIR1 on DP1 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 such information as the operation, input and output filenames, file types, record size, key specifiers, output field specifiers, audit filename, sequence filename, work filenames, time elapsed, and number of records input and output. Do not specify $LPT as both the audit and output file.

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. Rules for creating an alternate collating file are discussed below.

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-specifer-1* must precede *field-specifier-2, work-filename-1* must precede *work-filename-2, in-rec-size-1* must precede *in-rec-size-2,* etc.

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. 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 found in the file are assigned the first (lowest) precedence. This means that the 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.

Observe these rules when creating alternate collating sequences:

- The file can contain only the ASCII characters.
- No character can be repeated.
- The file cannot be longer than 128 characters.

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

```
XFER/A $TTI SWITCHAB <CR>
BA CTRL-Z
```

However, the resulting alternate collating sequence would be incorrect: B, A, digits, remaining upper-case letters, and lower-case 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 upper-case letters and lower-case letters in their normal order.

Using an Alternate Collating Sequence File

In the following example, the user wants to generate a report that indicates by salesperson the products being marketed. 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, 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 format of the stock item numbers was 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 to list the old stock items before the new ones. Thus this sort requires an alternate collating sequence in which uppercase precedes digits. The alternate collating sequence would be created as follows:

```
XFER/A $TTI UPPERCASE <CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZ CTRL-Z
```

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 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 on the screen.
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 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
^	Continuation sign. The sign permits the command to be continued on the next line.
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 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 order of the field specifiers in the command line determines the order of fields in the output file.

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
AUDIT 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 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 on the screen.
`EMPREC`	Input filename. The switch is omitted because the file is fixed sequential.
`95/N`	Required input record size, 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
AUDIT 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 duplicate balances occur, 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 on the screen.

`ACCT$REC/I` Input filename. /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. /O indicates that the file is an output file. /L indicates that the file type is line sequential.

`82:9/K/D/N/L` 82:9 is the field position and length of the account balance field, the major key specifier. /K indicates that the argument is a key specifier. /D indicates that the final order is descending. /N/L indicates that the data type is numeric with a leading sign.

`1:10/K/N` Minor key specifier. /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 th order of these fields stays 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 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

/N Suppresses the screen display of the audit information

/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 are /S for fixed sequential, /V for variable sequential, and /L for line sequential. The default is /S.

Local Switches

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

Command Line Arguments

in-file

The name of a sorted iput file. The maximum filename length is 13 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. The global /i switch names the file type; the default is fixed sequential.

in-rec-size/N

The input record size in decimal integer. If the input files are fixed sequential, the data record size must be included. If the input files are variable sequential, specify the maximum record size for input, if it is known. Otherwise, the default, 4096 bytes, will diminish the efficiency of the program.

out-file/O/t

The name of the output file. The maximum filename length is 13 characters. The output file type (/t) switches are /S for fixed sequential, /V for variable sequential, and /L for line 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 file 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 specify $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 merging 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 same record format, which is shown in the following table:

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	/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 have attempted to merge empty files.

COLLATING SEQUENCE FILE IS TOO LARGE

The collating sequence of a file cannot be longer than 128 characters.

DIRECTORY NOT INITIALIZED

You must initialize an RDOS directory before you attempt to access a file in it. Initialize a directory with the INIT command.

DUPLICATE KEY FIELDS

Key fields cannot be repeated.

DUPLICATE CHARACTERS WERE FOUND IN SEQUENCE FILE

No character may be repeated in the alternate collating sequence file.

END OF FILE

FILE ALREADY EXISTS FILE NAMED filename

FILE DOES NOT EXIST FILE NAMED filename

FILE NOT ACCESSIBLE BY DIRECT I/O

Check the file type. It is probably not random or contiguous.

FILE NOT OPENED

FILE POSITION ERROR

FOUND RECORD TOO SMALL FOR KEY OR FIELD SPECIFICATION

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

FOUND VARIABLE RECORD LENGTH EXCEEDING LIMIT

FOUND VARIABLE RECORD WITH ZERO OR NEGATIVE LENGTH

The length field in the record header contains an invalid value.

ILLEGAL COMMAND SWITCH COMBINATION

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

ILLEGAL FILE NAME FILE NAMED filename

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 for line sequential files, including the terminator, and 4096 for fixed and 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 length must be able to accommodate the position and length of the key and field specifiers.

INSUFFICIENT MEMORY FOR PRESORT

INSUFFICIENT MEMORY FOR THE SPECIFIED SORT
INSUFFICIENT MEMORY FOR THE SPECIFIED MERGE

The specified size of the sort or merge is too large for the amount of memory available for handling the operation. This size depends on the size of the records, and the size and number of the keys and fields. Some element of the sort or merge must be modified. Adapt the command line to include fewer or smaller keys or fields. If possible, allocate more memory to the ground running the CSSORT program.

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

The 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 IS SPECIFIED

Only one alternate collating sequence 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 and two input files when merging.

NO KEY OR OUTPUT FILE FOR SORT

One output file and one key specifier is required for sorting.

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 must be included in the command line if the input file is fixed sequential. It is specified with the /N switch.

OUTPUT FILE ALREADY EXISTS

The output file, which is specified with the /O switch, cannot currently exist.

OUTPUT SPECIFIER RANGE ERROR

OUTPUT RECORD IS TOO LONG FOR LINE SEQUENTIAL

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

PROGRAM ERROR IN GTYPE

PROGRAM ERROR # 1

RECORD SIZE SPECIFIER PERMITTED ONLY WITH FIXED SEQUENTIAL INPUT

You cannot specify a record size unless the input file is fixed sequential.

REVISION INCOMPATIBILITY

THIS OUTPUT FILE MUST BE LINE SEQUENTIAL

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

TOO FEW INPUT FILES

You must specify at least two input files for 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.

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. The work files take the form *workfilename*/W.

UNIDENTIFIED ERROR CODE

UNIDENTIFIED LOCAL SWITCH

An illegal local switch was specified.

UNRECOGNIZABLE ITEM IN COMMAND LINE

WARNING - RECORD LONGER THAN SPECIFIED SIZE

DEFLINES is an Interactive COBOL utility that allows you to set line parameters dynamically. With this utility, you can define line characteristics for any QTY line on an RDOS-based system. Each line can be set to a specific speed and access restriction.

With DEFLINES, you do not have to bring the terminals down and regenerate the system to redefine the line characteristics. The utility stores the parameter information in a sequential binary file named ICX.LD. When the runtime system is initialized, the information is taken from ICX.LD, if it exists, and line definitions are updated on every line that is open.

Procedure

To invoke DEFLINES, select the R option from the Logon Menu and enter DEFLINES at the **RUN PROGRAM** prompt. A sample screen is shown in Figure 2-5.

```
Rev 1.10        INTERACTIVE COBOL LINE DEFINITIONS        11-APR-83

   Specify filename: ICX.LD          Last line number defined: 23

   (1) Enter line number to be defined  25      0-63 for QTY:0-63

   (2) Is the line ALM (A) or ULM (U)?  A

   (3) What line speed do you want?     1       0: clock 0

                                                1: clock 1

                                                2: clock 2

                                                3: clock 3



   (4) Limit access to the line?        N       B-Background only

                                                F-Foreground only

                                                N-No restriction

      Any change? __     What number? __
```

Figure 2-5 *DEFLINES Screen for ALM Line*

DEFLINES prompts you for a filename. The default filename is ICX.LD. If you have several different configurations for QTY lines, you can use a different filename for each configuration and set up ICX.LD as a link to the filename containing the proper line definitions. DEFLINES then prompts

```
ENTER LINE NUMBER TO BE DEFINED
```

Enter any QTY line from 0 to 63. You do not have to define lines consecutively. Press ESC to exit from DEFLINES. If you exit from the utility before entering any line definitions, the file you specified contains default entries for each line. If the line has not been defined within that file, the default entries are zero line speed and no restriction on access. If a line has been defined within that file, the defaults are the previous line speed and access entries.

The next prompt asks you to specify the type of multiplexor attached to the line. The multiplexor type determines what the baud rates can be. You must respond U (for ULM) or A (for ALM). U and A responses can be specified in one file. DEFLINES then asks

```
WHAT LINE SPEED DO YOU WANT?
```

This prompt refers to the desired baud rate for the QTY line. For an ALM, lines are configured on *clocks*, which are jumpered to a certain baud rate. You can select clocks 0-3. For a ULM, the baud rates are directly defined; you can select a value from 0 to 15.

You are then prompted for access restrictions to the line:

```
LIMIT ACCESS TO THE LINE?
```

A B response allows access to the background only, F to the foreground only; N permits both grounds to access the line.

After a line has been defined, DEFLINES prompts `ANY CHANGE`.

- Enter Y to change the definition. You are prompted `WHAT NUMBER`. Enter the number of the entry to be changed.
- Enter any other letter to retain the definition
- Press ESC if you do not want to enter that definition. The message `DEFINITION NOT ENTERED` is displayed. You may continue to define other lines.

DEFLINES allows you to redefine a line if you have entered an incorrect definition. Simply enter the line number to be redefined, and respond to the prompts. The utility substitutes the new parameters for the old ones. If the incorrect line parameters are not the last set of parameters you have entered, you can redefine the incorrect line parameters without redefining the intervening lines. Suppose you are defining consecutive lines starting with 0. While defining line 10, you realize that you have entered an incorrect definition for line 0. Redefine line 0 only; you do not have to redefine the intervening lines.

Although a line can be redefined, it cannot be deleted from the file.

Exiting from DEFLINES

To exit from DEFLINES, press ESC at the prompt `ENTER LINE NUMBER TO BE DEFINED`. The message `LINE DEFINITIONS TERMINATED` appears on the screen, and you return to Logon.

If you make an error in DEFLINES that you cannot correct, you can exit from DEFLINES, delete the file, and start again.

Error Messages

`INVALID LINE NUMBER`

The line number to be defined must be between 0 and 63.

`MUST BE 'A' OR 'U'`

The multiplexor specified must be A (ALM) or U (ULM). Enter your response as a capital letter.

`MUST BE 'B,' 'F,' OR 'N'`

Your response to the access prompt must be B (background access only), F (foreground access only), or N (no restrictions). You must use a capital letter for your response.

`MUST BE BETWEEN 0 AND 3`
`MUST BE BETWEEN 0 AND 15`

If you have a ULM, select a value between 0 and 15; for an ALM, you must select a value between 0 and 3.

The DO utility executes a CLI command file. DO extends the CLI's ability to process command files (.MC files). Whereas an .MC file must contain complete CLI commands, a DO macro file may include up to 512 dummy arguments. When you invoke DO, arguments in the command line are substituted in the file's CLI commands before they are passed to the CLI for execution.

DO enables you to implement complex system functions (e.g., backup, startup, and shutdown) as single CLI commands. This simplifies training of operations personnel, reduces the risk of operator error, and helps in establishing operating standards.

Procedure

To execute a DO command file at the CLI, enter

```
DO command-filename [arg-1] [arg-2] ... [arg-n]
```

Command-filename is the name of a previously constructed file of CLI commands, and *arg-1* through *arg-n* are the actual arguments corresponding to the dummy arguments defined in that command file. The arguments are inserted into the command file according to their positions: arg-1 replaces each dummy argument %1%, arg-2 replaces dummy argument %2%, and so on.

The number of arguments in the DO command line need not match the number of dummy arguments in the DO command file. If you place fewer arguments in the command line, DO replaces the extra dummy arguments with nulls (ASCII 0). You may explicitly specify a null actual argument with the number sign (#).

When it has finished processing all commands in the file, DO returns control to the CLI. You may interrupt execution with the interrupt keys (CTRL-A or CTRL-C).

Command Files

A command file consists of CLI commands. Each command must be complete, although in place of actual arguments it may have dummy arguments of the form %n%, where n=1, 2, 3,...,512.

