TOPS-10/TOPS-20

RMS Reference Manual

Version 1

CONTENTS

CONTENTS (CONT.)

FIGURES

TABLES

PREFACE

The TOPS-10/TOPS-20 RMS Reference Manual is:

- An introductory manual to illustrate basic file concepts and their application to Record Management Services (RMS).

- A reference manual to define the components of RMS, its relationship with user written programs (COBOL-74, BASIC+2), and descriptions of RMSUTL commands.

The information in this document is organized as follows:

- Chapter 1 provides an overview of RMS, its facilities, and its concepts.

- Chapter 2 provides a description of the properties of RMS files.

- Chapter 3 provides a description of the RMS file management utility, RMSUTL, and the use of this utility to create and access RMS files.

- Chapter 4 provides descriptions of the sections within RMS files, based on their organization.

- Chapter 5 provides descriptions of all RMS status codes.

- Appendix A provides the rules for using an RMS file from both BASIC-PLUS-2 and COBOL-74.

RMS provides record management for COBOL-74 (Version 12B) and BASIC-PLUS-2 (Version 2.1) (TOPS-20 only). COBOL-74 programmers should reference the COBOL-74 Language Reference Manual (AA-5059B-TK). BASIC-PLUS-2 programmers should reference the TOPS-20 BASIC-PLUS-2 Language Manual (AA-H654A-TM). Both of these manuals provide additional information on creating and maintaining RMS files.

If you find any errors in this manual, please write them on a separate sheet of paper and address it to:

DIGITAL EQUIPMENT CORPORATION
Software Documentation
200 Forest Street
MR1-2/L12
Marlborough, Massachusetts 01752

All reported errors will be corrected as soon as possible.

Conventions Used In This Manual

Symbol	Meaning
<RET>	Press the key labeled RETURN or CR.
<ESC>	Press the key labeled ESC, ESCAPE, ALT, or PRE.
<CTRL/C>	Press the keys labeled CTRL and C simultaneously.

CHAPTER 1

OVERVIEW OF RMS

1.1 OVERVIEW OF RECORD MANAGEMENT

As a user writing application programs, you need to create programs that will do some or all of the following:

1. Accept new input

2. Read or modify data

3. Produce output in some meaningful form

RMS provides generalized routines that are useful to you in writing such programs.

1.2 INTRODUCTION TO RMS

RMS is a file/record management system for TOPS-10/TOPS-20 (KL and KS) systems. It provides an interface between the operating system and user-developed application programs. User programs can be written in COBOL-74 or BASIC-PLUS-2 (TOPS-20 only).

The RMS functions available from COBOL-74 are only those that result from support of multikey indexed files in Level-2 ANSI COBOL-74. The COBOL compiler and OTS automatically do the RMS setup operations and calls needed to support the RMS-based language features. Thus, the primary reference document for COBOL programmers wishing to use multikey indexed files is the COBOL-74 Language Reference Manual. In particular, these users should see Appendix I, Using Multikey Indexed Files.

As with COBOL, BASIC and BASOTS automatically do the RMS setup operations and calls needed to support the RMS-based language features. Additionally all RMS file organizations are available through BASIC-PLUS-2.

RMSUTL is an interactive file management utility. With it, you can examine and patch your RMS files. You can also define a file's properties and create it using RMSUTL. (Refer to Chapter 3 for a detailed description of RMSUTL.)

1.3 RMS FACILITIES

RMS provides a wide range of facilities. The file operations it provides are:

1. Create file
2. Open existing file
3. Close file
4. Delete file
5. Truncate file (TOPS-20 only)

Stream files and record files are supported. Stream files are ASCII text files, with or without line-sequence numbers. Record files may be sequential, relative, or multikey indexed, and are described in more detail in Chapter 2.

The record operations RMS supports are:

1. Put (create) record
2. Find and get record
3. Find record
4. Update existing record
5. Delete record

The access methods by which you can find a record are:

1. Sequential
2. Relative record number
3. Random by key (a value in a record)

However, some operations and access methods cannot be applied to all file organizations (see Chapter 2).

1.4 RMS CONCEPTS

Access Mode -- The method in which data records in a file are retrieved. The access modes supported by RMS are sequential and random.

Area -- An area is a region of a file with a particular bucket size. Thus, all RMS files consist of (at least) one area. For indexed files, you may declare additional areas.

Bucket -- Data in an RMS file is organized into buckets. A bucket is the unit of physical I/O RMS uses. When RMS reads or writes a record, it actually reads or writes the bucket(s) containing the record. A bucket consists of one or more file pages (= 512 words). At present however, multipage buckets can only be declared for indexed files.

Entry -- The data objects in a bucket are called entries. Thus, records are one type of entry. Entries are numbered, and the entry with the lowest offset in a bucket is entry 1. Entry numbers are not names; if entry 3 is expunged from a bucket, then entry 4 "becomes" entry 3.

File Organization -- The physical arrangement of records in a file when it is created. File organizations can be sequential, relative, or multikey indexed.

ID -- A per-bucket unique ID is associated with each record in an indexed file. RMS assigns IDs in ascending sequence, starting with 1, as records are put in a bucket.

Prologue -- The beginning of an RMS record file is called its prologue section. It is the repository of the file's attributes.

Random Access -- An access mode in which the program-specified value of a key data item identifies the logical record that is to be accessed in a relative or indexed file.

RFA -- When RMS puts a record in a file, it assigns it a "Record File Address" (RFA). You may think of an RFA as the physical location of a record. RMS guarantees that no other record will be assigned this location while the file remains open.

Sequential Access -- With sequential access, you do not directly identify the record you wish to access. Instead RMS chooses the logical next record. Initially, when the file is opened, the current record is the first record in the file. Also, except for sequential files, it can be reset by a random access.

CHAPTER 2

PROPERTIES OF RMS RECORD FILES

TOPS-10/TOPS-20 RMS supports a variety of file organizations, record access modes, and record formats. The specific use of the file determines which file organization is best. The sections that follow describe the meanings of each of the above items.

Each file organization has unique properties. However, there are some properties common to all record files.

- o Sequential access is supported.
- o Records can be deleted and/or updated.
- o RMS assigns each record a record file address (RFA).
- o