Examples

You may use the same dummy argument more than once in a command file. The commands in the file SWAPNAMES exchange the names of two files:

```
RENAME %1% $$$TEMP
RENAME %2% %1%
RENAME $$$TEMP %2%
```

To exchange the names of FILEA and FILEB, issue the command

```
DO SWAPNAMES FILEA FILEB
```

The following set of commands can be placed in a command file to compile a COBOL source file and announce its actions:

```
MESSAGE Compiling CRT-format COBOL program:%1%
MESSAGE Error File: %1%.ER
MESSAGE Listing File:%1%.LS
ICOBOL/D/X %1% %1%.LS/L %1%.ER/E
MESSAGE Printing Error File
MESSAGE Printing Listing File
PRINT %1%.ER %1%.LS
```

To terminate operations at the end of the day, a command file named SHUTDOWN is built that contains the following commands:

```
MESSAGE DIRECTORY LISTING IN PROGRESS
DIR %1%; LIST/A/S/E/L; DIR %2%; LIST/A/S/E/L
MESSAGE THE SYSTEM IS NOW SHUTTING DOWN FOR THE DAY.
RELEASE %MDIR%
```

The following command line initiates the actions listed below.

```
DO SHUTDOWN DP0 DP0F
```

1. Displays a message to the operator at the master terminal indicating "directory listing in progress."
2. Makes DP0 the current directory.
3. Outputs to the system printer all directory information for DP0.
4. Repeats steps 2 and 3 for DP0F.
5. Displays a message that the system is "shutting down."
6. Releases the master directory.

The command file could be expanded to include any number of disk drives supported by an RDOS system. Other procedures could be substituted or added to meet the requirements of a particular installation or application.

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. It can be run at any display terminal in the system, and at any number of terminals simultaneously.

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 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 allocation and contiguous allocation. Contiguous index files are larger by (5 * the number of keys) + 1 blocks, which is system overhead built in to insure file integrity while updating the file. 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, enter O at the Logon Menu, which displays the Optional Utilities Menu; then enter F at the Optional Utilities Menu.

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 at the screen. If the printer is busy, the system displays the results. At the second prompt, enter the file type to be examined. Pressing ESC in response to either of these questions stops the program.

FILESTATS then prompts for file data. Input limits appear in the prompts (see examples below). Pressing ESC when entering the data restarts 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, enter 50% packing for all index structures to get worst-case statistics. 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.

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 given 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:

```
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 Indexed Files*

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. 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
```

INQUIRE 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 and printed through PASS or from the CLI. 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 skeleton 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 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:

```
INQUIRE[/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. If a descriptor file was prepared before calling INQUIRE, its name should be included in the command line.

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

listfile/L
The name of an optional 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, INQUIRE 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, INQUIRE 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                        01 CUST-RECORD.
   LABEL RECORDS OMITTED                   03 CUST-AREA PIC XX.
   DATA RECORD IS CUST-RECORD.             03 CUST-KEY PIC X99.
                                           03 NAME PIC X(30).
01 CUST-RECORD.
   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 INQUIRE File and Record Descriptors*

When all keys have been specified, INQUIRE 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 the /L switch 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 INQUIRE cannot access a file if the reliability flags are set. 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 *INQUIRE 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. When the Inquiry program is ready to accept a command, it displays the Command prompt. 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 output directly to a system printer. 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

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 a group name is entered, 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 to be read.

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, 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 forward processing.

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 key 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 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 ESC, OP, or SR can be executed; all other program commands are disabled.

Open file: OP

Opens another ISAM file to be processed. Type the name of the ISAM file 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 or second system line printer, 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 COBOL inquiry program. Press CR to return to Logon.

Preparing a Record Descriptor

When preparing a record descriptor as input to INQUIRE, 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 INQUIRE, 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. INQUIRE 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

INQUIRE 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 INQUIRE 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 INQUIRE is invoked from the CLI.

MORE THAN 23 ELEMENTARY FIELDS

An XMODEL coding restriction has been violated.

OCCURS IS NOT ALLOWED IN FD FILE

An XMODEL coding restriction has been violated.

OUTPUT FILE MISSING

The /O switch is missing from the INQUIRE 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 INQUIRE 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 INQUIRE 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 appear in Table 2-5. For further information on file status codes, see the *Interactive COBOL User's Guide (RDOS)*.

Inquiry Program Error Message	Code
BOUNDARY ERROR, INDEX DEPTH EXCEEDED	24
CORRUPT INDEX STRUCTURE	9B
END OF FILE	10
FILE DESCRIPTOR INCONSISTENCY	9A
FILE IN USE OR RECORD LOCKED ON READ	94
FILE NOT OPEN OR OPEN ON FILE WITH LOCK	92
FILE OPEN ERROR	91
INADEQUATE CONTIGUOUS INDEX BLOCKS	9D
INDEX (.NX) FILE IS FULL	9C
INVALID KEY (DUPLICATE KEY)	22
INVALID KEY, RECORD NOT FOUND	23
INVALID KEY, OUT OF SEQUENCE	21
LINE PRINTER ACCESS TABLE FULL	99
LOGICAL FILE STRUCTURE MAY BE CORRUPT	9F
NOT ENOUGH DISK SPACE TO CREATE FILE	98
OPEN ERROR (DIRECTORY NOT INITIALIZED)	96
PERMANENT HARDWARE ERROR	30
RECORD LOCK LIMIT EXCEEDED	9E
SIMULTANEOUS ISAM OPENS EXCEEDED	97
SYSTEM ERROR (DISK SPACE EXHAUSTED)	34

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 sound and the reliability flags are erroneously set, ISAMVERIFY clears them. It does not attempt to fix any errors it may find; you must do this by running REBUILD or REORG, or by using a backup copy of the file.

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

ISAMVERIFY tests four areas: data and index file system information, data records, key access path descriptors, and 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. 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 your backup file. Run ISAMVERIFY on your 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 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 there is a problem with the logical structure that describes the data records, 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 audit file, if one is specified.

ISAMVERIFY then begins checking the .NX portion of the file. It confirms that it is checking the .NX portion and examines the remaining .NX blocks and the B-tree structure. If ISAMVERIFY 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 specified.

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 operating 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 are only updated 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 ".XD Header or Both Portions Invalid" 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 is the number of alternate keys, the record length, and the next-record-to-append pointer. 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. Another possibility is that the file organization of the .NX or .XD portion, or both portions, is sequential rather than random. Before running ISAMVERIFY on the file again, run XFER on the file, using the local switch /R for a random file or /C for a contiguous file.

BLOCK 0 SYSTEM INFORMATION HAS NON-FATAL ERROR

ISAMVERIFY continues processing and displays the non-fatal error on the next line.

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 does not 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, 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. If you run REORG, assume that both portions of the file are corrupt, and supply header information in the REORG command line. See ".XD Header or Both Portions Invalid" in the REORG section.

INDEX DEPTH NOT BETWEEN 0 AND 6

The .NX portion is corrupt. The index level depth for each key path must be between 0 and 6. ISAMVERIFY stops processing the file.

INVALID FIRST RECORD POINTER

The .XD portion is logically 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 .NX portion is corrupt. The key length must be between 1 and 100 bytes and must be less than the record length. ISAMVERIFY stops processing the file. If you run REBUILD or REORG, specify the key length.

INVALID KEY OFFSET IN RECORD

The .NX portion is corrupt. 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, and stops processing the file. If you run REBUILD or REORG, specify the key's starting position in the record.

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 between 512 bytes and 33 MB, 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 ".XD Header or Both Portions Invalid" 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. If you run REORG, assume that both portions of the file are corrupt and supply header information in the REORG command line. See ".XD Header or Both Portions Invalid" 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 .NX portion is corrupt. The offset of the half block pointer, calculated by using the key length, must agree with the value in the key path descriptor table. ISAMVERIFY stops processing the file.

KEY LENGTH-KEYS PER BLOCK INCONSISTENCY

The .NX portion is corrupt. The maximum number of keys in an index block, computed using the key length value, must agree with the value in the key path descriptor table. ISAMVERIFY stops processing the file.

KEY LENGTH-KEYS PER HALF BLOCK INCONSISTENCY

The .NX portion is corrupt. The maximum number of keys in half of an index block, computed using the key length value, must agree with the value in the key path descriptor table. ISAMVERIFY stops processing the file.

KEY LENGTH-WORDS PER HALF BLOCK INCONSISTENCY

The .NX portion is corrupt. 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. ISAMVERIFY stops processing the file.

KEY LENGTH-WORDS PER KEY INCONSISTENCY

The .NX portion is corrupt. The key entry, as computed by the key length, and the key entry in block 0 must be the same. ISAMVERIFY stops processing the file.

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 are not in strict ascending order. In the case of alternate keys, the feedback, which is appended to each alternate key to insure 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 NOT RECORD ALIGNED

The .XD portion is corrupt. The link field is not 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. 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. If you run REORG, assume that both portions of the file are corrupt and supply header information in the REORG command line. See ".XD Header or Both Portions Invalid" in the REORG section.

NO FILENAME ARGUMENTS

Reenter the command line, specifying filename arguments.

NO RECORDS IN FILE

The .XD portion of the file is empty. 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 ADDRESSED BY LINK IS 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 a lesser value. ISAMVERIFY stops processing the .XD portion and attempts to process the .NX.

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, you may 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.

.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.

The Local Job Entry Monitor (LJE) is an electronic terminal operator that executes files containing CLI commands. In two-ground systems, LJE effectively gives users access to the CLI from the Interactive COBOL runtime environment. In one-ground systems, LJE serves as a batch processor. It allows users to submit jobs during runtime system operation to be executed when the runtime system is stopped.

The command files, or local job files, that LJE executes can be created manually or by applications programs (see "Creating Local Job Files" below). Similarly, job files can be placed on the LJE queue either manually by operators or automatically by programs. LJE takes job files from the queue on a first-come-first-served basis and executes their contents. LJE maintains a number of log files to provide a detailed record of local job execution. Any operator in the COBOL runtime environment can submit jobs. Operators can determine the current status of LJE through Logon or ICEDIT.

LJE is particularly valuable to a large applications development operation. The multiterminal editor ICEDIT includes special facilities for creating and queuing program compilation jobs. Together, ICEDIT and LJE support several programmers concurrently in their tasks of writing, editing, compiling, and testing programs.

Procedure

To activate the Local Job Entry Monitor in the foreground, type the command LJE at the foreground CLI. LJE immediately begins executing any local job files that have been placed on its queue since the last time it was activated.

The DO macro file CSLJE, supplied with Interactive COBOL system software, starts the runtime system, and starts local job execution as soon as the runtime system is stopped. To execute this macro file in a system without concurrency, type the following command at the CLI, where *n* is the number of terminals:

```
DO CSLJE runtime-system-name,n
```

Local Job Files

A local job file consists of a series of CLI commands to be executed as a unit. The commands must be explicit: no dummy parameters of the form %n% are permitted; DO commands, however, are permitted. LJE begins processing each job in the master directory. Thus, jobs that process files in another directory must include an appropriate DIR command or directory specifier prefixes to filenames.

Job filenames must end with an .MC or an .LJ extension. From the viewpoint of the CLI, these files are equivalent. From the viewpoint of LJE, however, there is an important difference:

- When LJE executes *jobfile*.LJ, it deletes the file to prevent inadvertent reexecution at a later time.
- When LJE executes *jobfile*.MC, the original file is not deleted.

Thus, store a job to be executed repeatedly in an .MC file; store a one-time-only job in an .LJ file.

Note: This definition of a CLI command file with an .MC extension is completely consistent with that of standard RDOS macro files with an .MC extension. Such files can be executed manually or submitted to LJE. A local job file *jobfile*.LJ can be executed independently of LJE by entering either of the following at the CLI:

```
@jobfile.LJ@    or    DO jobfile.LJ
```

Creating Local Job Files

Local job files may be created with any system editing program. The .LJ or .MC extension must be included in the job file's name. Job files can include comments: a backslash in column 1 makes the entire line a comment.

If there are no line numbers, each line of the job file begins at column 1. However, ICEDIT reserves the first seven columns. Thus, in files created with ICEDIT, each CLI command line must start in column 8. A backslash in this column flags the line as a CLI comment. Since local job files must be in line sequential format, do not submit an ICEDIT "program" (indexed organization) file to LJE. Instead, use ICEDIT's OUTPUT command, which is specifically designed to facilitate local job file creation. OUTPUT creates the job file in line sequential (CLI) format, names it according to the program being edited and the current terminal number, and strips off the line number or comment field in the first seven columns.

Queuing Local Jobs Manually

To initiate a local job (i.e., to place it in the LJE execution queue), use the JOB command of ICEDIT, or the (J)OB option on the ICEDIT or Logon Menu. This calls a COBOL program that displays the printer check screen shown in Figure 2-12.

```
LJE-00                LOCAL JOB ENTRY          4-11-83
REV. 1.10                                      13:54




          LAST JOB STARTED AT:  12/03/83  13:49:14


CR TO CONTINUE, ESC TO END, OR FUNCTION KEY #1 TO CHECK PRINTER
```

Figure 2-12 *LJE Printer Check Screen*

At the screen in Figure 2-12 press CR to continue. Normally the screen in Figure 2-13 then appears.

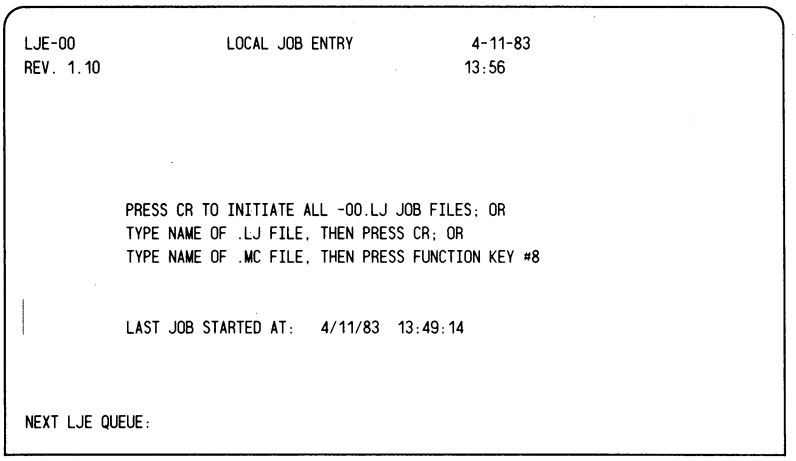


***Figure 2-13** LJE Job Entry Screen*

The job entry screen in Figure 2-13 does not appear if another terminal is displaying the screen or if LJE is momentarily monitoring its execution queue. In these cases, the following message appears:

```
LJE EXECUTION QUEUE IS BUSY -- PLEASE WAIT
```

If the queue remains busy for 30 seconds, the printer check screen returns. Press CR to proceed to the job entry screen, or press ESC to terminate the local job initiation process. The job entry screen indicates the following choices:

- Queue one temporary (.LJ) job file. Enter the name of the job file with or without the .LJ extension. ICEDIT-named files must include the terminal number.
- Queue one permanent (.MC) job file. Type the name of the job file with or without the .MC extension, then press F8. ICEDIT-named files must include the terminal number. A temporary (.LJ) job file is created using the name of the .MC file to be executed.
- Queue all job files with names ending with *nn*.LJ. Type *-nn,* then press CR. This option queues all ICEDIT-named job files created with the OUTPUT command at terminal *nn*, as in the example above. This option has two special cases: (1) press CR to queue all job files created at the user's terminal; (2) at terminal 0, enter a hyphen to queue all local job files created at all terminals in the runtime environment.

After a job is queued, you are returned to the program you came from, ICEDIT or Logon.

Queuing Local Jobs with COBOL Programs

Interactive COBOL programs running at any terminal can place local job files on the LJE queue. Figure 2-14 shows an example of a program written to place a job file on the LJE queue. After a job is queued, control can return to the same program or pass to another COBOL program. The program submitting a local job must perform two tasks:

1. Build a special *passing file* named LJEPASS*nn*, where *nn* is the terminal number. The SELECT statement that defines this file must assign the file to DISPLAY. (This makes LJEPASS*nn* a line sequential file.) The file should contain two lines, each written with a WRITE...BEFORE ADVANCING 1 LINE statement.

   Line 1: the name of a local job file, complete with its .LJ or .MC extension.

   Line 2: the name of a COBOL program to which control passes after the job file is queued. If you omit this line from the passing file, control automatically returns to Logon after LJE queues the job.

2. Pass control to the queuing program LJE with one of the following statements:

   CALL PROGRAM "LJE/M" (if line 1 names an .MC job file)
   CALL PROGRAM "LJE/L" (if line 1 names an .LJ job file)

```
ENVIRONMENT DIVISION.
...
INPUT-OUTPUT SECTION.
FILE-CONTROL.
...
    SELECT LJE-PASSFILE ASSIGN TO DISPLAY, PASSFILE-NAME.
DATA DIVISION.
FILE SECTION.
FD  LJE-PASSSFILE
    LABEL RECORDS ARE OMITTED.
    01  LJE-PASSFILE-RECORD        PIC X(13).
...
WORKING-STORAGE SECTION.
    01  PASSFILE-NAME.
        05  FILLER                 PIC X(7)   VALUE "LJEPASS".
        05  TERM-NUM               PIC XX.
...
PROCEDURE DIVISION.
ACCEPT TERM-NUM FROM LINE NUMBER.
OPEN OUTPUT LJE-PASSFILE.
MOVE "JOB1234.LJ" TO LJE-PASSFILE RECORD.
WRITE LJE-PASSFILE-RECORD.
MOVE "NEXTPROG" TO LJE-PASSFILE-RECORD.
WRITE LJE-PASSFILE-RECORD.
```

Figure 2-14 *Queuing Local Jobs*

LJE Operation

LJE monitors its execution queue periodically to determine if any local jobs have been submitted. In this way, LJE executes jobs as they are placed on the queue. New local jobs can be queued at any time, even while LJE is executing a job. If no jobs have been queued, LJE pauses and tries again. Each time it performs this loop, LJE confirms its operation and displays the date and time:

```
LJE IS RUNNING

mm/dd/yy hr:min:sec
```

When LJE finds that one or more local job filenames have been placed on its execution queue, it begins job execution. For each job that it executes, LJE:

1. Displays the start time and name of the local job file
2. Displays the contents of the job file
3. Renames the original .LJ file to LJE$JOB, so that a previously processed job file is not processed again
4. Executes the commands in LJE$JOB, with the normal display of results on the terminal screen
5. Displays the stop time
6. Updates a number of log files that record the results of jobs. Table 2-6 lists these files and the information they contain. The logfiles files can be displayed using the TYPE or PRINT command from within ICEDIT or from the CLI.
7. Resumes monitoring of the execution queue

Filename	Contents
DONE	Names of each jobfile processed by LJE, with the date and times that execution begins and ends. It can be printed using the CLI command PRINT. This file should be deleted periodically.
LOG	Results of the previous job performed by LJE. This file is overwritten when the next job is executed.
TIME	Start and stop time of the last local job
LJExx.DJ	Start and stop times, and filenames of all jobs performed for terminal number nn. This file should be printed and deleted periodically.
jobfile.DJ	Contents of jobfile.LJ, the start and stop times of the job, and terminal displays associated with the job, such as lists of filenames and CLI error messages

Table 2-6 LJE Filenames and Contents

Examples

You are operating ICEDIT at terminal 2 in subdirectory FRED while writing a COBOL program named INVOICE. Use the following procedure to have LJE compile the program and print the listing and error message files:

1. Insert the CLI commands shown below into the INVOICE program file as comment lines. Start the command lines in column 8. Alternatively, create a separate file containing the commands.

```
003100*DIR FRED
003200*ICOBOL/C/L INVOICE/I INVOICE.QK/E
003300*PRINT INVOICE.<LS QK>
```

2. Use the OUTPUT command to create a local job file containing the three commands. Type LJ followed by a space, and then press CR at the TO FILE prompt. ICEDIT names the job file INVOICE02.LJ, and strips off the line numbers and asterisks as it outputs the lines. Alternatively, type another name at the TO FILE prompt, including the extension .LJ or .MC. All LJE job files must have one of these extensions.

The example above deals with the compilation and listing of the program INVOICE. The local job file INVOICE02.LJ has been created using ICEDIT. Take the following steps to place this job file on the LJE queue:

1. Type JOB at the command prompt after ICEDIT completes the OUTPUT command. The LJE printer check screen appears.
2. Press CR to proceed directly to the job entry screen.
3. Enter the filename INVOICE02 with or without the .LJ extension, terminating with CR to indicate an .LJ file. (F8 indicates an .MC file.)

INVOICE02.LJ is placed on LJE's execution queue, and the ICEDIT main menu returns. At the foreground terminal, LJE displays the following messages as it begins to execute the job:

```
 INITIALIZING JOB

JOB STARTED AT:
4/11/83 14:30:15
INVOICE02,LJE02^

EXECUTING JOB INVOICE02.LJ

DIR FRED
ICOBOL/C/L INVOICE/I INVOICE.QK/E
PRINT INVOICE.<LS QK>
```

When the compilation is complete, the listfile INVOICE.LS and the error message file INVOICE.QK are output to the system printer. After completing the job, LJE displays the following message:

```
INVOICE02.LJ COMPLETED AT:
4/11/83 14:34:20
```

When LJE has completed all other jobs on its queue, it resumes its monitoring routine:

```
JOB COMPLETED AT:
4/11/83 14:34:30
LJE IS RUNNING
4/11/83 14:34:40
```

Monitoring LJE Status

In dual-ground RDOS systems, background runtime system users can determine the status of LJE running in the foreground with the COBOL program LJESTATS. This program is available through the ICEDIT command STATUS or through the utility menus offered by Logon and ICEDIT.

LJESTATS, invoked as a COBOL program, reports the most recent information displayed by LJE in its monitoring routine or in its execution of a job file. LJESTATS also names the job file(s) most recently submitted at the current terminal. It allows the operator to determine whether the system printer is online, offline, or busy. If printer spooling is enabled, the `PRINTER BUSY` message does not appear.

Note: If the system printer is offline, LJESTATS blocks further processing at the terminal until the printer is online again. If the user is unable to bring the printer online, the runtime system must be stopped and restarted before using the terminal again.

Terminating and Restarting LJE

Since LJE is designed to operate continuously, the only way to terminate it is with the interrupt characters (CTRL-A or CTRL-C). If a local job is executing when you interrupt LJE, requeue it manually. You may automatically requeue all remaining jobs on the queue the next time you start LJE.

An incorrect or incomplete CLI command in a local job file ends LJE's execution and returns you to the CLI prompt. You may employ the CLI command TYPE or the (T)YPE function of ICEDIT or Logon to examine the contents of the offending job file, now contained in LJE$JOB. To be saved, LJE$JOB must be renamed before executing LJE again.

To execute the remaining jobs in the queue, issue the following commands:

```
DIR %MDIR%
LJERESTORE
```

You may also use these commands to restart LJE when its operation has been interrupted for any other reason. Issuing the LJE command also restarts local job processing but with an empty execution queue.

The MESSAGES utility enables you to modify the contents of the Interactive COBOL runtime system's error message file. Using MESSAGES, you can translate the runtime system's error messages to another language or make whatever modifications are necessary to meet the needs of the operator. You may examine or print the error message file, and then modify specific error messages. These steps are carried out through an interactive dialog.

Each message in the error message file is identified by a number. Error messages are coded into five categories:

- Data validation error. A message is displayed when an operator enters data that is invalid according to its PICTURE specification.
- Fatal runtime error. The system stops the program's execution and displays a message.
- Status message. The program's execution is stopped or interrupted. The status of the program is displayed.
- Debugger error. If the debugger is running, input that does not conform to the debugger commands produces an error message.
- Master terminal message. Certain messages are displayed only with functions performed at the master terminal.

Procedure

MESSAGES is invoked from the CLI. To start the program, enter MESSAGES at the master terminal. The following prompts appear:

```
MESSAGE FILENAME : ________________
MESSAGE NUMBER : ___
MESSAGE :
NEXT : _
```

Make entries according to the following instructions. Terminate all entries with CR.

1. Enter ICX.ER, the name of the error message file at MESSAGE FILENAME. Press CR alone to return to the CLI.
2. Enter the number of the message at MESSAGE NUMBER. The text of the message appears on the MESSAGE line. Pressing CR alone moves the cursor to NEXT.
3. After the current message appears at MESSAGE, you can modify it. The maximum message length is 60 characters. When you press CR, the error message file is updated with the displayed line, and the cursor moves to NEXT. Press CR alone to retain the original message.
4. At NEXT:

   Enter P to print the entire message file.

   Enter U (Up) to display the number of the preceding error message at the MESSAGE NUMBER prompt; press CR at MESSAGE NUMBER to display the corresponding message.

   Enter D (Down) to display the number of the next error message at the MESSAGE NUMBER prompt; press CR at MESSAGE NUMBER to display the message.

   Enter any other character to return to MESSAGE NUMBER for another entry. Press CR alone to return to the CLI.

Table 2-7 gives the code, number, and text of each message in the runtime system's standard error file.

Code*	Number	Message
V	01	CHARACTER MUST BE ALPHABETIC
V	02	CHARACTER MUST BE ALPHANUMERIC
V	03	CHARACTER MUST BE NUMERIC
V	04	SIGN MUST BE RIGHTMOST CHARACTER
V	05	TOO MANY SIGNS ENTERED
V	06	SIGN MUST BE LEFTMOST CHARACTER
V	07	FIELD DOES NOT PERMIT A DECIMAL POINT
V	08	TOO MANY DECIMAL POINTS
V	09	NO DIGITS ENTERED
V	10	DATA ENTRY IS REQUIRED
V	11	FULL FIELD IS REQUIRED
V	12	FIELD DOES NOT PERMIT A SIGN
V	13	TOO MANY INTEGER PLACES ENTERED
V	14	TOO MANY DECIMAL PLACES ENTERED
V	15	ILLEGAL IMBEDDED BLANKS
V	16	TRANSMISSION ERROR, RE-ENTER LAST CHARACTER
D	17	ILLEGAL COMMAND
D	18	UNDEFINED NAME
D	19	ILLEGAL CHARACTER IN NUMERIC FIELD
D	20	SUBSCRIPT ERROR
D	21	WRONG DATA TYPE
M	22	ENTER TERMINAL NUMBER TO ABORT
M	23	TERMINAL ABORTED. <CR> TO REPEAT, <ESC> TO RETURN TO LOGON
M	24	TYPE <CR> TO ENTER MESSAGE, <ESC> TO RETURN TO LOGON
M	25	A TERMINAL IS STILL ACTIVE. <CR> OR <ESC> TO RETURN TO LOGON.
M	26	A PRINTER IS STILL ACTIVE. <CR> OR <ESC> TO RETURN TO LOGON.
M	27	THE SPECIFIED LINE PRINTER IS BUSY. TRY AGAIN? (Y/N)
M	28	THE PRINTER DOES NOT RESPOND. THE JOB CAN NOT BE ABORTED.
M	29	JOB ABORTED BY OPERATOR AT PAGE
M	30	THE FILE IS IN USE
F	31	INDEX REGISTER OVERFLOW
F	32	SUBSCRIPT OUT OF RANGE
F	33	PERFORM n TIMES
F	34	PERFORM STACK OVERFLOW
F	35	FATAL COBOL PROGRAM I/O ERROR. PROGRAM ABORTED.
F	36	UNDEFINED PROCEDURE
S	37	JOB ABORTED BY OPERATOR
S	38	STOP RUN
S	39	FATAL RUNTIME SYSTEM INTERNAL ERROR
F	43	LOGON PROGRAM TOO LARGE. LOGGING OFF.
F	44	LOGON CODE REVISION IS NOT VALID. LOGGING OFF.
F	45	LOGON PROGRAM FILE IS NOT VALID. LOGGING OFF.
F	46	LOGON PROGRAM WAS NOT FOUND. LOGGING OFF.
F	47	SYSTEM ERROR LOADING THE PROGRAM. LOGGING OFF.
F	48	SYSTEM ERROR PROCESSING THE USING DATA. LOGGING OFF.

*V=data validation; M=master terminal; D=debugger; S=status; F=fatal program

Table 2-7 Runtime Error Messages

NOTES is an interactive facility for preparing and sending messages and announcements and soliciting responses. Each note is written once and stored by the computer for later recall. Each note may then be read and reread by anyone with access to the notes file. Any user may write a response to that note, which is added to the notes file, thus giving all users access to the response. When a note is written, the program stores the user's name with the note. NOTES associates a password with each name to ensure that the name is authorized.

Procedure

NOTES is actually a series of programs linked together. The NOTES programs must be stored in the same directory as the Interactive COBOL runtime system. To invoke NOTES, select (N)otes from the Logon Optional Utilities screen, or select (C)obol Program from the Logon Menu and type NOTES at the RUN PROGRAM prompt.

NOTES employs a number of special function keys. Table 2-8 identifies and defines these keys.

Key	Function
Help (F1)	Provides an explanation of available functions
Back (F2)	Moves cursor to a previous note, response, or line
Start (SHIFT-F2)	Moves cursor to the first note or line Next (F3) Moves cursor forward to the more recent notes in a file or to the next response (often duplicated by CR)
End (SHIFT-F3)	Moves cursor to the end of the file, the remaining responses, or the last line
Choose (F4)	Enables you to enter the number of a response, choose a line number, or choose an option
Write (F8)	Adds information, such as a note or a line of text
Erase (SHIFT-F8)	Removes information
Sure (CTRL-SHIFT-F8)	Confirms a request for a destructive action
ESC	Acts as an escape mechanism throughout the NOTES system. Press ESC repeatedly to exit NOTES.

Table 2-8 NOTES Function Keys

Entry to NOTES

The program first asks for your name. If NOTES cannot find your name, it asks if it is spelled correctly. If you answer N, the program prompts for reentry of the name. If you answer Y, NOTES asks you to choose a password. When this is done, the name (15 characters) and the password (8 characters) are stored for future reference.

If NOTES finds the name, it asks for the corresponding password. This password is echoed with asterisks as it is typed in. If the password is correct, the notes file choice page appears (see Figure 2-15). If the password is incorrect, you are notified and can reenter the password or press ESC to enter a different name.

To assign or change a password, press WRITE (F8) after the name has been typed. NOTES then asks for a new password. Since the new password cannot be seen, NOTES requests that you type it twice to make sure it is spelled correctly.

Notes File Choice Page

The notes file choice page prompts for the name of the notes file to be read (see Figure 2-15).

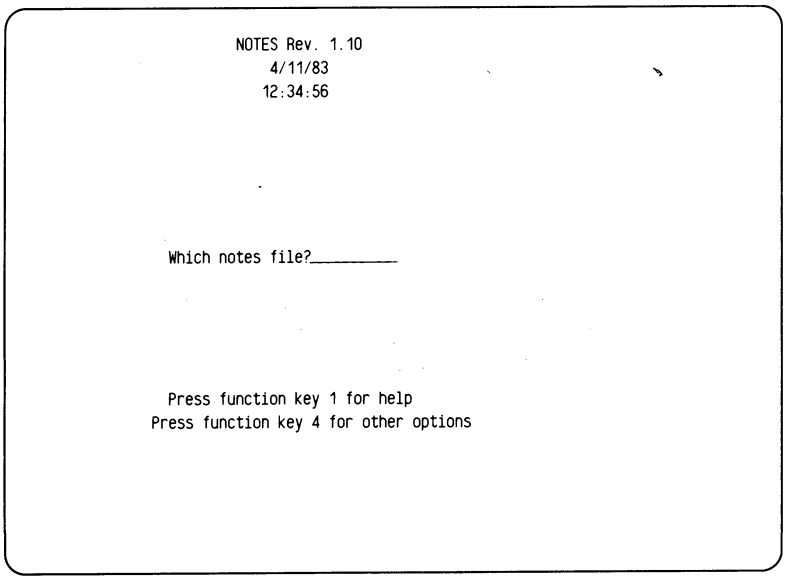


Figure 2-15 *The Notes File Choice Page*

NOTES notifies you if the name of the notes file you enter does not correspond to an actual notes file; you may enter another name. If the second name does not correspond to an existing notes file, the program moves to the notes file index, which is described below.

If you do not know the name of a given notes file, you may press the Choose key (F4) to display the special options page. One of the options is a list of all the notes files that currently exist (see "Special Options" below). To exit NOTES at the choice page, press ESC.

Special Options

Special options are available from the notes file choice page by pressing the Choose key (F4). The options are:

- Create a new notes file.
- Destroy an old one.
- See a list of existing notes files.

Anyone may create a new notes file, but, in general, a notes file may be destroyed only by the person who created it.

A title is required when creating a notes file. This is usually a word or phrase that describes the contents of the notes file. The title should not be confused with the notes file name. The name is the external filename; the title is merely an explanatory heading.

The supervisor is afforded wider privileges under the special options page. This person may destroy a notes file created by someone else. This is the only exception to the rule noted above. The supervisor may also clear passwords and remove inactive names from the names file.

Notes File Director

Each notes file has a director who oversees the contents of that file. When a notes file is first created, the name of the person creating it is entered as the director of the file. A notes file director can:

- Determine or change the title of the notes file
- Delete notes
- Choose a new director for the notes file

As a general rule, users may change or remove notes they have written only if there are no responses to a given note. However, the notes file director may remove a response from the middle of a string of responses, or remove a main note and all of its responses.

A director may not alter the contents of any note written by someone else or change one of his or her own responses if other people have since responded. When a director deletes a note, a notice to this effect is left in place of the note. This notice cannot be removed except by destroying the entire notes file and creating a new one.

Notes File Index

NOTES invokes the notes file index if your second attempt to enter the name of a notes file fails. The first screen for the notes file index lists the most recent main notes written in the current notes file. Each line of the list gives the following information, from left to right:

- Sequence number of the note
- Date the note was written
- Title of the note (a short phrase characterizing its contents)
- Number of responses to the note
- Author of the note

Certain function keys enable you to employ the notes file index. These are defined in Table 2-9. You can also read a note from the notes file index by entering its number and pressing NL.

Key	Function
Next (F3)	Shows recent notes in the file. This key has no effect if you are viewing recent notes.
End (SHIFT-F3)	Moves to the end of the file, showing the most recent notes written
Back (F2)	Shows earlier notes in the file
Start (SHIFT-F2)	Shows the notes starting with #1
Write (F8)	Moves to the Note Editor to write a new main note
Help (F1)	Indicates the available options
ESC	Returns you to the notes file choice page

Table 2-9 Function Keys for Notes File Index

Note Display Page

The note display page shows the text of the note, who wrote it, when it was written, the title of the notes file, the number of the main note and its title. If a main note is being read, the number of responses is displayed; if a response is being read, NOTES displays the response number of the note and the total number of responses to this main note.

The function keys that are active for the note display page are defined in Table 2-10.

Key	Function
Back (F2)	Shows the previous response. If a main note is being read, Back does nothing.
Start (SHIFT-F2)	Moves to the previous main note
Next (F3)	Moves to the next response to the current main note. If there are no more responses, Next does nothing. Pressing NL displays the next main note.
End (SHIFT-F3)	Skips any remaining responses to the current main note and moves to the next main note
Choose (F4)	Allows you to enter the number of the response to be read
Write (F8)	Moves to the Note Editor so you can write a response
Plus (+)	Displays the last response to the current main note
Hyphen (-)	Displays the current main note (useful while reading responses)
ESC	Returns you to the notes file index

Table 2-10 Function Keys for the Note Display Page

Once a note has been written and stored, it can be read by anyone who has access to the notes file. If the note has no responses, the writer of the note may choose to edit the note or remove it completely. However, once a note is no longer the last one in the chain, it may not be changed in any way — this includes removing the note from the file.

The Note Editor

The Note Editor is a line-oriented editor. A note may be up to 20 lines long. If more room is needed, continue the note in a response (see the note below for deleting a multipage note). The Note Editor starts at line 1. As each line is completed, the Note Editor adds 1 to the line counter. The editor wraps around to the top again when line 20 is completed.

To edit a note, press Erase (SHIFT-F8) while reading it. The next screen gives you three choices. To remove the note, press Sure (CTRL-SHIFT-F8); to edit it, press NL; and to leave it as is, press ESC. If you choose to edit the note, the Note Editor copies the note into memory and removes it from the notes file. If you exit the editor without storing the note, the contents are lost, even if no changes were made. The function keys that are defined for the Note Editor are given in Table 2-11.

Key	Function
Back (F2)	Backs up to a previous line. Pressing BACK from line 1 causes the editor to wrap around to the bottom line.
Start (SHIFT-F2)	Moves to line 1
Next (F3)	Moves to the next line
End (SHIFT-F3)	Moves to the last line
Choose (F4)	The editor prompts for a specific line
Write (F8)	Adds lines to the note
Erase (SHIFT-F8)	Removes lines from the note

Table 2-11 Note Editor Function Keys

If you are writing a main note, the Note Editor requests a title for it. This should be a short phrase describing the contents of the note, for example, "Vacation policy." When the note has been edited, press ESC. The editor asks for confirmation. Pressing NL stores the note; ESC resumes editing of the note. To abandon the note entirely, press Sure (CTRL-SHIFT-F8).

To delete a multipage note: (1) delete the last response, (2) delete each previous response until all responses have been removed, and (3) delete the main note.

Error Messages

Under certain exceptional conditions, NOTES indicates that a fatal error has occurred and displays an error message. These error messages are explained below.

`INVALID CONNECTOR FILE`

When NOTES moves between programs, it writes a special file containing information that must be saved across the transfer. This message appears if this special file is not the expected size. This may be caused by a disk error, or by trying to enter the NOTES system at a place other than its beginning.

`HELPFILE NOT FOUND: filename`

NOTES was unable to locate the file containing help text. This may be caused by a disk error, or by trying to enter the NOTES system at a place other than its beginning.

`INVALID KEY ENCOUNTERED IN NOTESFILE`

NOTES was unable to find a record in the notes file. This may be caused by a disk error or a memory error. If this happens repeatedly with a given file, the file is probably unsalvageable and should be deleted.

`INVALID KEY WHEN WRITING IN NAMES FILE`

NOTES was unable to find a record in the file that stores the user list. This may be caused by an error in disk or memory. If this happens repeatedly, the names file (NOTE$NAMES.NX and NOTE$NAMES.XD) should be deleted. NOTES re-creates the file the next time someone enters NOTES, and all names and passwords must be reentered. However, no other information is lost.

`NOT A NOTESFILE: filename`

After a transfer between programs, NOTES found the specified file, but it is not a notes file. This may be caused by a disk error, or by entering the NOTES system at a place other than its beginning.

NOTESFILE NOT FOUND: filename

After a transfer between programs, NOTES was unable to find the notes file. This may be caused by a disk error, by destroying the notes file between transfers, or by trying to enter NOTES at a place other than its beginning.

PROGRAM NOT FOUND: program-name

When attempting to transfer between two programs, a program was not found. This can happen if the Interactive COBOL runtime system is started in a directory other than the one containing the NOTES programs, if one or more of the NOTES programs has been destroyed, or if there is a disk error.

INVALID KEY ENCOUNTERED IN INDEX

NOTES is unable to find a record in the index of notes files. If this happens repeatedly, you may have to re-create the listing of notes files. To do this:

1. Determine the names of all notes files. Since these are index files (implemented as *filename*.NX and *filename*.XD), you can use the CLI LIST command.
2. Rename the existing files or move them to another directory.
3. Delete NOTES' list of notes filenames. The list is stored in NOTE$FILES.NX and NOTE$FILES.XD.
4. Using the notes file choice page, build a new NOTE$FILE by creating notes files with the names from step 1.
5. At the CLI, delete the new notes files.
6. Restore the real notes files by renaming or moving them from their temporary storage place.

Note: By completing the above procedure, you are listed as director of all notes files in the rebuilt NOTE$FILES. But the notes files themselves retain the names of their original directors. They can correct the NOTE$FILE listing with the Director Option on the notes file choice page.

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.

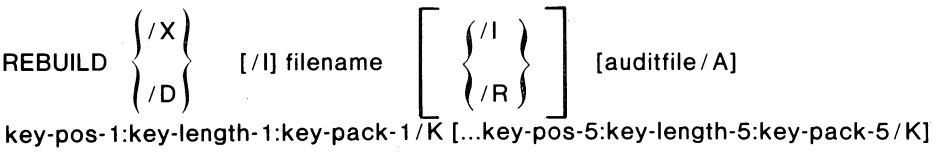
If the .XD portion of the file is sound, REBUILD attempts to restore corrupted links from one record to another, find loops in the record links, and fix the links by using the .NX file. If the .XD portion of the file is in good order, it rebuilds the .NX file from the bottom up, i.e., from the terminal levels up to block zero.

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 operates with as few assumptions as possible; however, an underlying assumption is that the RDOS file structure is intact. REBUILD can repair the file structure but it cannot fix the data itself if it has become corrupted.

REBUILD should be run only after ISAMVERIFY has determined that either the .NX or .XD portion of an ISAM file is corrupt. ISAMVERIFY reports how the file has been corrupted; you must supply the information necessary to enable 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. The local switches /I and /R indicate whether the file is indexed or relative.

auditfile/A
The name of the auditfile. If you omit this argument, REBUILD creates an auditfile named *filename*.AU. If this file already exists, it is deleted and re-created. If you are using REBUILD in interactive mode, you are asked to confirm this deletion. If you do not, REBUILD terminates without operating on your file.

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 above key arguments are identified by the /K switch. REBUILD interprets the number of alternate keys as one less than the number of times the /K switch is specified.

Be sure to specify the key position and length arguments exactly as they occurred in the original file. If they are specified incorrectly, REBUILD may work but the file will be corrupt and will not pass ISAMVERIFY's tests.

You may specify a different key packing density, however, as long as it falls within REBUILD's limits. For example, if the original file had a packing density of 60%, 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 it is corrupt. If you use this switch, 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 you include this switch, the .NX portion of the file must have passed all of ISAMVERIFY's tests for file integrity. REBUILD reconstructs the .XD file and links using the .NX file.

Note: The /X and the /D switches are mutually exclusive; both cannot be used in one command line.

/I Invoke REBUILD interactively. Interactive use of REBUILD is recommended. However, REBUILD cannot be submitted to the RDOS batch processor if it is invoked interactively.

Local Switches

/I Indicates an indexed file (default)

/R Indicates a relative file

/K Specifies the key information: position in record, length, and packing density

Rebuilding the .XD Portion of an ISAM File

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

REBUILD/D[/I] filename {/I, /R} [auditfile/A]

The local /I switch indicates an indexed file, the /R switch, a relative file. Because you are using the good .NX to rebuild the .XD, no key information is needed. If you enter any key information, an error message is displayed.

Rebuilding the .NX Portion

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

REBUILD/X[/I] filename/R [key-pack/K] [auditfile/A]

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 they are entered. The packing density, in the form *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, as indicated in the general form of the command line.

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 you enter the command line, except by entering CTRL-C or CTRL-A. If you enter one of these interrupts 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 invoked in a batch stream so that it can run 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/I AUDIT/A :15:/K 16:10:/K 20:17:85/K 40:5:70/K 46:12:95/K
```

/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	DATA is the file that REBUILD is to process. The file type switch is omitted, so an indexed file is assumed.
AUDIT/A	AUDIT is the name of the audit file. If no audit file had been specified, REBUILD would have created the audit file DATA.AU.
:15:/K	The primary key specifier. No key position is specified, so it is assumed to be 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 a key.
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 bytes, and the packing density 70%.
46:12:95/K	The fourth alternate key specifier. The key position is 46, the key length 12 bytes, 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 in Figure 2-16. Since REBUILD is running in interactive mode, it prompts:

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

The default is YES. If you enter NO, REBUILD terminates without accessing the corrupted file. This is the last point that you can exit from REBUILD and be sure that the file has not been altered. If you invoke REBUILD in batch mode, you have 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 the .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 2-16** REBUILD Screen*

Messages

Because of its large number of error and warning messages, REBUILD has a specific error message file called REBUILD.ER. If the error file is missing or is in the wrong directory, an error number rather than the message is displayed. If the utility can access the file, the number is not displayed. The error number is listed at the right of each message.

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.
- RDOS system errors

ARITHMETIC ERROR 62

Internal consistency error.

ARITHMETIC ERROR IN COMPUTING DISK SPACE 52

Internal consistency error. REBUILD has failed an internal consistency check in determining disk space. For example, it may have computed a disk space greater than (2**31) - 1, which is impossible.

```
ATTEMPT TO REBUILD THE DATA (.XD) FILE HAS FAILED    110
```

Use REORG on the file instead.

```
AUDIT FILE WILL NO LONGER BE USED    89
```

REBUILD is unable to write to the audit file. REBUILD continues operating but no longer writes to the audit file.

```
CHANNEL UTILIZATION INCONSISTENCY    90
```

Internal consistency error.

```
CORRUPT LINKS WERE FOUND IN THE .XD FILE    98
UNABLE TO RESOLVE BAD LINKS WITH THE .NX FILE    99
```

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.

```
DEFAULT AUDIT FILE SELECTED    25
```

Status message.

```
DO NOT UTILIZE EXCESSIVE DISK SPACE WHILE REBUILD IS IN PROGRESS    53
```

Status message.

```
ERROR IN COMMAND LINE PROCESSING
```

```
FATAL ERROR: PROGRAM STACK OVERFLOW    92
```

Internal consistency error.

```
FILE ALREADY EXISTS - DELETING IT filename    77
```

The audit file specified already exists. REBUILD deletes the old audit file before creating a new one. If you are in interactive mode, REBUILD will ask you to confirm the deletion.

```
GLOBAL /X AND /D SWITCHES ARE MUTUALLY EXCLUSIVE    18
```

Syntax error.

```
INCONSISTENCY IN ERROR HANDLING
ABORTING OPERATIONS DUE TO FATAL ERRORS
```

Internal consistency error. There is a software or hardware problem in your CPU. Contact your Data General representative.

```
INDEX DEPTH OVERFLOW WHILE REBUILDING THE .NX FILE    87
```

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

```
INSUFFICIENT CONTIGUOUS DISK SPACE - CONTINUING ANYWAY    51
```

If the .XD portion of the file is contiguous, REBUILD attempts to create a contiguous .NX portion. If disk space is insufficient, REBUILD displays this message and continues, creating a random .NX portion.

```
INSUFFICIENT DISK SPACE TO CONTINUE OPERATIONS    50
```

REBUILD terminates after displaying this message; it has not modified the corrupted file. Delete enough files to give REBUILD sufficient space and run REBUILD again.

```
INVALID COMMAND LINE ARGUMENT(S)    6
```

Syntax error.

INVALID FILE REVISION IN .NX FILE 41
INVALID FILE REVISION IN .XD FILE 42

The file revision in the specified portion is not current.

INVALID GLOBAL SWITCH SPECIFIED 17

Syntax error. Permissible global switches are /X, /D, and /I.

INVALID KEY LENGTH ARGUMENT SPECIFIED 21

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 20

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 7

Syntax error.

INVALID NEXT RECORD TO APPEND POINTER IN THE .NX FILE 35

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

INVALID NEXT RECORD TO APPEND POINTER IN THE .XD FILE 32

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

INVALID NUMBER OF ALTERNATE KEYS FOUND IN THE .NX FILE 33

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

INVALID NUMBER OF ALTERNATE KEYS FOUND IN THE .XD FILE 30

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

INVALID PACKING LIMIT ARGUMENT SPECIFIED 22

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

INVALID RECORD ADDRESS ENCOUNTERED 109

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 34
INVALID RECORD LENGTH FOUND IN THE .XD FILE 31

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.

ISAM REBUILD INTERNAL ERROR CODE = nnn

REBUILD has attempted to access the error file REBUILD.ER, and it does not exist in your current directory or you are not linked to it.

KEY COLLATING SEQUENCE ERROR 19

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

KEY IS BEYOND END OF RECORD 23

Syntax error.

KEY UTILIZATION INCONSISTENCY 91

Internal consistency error.

MAP.DR ERROR - TOO MANY BLOCKS ON DISK 54

A hardware error has occurred.

MULTIPLE AUDIT FILES SPECIFIED 12

Syntax error. Only one auditfile can be specified. If you do not specify an auditfile, REBUILD creates *filename*.AU.

MULTIPLE FILE NAMES SPECIFIED 13

Syntax error. You may have forgotten to use /A after the auditfile name.

NAME OF FILE TO BE REBUILT IS NOT THE FIRST ARGUMENT 10

Syntax error.

NO ACTION SPECIFIED 19

Syntax error.

NO FILE NAME SPECIFIED 16

Syntax error.

NO KEY ARGUMENTS SPECIFIED 12

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

NO KEY POSITION SPECIFIED - ASSUMED ONE 24

Status message.

NO PROBLEMS WERE ENCOUNTERED WHILE SCANNING THE .XD FILE 101

Status message.

NUMBER OF KEYS SPECIFIED DOES NOT MATCH FILE INFORMATION 36

Status message in interactive mode. The .NX and .XD portions both specify a certain value for the number of keys, and you have specified another value. You have the option of confirming your entry or directing REBUILD to use the value found in the .NX and .XD.

NUMBER OF RECORDS SCANNED IN .XD FILE IS: nnn 104

Status message. REBUILD displays this update message for each key path scanned.

PLEASE BACK UP YOUR FILES BEFORE RUNNING REBUILD ON THEM 94

Status message.

PROCESSING TERMINATED BY A CTRL-A 0

You have used a control character to interrupt processing.

PROGRAM ERROR - SOFTWARE INCONSISTENCY 95

Internal consistency error.

RECORD ADDRESS COMPUTATIONAL ERROR 61

Internal consistency error. For example, REBUILD may have determined that a record address is negative or that it is greater than (2**31) - 1.

TOO MANY ARGUMENTS 27

Syntax error.

TOO MANY BLOCKS WRITTEN TO .NX FILE 105

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

TOO MANY KEY ARGUMENTS SPECIFIED 8

Syntax error.

UNABLE TO CLOSE FILE filename 75

System error.

UNABLE TO CREATE NEW FILE filename 71

System error. The most likely cause is that REBUILD does not have enough disk space to continue.

UNABLE TO DELETE FILE filename 72

The file has a permanent attribute, a system error has occurred, or the file is being accessed by another user.

UNABLE TO FIND .XD RECORD KEY IN THE .NX FILE 86
.XD OR .NX FILE MAY BE CORRUPTED 96

REBUILD stops operating. Use REORG on the file instead.

UNABLE TO GET FILE STATUS filename 79

Fatal error. REBUILD has encountered a problem while attempting to read the disk. You most likely have a hardware problem.

UNABLE TO GET POSITION IN FILE filename 80

Fatal system error. REBUILD is unable to determine the next character position in a file.

UNABLE TO LOAD PROGRAM OVERLAY FILE 2

Fatal system error. REBUILD is unable to load the overlay file REBUILD.OL.

UNABLE TO OPEN CONSOLE FOR I/O 1

System error. You most likely had a hardware failure.

UNABLE TO OPEN FILE filename 70

System error. If REBUILD is unable to open the .NX file, it displays another message to that effect.

UNABLE TO OPEN .NX FILE 43

The .NX file does not exist, the directory is not initialized, the filename is illegal, or the device is not in your system.

UNABLE TO READ FROM FILE filename 73

A system error has occurred, or the file is read-protected. If you have used XFER to copy the file, you may have forgotten the local /R switch.

UNABLE TO READ RECORD KEY IN .XD FILE 63

System error. You may have a hardware problem.

UNABLE TO READ THE .NX FILE BLOCK ZERO 44

REBUILD stops after displaying this message. Most likely, the file is not random or contiguous. Check the file organization.

UNABLE TO SET POSITION IN FILE filename 76

Fatal system error. REBUILD is unable to set the current file pointer.

UNABLE TO WRITE TO FILE filename 74

A system error has occurred or the file is write-protected. If you have used XFER to copy the file, you may have forgotten the local /R switch.

UNABLE TO WRITE TO THE AUDIT FILE 88

The audit file is write-protected, a disk failure has occurred, or a memory failure has occurred.

USER WILL NOT CONFIRM FILE DELETION 78

Status message appearing only in interactive mode. REBUILD terminates operation.

USER WILL NOT CONFIRM FILE INFORMATION AS DISPLAYED 81

Status message appearing only in interactive mode. The user has refused to confirm that system information is to be used if discrepancies exist between user-supplied and system information. REBUILD terminates operation after displaying this message.

WARNING: TIME CONVERSION OVERFLOW 92

Internal consistency error.

REORG is a file conversion utility that enables you to change the logical and/or physical 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
- Converting between random and contiguous disk allocation schemes
- Magnetic tape import/export of fixed-length records

Reliability Checking

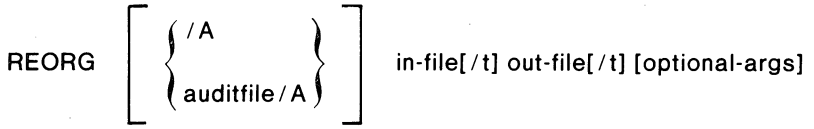
If an ISAM file's reliability flags are set and you attempt to use it as an input ISAM file without the /D or /E switch, REORG displays the following message:

```
LOGICAL FILE STRUCTURE MAY BE CORRUPT. RUN ISAMVERIFY.
```

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 auditfile. 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. Do not use the same filename for both the input and output files if both files are the same file type. If the output file does not exist, REORG creates it. If the output file is an existing indexed or relative file, REORG inserts the additional records, overwriting duplicates. 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). Do not specify the .NX or .XD extension.

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. This switch cannot be used with the local /A switch, which specifies an auditfile.

Local Switches

auditfile/A
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 global /A switch, which directs audit information to the screen. The audit file 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 file and the output file the same name.

/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.

Note: When a line sequential file is to be 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 for REORG.

Optional Arguments

The syntax of the optional arguments and a brief description of their uses appear in Table 2-12. 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.

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 keys when output file is indexed. Up to 4 can be specified.
Contiguous output file	data-size:index-size/C	Output ISAM file is contiguous.
	file-size/C	Output sequential file is contiguous.
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 *character count* times in the output record

Table 2-12 REORG Optional Arguments

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 1. 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 is the size as declared in the File Section of the original COBOL program. Use this argument if the input file is fixed sequential. Do not include the input record size if the input file is an ISAM file.

When recovering an ISAM file by dumping the renamed .XD portion to tape, REORG treats the renamed portion as a sequential file. In this case, the size of the input record must be declared. However, it is determined differently than that of a sequential file (see, for example, "Recovering an ISAM File from Tape").

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 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 primary key or the first, second, third, or fourth alternate key. The form of the argument is n/S, where *n* is 0, 1, 2, 3, or 4. 0/S is the primary key, 1/S the first alternate key, and so on.

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.

contiguous output file

An output file can be allocated contiguous space on disk with the following argument:

data-size { . : } index-size/C

Data-size is the size of the data portion in blocks. *Index-size* is the size of the index portion in blocks. Either or both portions of the file may be contiguous. To specify a contiguous data portion, enter 0 for index-size. To specify a contiguous index portion, enter 0 for data-size.

The following argument gives a sequential file contiguous space on disk, where *file-size* is given in blocks:

file-size/C

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 of the access key values. 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, however, 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.

.NX Portion Is Bad

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 2-17. 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 2-17** REORG Audit Information*

.XD Header or Both Portions Invalid

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

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

in-file
The name of the renamed .XD portion. The /D switch (purge logically deleted records) or the /E switch (retain logically deleted records) must appear after the input file.

/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 keys (/Q). The default is indexed.

key specifiers
The position and length of a key, in the form *key-pos:key-length*, followed by /K switch for the primary key and the /A switch for any alternate keys.

num-alt-keys/N
The number of alternate keys in the file. 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 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 record 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.

Recovering Relative Files

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 $LPT/A CUSTOMERS1/D CUSTOMER00/Q
```

Audit information is printed on the system printer. The /D switch indicates that logically deleted records should not be included as the new index is built. The /Q switch allows you to preserve 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 ITEMS$83 to the indexed file CLEANUP, retaining the same record size and key:

```
REORG $LPT/A ITEMS$83 CLEANUP
```

All logically deleted records of ITEMS$83 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 SALESAUDIT/A 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 ITEMS$83 to the system printer:

```
REORG/A ITEMS$83 $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 screen. To obtain a permanent copy of ITEMS$83 to print, simply specify an output filename with the /L switch rather than $LPT.

Relative to Relative

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

```
REORG $LPT/A CUSTOMERS1/R CUSTOMER00/R
```

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

```
REORG $LPT/A 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>
```

These commands give the new version of the file the proper name:

```
RENAME CUSTOMER00.NX CUSTOMERS1.NX
RENAME CUSTOMER00.XD CUSTOMERS1.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 customer number as the primary key, and has order number and customer region as alternate keys, enter:

```
REORG $LPT/A ORDERS/S 75/I CUSTOMERS 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 INV 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 INV INV$OUT 1:5/K 10:20/A 50:11/A
```

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

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

```
REORG/A INV$OUT INV$PRINT/L 2/S
```

This command creates the line sequential file INV$PRINT, 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 on the screen.

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 ITEMS$83 to the sequential file DESC:

```
REORG $LPT/A ITEMS$83 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.

Creating a Contiguous File

The following command copies the sequential file DESC to the relative file WELL:

```
REORG DESC/S WELL/R 25/I 0:10/C
```

The input record size is 25 bytes, which is also the output record size. The index portion of the file WELL is created as a 10-block contiguous file.

Transporting Files on Tape

REORG, with the utilities VINIT and VSTAT, prepares magnetic tapes for export and processes tapes imported from other installations. You can create and process magnetic tapes in many different formats, but REORG processes only fixed-length record formats. Several files can be placed on a single export tape. A large file can be divided among several tapes.

Import and export tapes can have ANSI standard labels, IBM standard labels (used with OS/VS or DOS), or no labels. Nonstandard labels are restricted to one data file on a single reel for import processing only.

The tape labels used on import and export tapes include a volume label, headers, end-of-volume labels and/or end-of-file labels. The auxiliary utility VINIT writes or verifies volume labels in ANSI and IBM formats, and erases all files from the tape. The auxiliary utility VSTAT reports the format and contents of a tape by reading its labels, without destroying any information.

For data safety, records are processed in tape blocks. If a transmission or media error renders a block unusable, only the records in that block are lost: the remainder of the tape is not compromised. The block length must be less than or equal to 8192 bytes. Records must be less than or equal to 4096 bytes and cannot span tape block boundaries.

Character Codes and Formats

ANSI and IBM are the two most common interchange formats for files on magnetic tape. The formats are very similar. In general, ANSI tapes are recorded in ASCII, and IBM tapes in EBCDIC. REORG can process tapes coded in either ASCII or EBCDIC. If several files are placed on an export tape, all are written with the same character code. Files on EBCDIC tapes are routinely translated to ASCII so they can be used by Interactive COBOL software. However, you may choose not to translate part or all of each input record.

If the files to be transported include records with computational fields, results are unpredictable. Use the /F switch (explained below) so that the computational fields are not translated to ASCII.

ANSI format has three levels of implementation; IBM has two. REORG currently reads or writes level 3 ANSI or level 2 IBM. The formats for both ANSI and IBM tape files are shown in Table 2-13. Table 2-14 to Table 2-18 give label block formats for ANSI and IBM labels.

Single File/Single Vol	Single File/Multivol	Multifile/Single Vol
VOL1	VOL1	VOL1
HDR1	HDR1	HDR1
HDR2	HDR2	HDR2
UHLa (ignored)	UHLa[2] (ignored)	UHLa (ignored)
TM[1]	TM	TM
Block 1 of file	1st part of data set	Block 1 of file
...	TM	...
Block n of file	EOV1	Block n of file
TM	EOV2	TM
EOF1	UTLa[3] (ignored)	EOF1
EOF2	TM	EOF2
UTLa (ignored)	TM (required for ANSI format)	UTLa (ignored)
TM	(Next volume:)	TM
TM	VOL1	HDR1
	HDR1	HDR2
	HDR2	UHLa (ignored)
	UHLa (ignored)	...
	TM	TM
	Last part of data set	TM
	TM	
	EOF1	
	EOF2	
	UTLa (ignored)	
	TM	
	TM	

[1]TM is tape mark (end-of-file).

[2]UHLa is the user header label, an optional label not processed by REORG.

[3]UTLa is the user trailer label, an optional label not processed by REORG.

Table 2-13 ANSI and IBM Tape File Formats

Field	Length	Name	Input	Output
1-4	4	Label ID	VOL1	VOL1
5-10	6	Volume ID	Audit	User
11	1	Accessibility		Space
12-37	26	Reserved		Spaces
38-51	14	Owner		User
52-79	28	Reserved		Spaces
80	1	Version		3

Table 2-14 ANSI Volume Label

Field	Length	Name	Input	Output
1-4	4	Label ID	VOL1[1]	VOL1
5-10	6	Volume serial	Audit	User
11	1	Reserved		0
12-41	30	Reserved		Spaces
42-51	10	Owner		User
52-80	29	Reserved		Spaces

[1]A block is accepted as a VOL1 block if it is at least 80 bytes long and the first 4 bytes are VOL1.

Table 2-15 IBM Volume Label

Field	Length	Name	Input	Output
1-4	4	Label ID[2]	HDR1	HDR1
			EOF1	EOF1
			EOV1	EOV1
5-21	17	File ID	Audit	Input filename
22-27	6	File set ID		Spaces
28-31	4	File section		0001
32-35	4	File sequence		0001
36-39	4	Generation no.		Spaces
40-41	2	Version no.		Spaces
42-47	6	Creation date		Date: byyddd
48-53	6	Expiration date		Date: b00000
54	1	Accessibility		Space (0 for IBM)
55-60	6	Block count	000000	000000 (HDR1)
			nnnnnn	nnnnnn (EOF1/EOV1)
61-73	12	System code		CS-40bbbbbbb
74-80	7	Reserved		Spaces

[1]IBM HDR1 and EOF1 labels have a format identical to ANSI, except for names given to various fields.

[2]A block is accepted as HDR1 (EOF1, EOV1) if it is at least 80 bytes long and the first 4 bytes match the label ID. On EOF1 blocks, the block count must equal the number of blocks read on the file.

Table 2-16 ANSI and IBM[1] HDR1, EOF1, EOV1 Labels

Field	Length	Name	Input	Output
1-4	4	Label ID	HDR2	HDR2
			EOF2	EOF2
			EOV2	EOV2
5	1	Record format	F	F
6-10	5	Block length	nnnnn	nnnnn
11-15	5	Record length	nnnnn	nnnnn
16-50	35	Reserved		Spaces
51-52	2	Buffer offset	00	00
53-80	28	Reserved		Spaces

Table 2-17 ANSI HDR2, EOF2, EOV2 Labels

Field	Length	Name	Input	Output
1-4	4	Label ID[1]	HDR2 EOF2 EOV2	HDR2 EOF2 EOV2
5	1	Record format[2]	F	F
6-10	5	Block length[3]	nnnnn	nnnnn
11-15	5	Record length	nnnnn	nnnnn
16-34	19	Reserved		Spaces
35-36	2	Recording		Spaces
37	1	Control char.		Spaces
38	1	Reserved		Spaces
39	1	Block attribute[4]	B or space	B
40-80	41	Reserved		Spaces

[1]A block is accepted as a HDR2 (EOF2, EOV2) block if it is at least 80 bytes long and matches the label ID. No further checking on input is done for an EOF2 or EOV2 label.

[2]Legal record formats are F (fixed), D (ANSI), V (variable), or U (undefined). Only F is currently permitted.

[3]Block length must be <=8192. Record length must be <=4096. IBM recording must be spaces (9-track tape).

[4]Legal block attributes are B (blocked), S (spanned), R (blocked and spanned), or space (not blocked or spanned). Only B and space are currently permitted.

Table 2-18 IBM HDR2, EOF2, EOV2 Labels

Initializing Export Tapes: VINIT

An export tape must be initialized to establish the interchange format — ASCII or EBCDIC — and to write header labels containing file and owner identification data. You can select full or partial initialization. Full initialization destroys all files on the tape and writes a new VOL1 label (specifying either ASCII or EBCDIC format) and includes a volume identifier and an owner identifier. Partial initialization reads the existing VOL1 label and reports its contents at the display terminal. The command for tape volume initialization is:

VINIT[/P] [auditfile/A] MT0[/I] volume-id [owner-id]

/P Indicates partial initialization. If omitted, full initialization is performed.

auditfile
Writes audit information to the specified file. If omitted, audit information is displayed at the master terminal.

/I Specifies EBCDIC format. If omitted, ASCII is assumed.

volume-id
Specifies volume ID, for full initialization only: 1-6 alphanumeric characters

owner-id
Specifies owner ID, for full initialization only: 1-14 alphanumeric characters (ASCII) or 1-10 alphanumeric characters (EBCDIC)

In full initialization, a volume label followed by a tape mark is written to the tape. In partial initialization, a volume label is read and audited, then a tape mark is written to the tape. If the label has been written at a density different from that set for the tape drive, VINIT fails and displays the error message TAPE ERROR — LOAD POINT.

On both input and output, the first block of the tape is read to verify that it is a VOL1 label. The format of this label is then used to determine whether processing continues in ANSI or IBM mode.

Reading Standard Tape Labels: VSTAT

The VSTAT utility reports essential statistics for an import or export tape with standard labels. In particular, VSTAT provides enough information about an "unknown" tape to build REORG commands that read the tape's files. The reporting process is fast, since VSTAT needs only to read the volume and file labels, not the files themselves. The sample report in Figure 2-18 illustrates the statistics that VSTAT generates:

```
TAPE STATISTICS UTILITY
DATE:                  04/11/83

VOLUME FORMAT:       IBM
VOLUME ID:       SALES29

FILE SEQUENCE NUMBER:    1

        FILE ID:            DAN
        CREATION DATE:      06/20/82
        RECORD LENGTH:      453
        BLOCK LENGTH:       454
        RECORD COUNT:       496
        BLOCK COUNT:        496
        FILE CONTINUED FROM PREVIOUS VOLUME

FILE SEQUENCE NUMBER:    2

        FILE ID:            BRUCE
        CREATION DATE:      09/01/82
        RECORD LENGTH:      453
        BLOCK LENGTH:       906
        RECORD COUNT:       538
        BLOCK COUNT:        269

END OF VOLUME
```

Figure 2-18 *Tape Label Information via VSTAT*

The VSTAT command syntax is as follows:

```
{ VSTAT auditfile/A }
{                   } MTO
{ VSTAT/A           }
```

The global switch /A tells VSTAT to send its report to the display terminal. Alternatively, the report is appended to the specified audit file. VSTAT creates this file if it does not already exist. To use a system printer as the audit file: specify $LPT (first printer) or $LPT1 (second printer).

If HDR2 is not present, VSTAT cannot report the record length, block length, record count, or block count. Instead it reports the other information and gives the following message:

```
FILE INFORMATION UNAVAILABLE -
FILE LABEL 2 NOT PRESENT IN LABEL BLOCK
```

VSTAT fails if the tape has a nonstandard label or is unlabeled. If the tape was written at a density different from that set for the tape drive, VSTAT fails and gives the error message TAPE ERROR--LOAD POINT.

Reading and Writing to Magnetic Tapes

Import and export tape processing with REORG is similar to disk file processing, but there are several arguments used only in tape processing. These arguments identify the magnetic tape as the input or output device, handle multivolume file and multifile volume situations, and indicate selective translation of fields from EBCDIC to ASCII coding.

When processing an import or export tape with standard labels, REORG verifies that the first block of the tape is a VOL1 label. This label tells REORG to write the tape using ASCII or EBCDIC format. REORG automatically writes the header blocks and end-of-file blocks required for the specified format.

If the tape drive is not on-line or is incorrectly selected, REORG issues a wait message. REORG releases the tape drive after it has read the file from the import tape. If the import tape was written at a density different from that set for the tape drive, the tape cannot be read. REORG issues the message TAPE ERROR--LOAD POINT.

Reading an Import Tape

To read a file from an import tape with standard labels, use VSTAT to obtain a listing of the tape's contents. Then issue a REORG command, using the format:

```
REORG [ { /A                } ]  MT0/T[E/N/U/B] out-file[/t] in-rec-size/I
      [ { auditfile/A       } ]
[tape-file[:tapemark-flag]/M] [volume-count/V] [out-rec-size/O]
[key-specifiers] [data-size:index-size/C] [field-pos:field-length/F]
[datafile-name/D]
```

Switches to the MT0 Specifier

/T	Indicates that a tape in ASCII format with standard labels
/T/E	Indicates a tape in EBCDIC format with standard labels
/T/N	Indicates a tape in ASCII format with nonstandard labels. REORG verifies that the volume label is nonstandard and begins processing at the first data record.
/T/N/E	Indicates a tape in EBCDIC format with nonstandard labels. REORG verifies that the volume label is nonstandard and begins processing at the first data record.
/T/U	Indicates an unlabeled tape in ASCII format
/T/U/E	Indicates an unlabeled tape in EBCDIC format
/T/B	Bypasses label verification. REORG processes the tape as unlabeled in ASCII format.

/T/B/E Bypasses label verification. REORG processes the tape as unlabeled in EBCDIC format.

The /B switch allows you to access a tape, even if the label is corrupted or if the tape origination site does not support the same level of standard label implementation.

Other Command Line Arguments

in-rec-size/I
The number of bytes in the input record. This number is identical to the PICs in the COBOL program that created the record. The input record size is required because the file on tape is sequential.

tape-file:tapemark-flag/M
Tape-file indicates the relative location of the desired data file on a multifile volume, i.e., 1, 2, 3, etc.

Tapemark-flag tells REORG whether or not to ignore a leading tape mark on an unlabeled tape. It may be set to 1 or 0. If the tapemark flag is 1, REORG ignores a leading tape mark. This guarantees that the first actual data file on the tape is counted as tape file 1. If the tapemark flag is 0, REORG reads the tape mark (if it exists) and identifies the first actual data file as tape file 2.

volume-count/V
The number of tape reels in a multivolume file. This argument must be specified for unlabeled tapes. (REORG can read the necessary volume count information from standard labels.)

out-rec-size/O
The number of bytes in the output record. Include this argument only if the output file is to be fixed sequential.

key-specifiers
If the original file input to tape was an ISAM file, the key specifiers must be included.

field-pos:field-length/F
Indicates that the field in an EBCDIC input file is not to be translated from EBCDIC to ASCII.

datafile-name/D
The filename in the HDR1 tape label. When processing a tape with standard labels, REORG compares this filename with the filename in the HDR1 tape label. If they do not match, REORG does not input the file.

The other command line options are the same as for disk file input, described above.

Examples
The following commands illustrate the use of REORG to read magnetic tape files:

```
REORG/A MTO/T/U DISKFILE/S 50/I
```

The /A switch displays audit information on the screen. The /T/U combination indicates that the input tape is in ASCII format and is unlabeled. DISKFILE, the output file, is fixed sequential. The input record size must be specified (50/I) because the input tape file is fixed sequential. (REORG can process only fixed sequential input files.)

```
REORG/A MTO/T/N/E DISKFILE/R 20/I
```

The /A switch displays audit information on the screen. The switch combination /T/N/E indicates that the input tape file is in EBCDIC format with nonstandard labels. Output file DISKFILE is relative, as shown by the /R switch. The input record size, required for all fixed sequential input files, is indicated by the /I switch.

```
REORG/A MTO/T DISKFILE/I 20/I 1:5/K
```

Audit information is to be displayed on the screen. The /T switch, when used alone after the MT0 specifier, indicates that the input file is in ASCII format with standard labels. Output file DISKFILE is indexed, as shown by the /I switch. The argument 20/I specifies the input record size, which is required since REORG processes only fixed sequential input files. The argument 1:5/K specifies the key position and the key length, which are required when the output file is indexed.

Writing to an Export Tape

REORG can create a new tape for export or append another file to an existing tape. All files on a single tape must have the same format (ASCII or EBCDIC) and must be written in the density of the receiving tape drive.

To create a new tape with standard labels, perform a full initialization with VINIT. If the file spans more than one tape volume, initialize enough tapes to accommodate the entire file. REORG pauses and tells you to mount a new tape as it fills each reel.

To create a new tape with no labels, perform a full initialization with the CLI command INIT/F. If the file spans more than one tape volume, initialize enough tapes to accommodate the entire file.

To append to an existing tape, do not use VINIT or INIT/F — they erase files. Use VSTAT to obtain a listing of the tape's contents. If the tape has a nonstandard label or is unlabeled, VSTAT fails. It is not necessary to know the number of files on the tape; REORG can output to the end of the tape and report the file's location in the audit file.

To create or append tape files, issue a REORG command using this format:

```
REORG [ { /A          } ] in-file[/t] MTO/T[E/N/U/B] [in-rec-size/I]
      [ { auditfile/A } ]
[out-rec-size/O] [blocking/B] [field-pos:field-length/F]
[tape-file[:tapemark-flag]/M]
```

Command Line Arguments

MTO
The use of the MT0 specifier and its various switches is the same as for tape import (see above).

in-rec-size/I
Include this argument only if the input file type is fixed sequential. Omit the input record size if the input file is an ISAM file.

out-rec-size/O
If the input file has variable-length records (a line sequential or variable-length sequential file), the output record size must be fixed either by specifying out-rec-size or field specifiers.

blocking/B
The number of records to be packed in a tape block. If omitted, REORG places one record in each block. The block size (*blocking* * *out-rec-size*) must not exceed 8192 bytes. The actual block size is always an even number; REORG rounds upward if the computed block size is odd.

tape-file:tapemark-flag/M
These specifiers are the same as for tape import (see above).

The other command line options are the same as for import tape (see above).

When the input file has variable-length records (a line sequential file or a variable-length sequential file), the output record size must be fixed, either by specifying *out-rec-size* or field specifiers. If the input file is line sequential, blank lines and form feeds are not written to the output file.

Recovering an ISAM File from Tape

When recovering an ISAM file, there may not be enough room on disk to accommodate the original .XD file in addition to the new .NX and .XD files. You can determine the number of blocks used and the number still available with the CLI command DISK. If the system response indicates that you do not have enough blocks, you follow this procedure:

1. Rename the .XD portion of the file to a name without an extension.
2. Delete the .NX portion of the file.
3. Use REORG to dump the renamed .XD file to tape. You can only dump the tape to MT0; other tape units cannot be specified. The command line is as follows:

   REORG [/A | auditfile/A] in-file/E MT0/T[E/N/U/B] in-rec-size/I
   out-rec-size/O [blocking/B] [field-pos:field-length/F]
   [tape-file[:tapemark-flag]/M]

   The input and output record sizes are required and should have the same values. They are calculated using the following formula:

   2 (number of keys in corrupted file) + 2 + number of bytes in data record

   If an ISAM file has two alternate keys and a record length of 80 bytes, the input and output record sizes would be calculated as follows:

   2(3) + 2 + 80 = 88

   Therefore, you would specify 88/I and 88/O for the input and output records, respectively.
4. Delete the corrupted file from the disk.
5. Use the tape as input to reconstruct the file on disk, using the following command line. Switches and arguments are defined below.

   REORG [/A | auditfile/A] MT0 { /F | /G } [/E/N/U]/X out-file { /R | /Q | /I }
   key-specifiers alt-key/N rec-length/L [num-recs/R]

/F	The input file is a tape file. Logically deleted records are to be physically deleted.
/G	The input file is a tape file. Logically deleted records are not to be physically deleted.
/E	The tape file is in EBCDIC format with standard labels.
/N	The tape file is in ASCII format with nonstandard labels.
/N/E	The tape file is in EBCDIC format with nonstandard labels.
/U	The tape is unlabeled and is in ASCII format.

/U/E The tape is unlabeled and is in EBCDIC format.

Note: If you do not specify the /E, /N, or /U switches, or any combination of these switches, REORG assumes that the tape is in ASCII format with standard labels.

/X The .XD header information is to be taken from the command line instead of the .XD header.

/I The output file is indexed (default).

/R The output file is relative. Records are renumbered, starting with 1.

/Q The output file is relative. The record numbers are preserved.

key-specifiers
The key specifiers are in the form *key-pos:key-length*, followed by /K for the primary key and /A for the alternate keys.

alt-key/N
The number of alternate keys in the file. If the file has no alternate keys, enter 0/N.

rec-length/L
The record length: the total number of characters in the PICs for that record, as defined in the original COBOL program.

num-recs/R
Indicates the number of records in the uncorrupt version of the file.

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.

BLOCK COUNT ERROR

BLOCK SIZE > 8192 CHARACTERS

BLOCKING SPECIFIED FOR NON-TAPE OUTPUT FILE

CHARACTER INSERTION OVERFLOW (> 33)

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

DIRECTORY ANNEX BLOCK ERROR

DUPLICATE KEY

DUPLICATE NON-TRANSLATION FIELD SPECIFIED

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.

FILE ID DOES NOT MATCH SPECIFIED FILE ID

FILE ID NOT SPECIFIED

FILE SEQUENCE ERROR

FILE SEQUENCE NO. OUT OF RANGE

INDEX DEPTH EXCEEDED

KEY NOT FOUND

KEY SIZE OUT OF RANGE

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 (/S) that is greater than 4.

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, /Q (output only), /R, /S, /V.

ILLEGAL FILE SEQ. NO. FOR LABEL TYPE

ILLEGAL INTEGER

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

ILLEGAL LABEL BLOCK - LABEL IDENTIFIER

ILLEGAL LABEL BLOCK - LESS THAN 80 CHARACTERS

A block must be at least 80 bytes long for REORG to accept it as a header, end-of-file, or end-of-volume block.

ILLEGAL LOCAL SWITCH

You have specified an incorrect local switch.

ILLEGAL SWITCH-TRANSLATION NOT NECESSARY

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 you are attempting to use REORG on a directory.

INPUT FILE NOT FOUND

You have tried to use REORG on 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.

INPUT TYPE MUST BE TAPE

ISAM FILE REVISION INCOMPATIBILITY

The ISAM revision number, which is stored in the .NX and .XD headers, is less than 5.00.

KEY LENGTH INCOMPATIBLE WITH EXISTING FILE

KEY SPECIFIED FOR NON-INDEXED OUTPUT FILE

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

LOCKED RECORD TABLE FULL

LOGICAL FILE STRUCTURE MAY BE CORRUPT. RUN ISAMVERIFY

A file's reliability flag is set, and you have attempted to use the file as an input ISAM file. If you have not run ISAMVERIFY, do so. If you are attempting to recover a corrupted file with REORG, consult the section on ISAM file recovery.

MOVES SPECIFIED

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

NO AUDIT FILE SPECIFIED

NO INPUT FILE SPECIFIED

You have not specified any files.

NO OUTPUT FILE SPECIFIED

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

NO TAPE DEVICE SPECIFIED

The tape device, MT0, was not specified.

NO VOLUME ID SPECIFIED

A volume ID is required when performing full initialization with VINIT.

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 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.

POSITION ERROR

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

RECORD FORMAT NOT FIXED

RECORD LOCKED

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 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 accommodate the record size.

RECORD SIZE OUT OF RANGE

RELATIVE KEY OVERFLOW

The relative key is larger than 65,535. You may have attempted to REORG a very large sequential file to relative format.

SPECIFIED FILE NOT FOUND ON TAPE

You have specified a file that REORG cannot find. Recheck the command line.

SPECIFIED KEY LENGTH OUT OF RANGE (1-100)

STANDARD LABEL FOUND - NOT EXPECTED WITH N OR U

REORG has determined that the label is standard, and you have specified /N (not standard labels) or the /U (unlabeled).

STANDARD VOLUME LABEL NOT FOUND

You have specified a standard label (/T or /E switch), which REORG cannot find.

TAPE ERROR - BAD TAPE

TAPE ERROR - END OF FILE

TAPE ERROR - END OF TAPE

TAPE ERROR - ODD CHARACTER COUNT

TAPE ERROR - PARITY

TAPE ERROR - WRITE LOCK

REORG cannot write to the file because the write lock is in place.

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 form lengths 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

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

```
ICX SETFORMS
```

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, QTY:3, 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 or the VERTICAL TABBING REQUIRED 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 6 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 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 and 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. You may press ESC at this point to exit 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 as many times as you wish by entering Y in response to 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; the utility informs you when it has finished. You may now print the special forms via your COBOL program. Figure 2-18 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 2-18 *Sample Program for Printing 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

The value for **NUMBER OF TABS** is greater than the maximum (6, 8, 12, or 16, depending on the length of the form and the number of lines per inch) or less than 1.

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 (RDOS)* 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 on page A-2 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
/L	Line sequential file type
/I	Indexed file type
/R	Relative file type
/T	Tape file
/Q	Relative output file. Preserve record numbers.
/T/U	Unlabeled tape in ASCII format
/T/N	Tape in ASCII format with nonstandard labels
/T/B	Bypass label verification. Treat as unlabeled ASCII format.
/F	Tape file: renamed .XD portion. Purge logically deleted records.
/G	Tape file: renamed .XD portion. Retain logically deleted records.
/D	Renamed .XD portion. Purge logically deleted records.
/D/X	Renamed .XD portion, with header information. Purge logically deleted records.
/E	Renamed .XD portion. Retain logically deleted records.
/E/X	Renamed .XD portion, with header information. Retain logically deleted records.

All switches in the body of the matrix are local switches. The switches in brackets are optional. Switches outside brackets are required to create the output file from the given input file. If no switches are shown or none appear outside brackets, no switches are required. An 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.

INPUT FILES	OUTPUT FILES: /S	/V or /L	/I	/R	/T	/Q
/S	/I [/O/F/P]	/I [/O/F/P]	/I/K [/A/O/F]	/I [/K/O/F]	/I [/B]	X
/V	/O [/F/P]	[/O/F/P]	/O /K [/A/F]	/O [/K/F]	[/B]	X
/R	[/O/F/P]	[/O/F/P]	/K [/A/O/F]	[/K/O/F]	[/B]	
/I	[/S/O/F/P]	[/S/O/F/P]	[/S/A/O]	[/S/K/O/F]	[/S/B]	X
/L	/O [/F /P]	[/O/F/P]	/O /K [/A]	/O [/K]	[/B]	X
/T	/I /D [/E/M/O/F/P]	/I /D [/E/M/O/F/P]	/I /D /K [/E/M/A]	/I /D [/E/M/K]	X	X
/T/U	/I [/E/V/M/O/F/P]	/I [/E/V/M/O/F/P]	/I /K [/E/V/M/A]	/I [/E/V/M/K]	X	X
/T/N	/I [/E/O/F/P]	/I [/E/O/F/P]	/I /K [/E/A]	/I [/E/K]	X	X
/T/B	/I [/E/O/F/P]	/I [/E/O/F/P]	/I/K [/E/A]	/I [/E/K]	X	X
/F	X	X	/N/L/K [/D/E/V/M/A]	/N/L [/D/E/V/M/K]		/N/L [/D/E/V/M]
/G	X	X	/N/L/K [/D/E/V/M/A]	/N/L [/D/E/V/M/K]	X	/N/L [/D/E/V/M]
/D	X	X	/K [/A]	[/K]	/I/O [/B/U/N]	
/D/X	X	X	/N/L/K [/A/R]	/N/L [/K/R]	X	/N/L
/E	X	X	/K [/A]	[/K]	/I/O [/B/U/N]	
/E/X	X	X	/N/L/K [/A /R]	/N/L [/K/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 (RDOS) **069-705014**
Supplies the programmer with information relating specifically to Interactive COBOL on the Real-Time Disk Operating System (RDOS). 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.

RDOS Documents

Introduction to RDOS **069-000002**
Introduces RDOS concepts to readers who are unfamiliar with the operating system and its capabilities. It also provides an index to relevant documentation.

Learning to Use the RDOS/DOS System **069-000022**
Serves as a working introduction to RDOS. It leads the user who has a basic understanding of RDOS through practice sessions with the command line interpreter, Supereditor, FORTRAN, BASIC, and assembly language.

How to Load and Generate Your RDOS System **093-000188**
Guides the reader step by step through the RDOS system generation process. The manual serves the first-time user and the user who wants to generate a system tailored to specific requirements.

RDOS/DOS Command Line Interpreter User's Manual **093-000109**
Introduces the command line interpreter (CLI) and describes its operations and advanced functions. It also highlights the features and operating procedures of the batch monitor.

RDOS Reference Manual **093-000075**

Provides detailed explanations of file access, memory use, system calls, multitasking, foreground, background, and multiprocessing. The manual is intended for assembly language programmers and experienced programmers using higher level languages.

RDOS/DOS User's Handbook **093-000105**

Summarizes the command line interpreter's commands and error messages, batch commands and error messages, RDOS/DOS calls, and RDOS utility program commands and error messages.

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

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

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

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

Index

reader comment form

Interactive COBOL Utilities (RDOS)

069-705020-01

Your comments will help us improve the quality of this publication. They will be carefully reviewed by the writers. Please refer to page numbers if appropriate.

DID YOU FIND THE MATERIAL:

	YES	NO		YES	NO
• Useful?	☐	☐	• Well illustrated?	☐	☐
• Complete?	☐	☐	• Well written?	☐	☐
• Accurate?	☐	☐	• Easy to read?	☐	☐
• Well organized?	☐	☐	• Easy to understand?	☐	☐

COMMENTS:

_____

_____

_____

_____

_____

_____

_____

_____

HOW DID YOU USE THIS PUBLICATION?

- ☐ As an introduction to the subject
- ☐ For information about operating procedures
- ☐ To instruct in a class
- ☐ As a student in a class
- ☐ As a reference manual
- ☐ Other *(please explain)*:

Name _____ Title _____

Firm _____ Date _____

Street _____ State _____

City _____ Zip _____

First fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 WESTBORO, MASS 01580

POSTAGE WILL BE PAID BY ADDRESSEE:

DataGeneral

ATTN: SDD Documentation
62 Alexander Drive
Research Triangle Park, NC 27709
USA

CUT ALONG DOTTED LINE

Second fold

069-705020-01

DATA GENERAL CORPORATION, Westboro, Massachusetts 01580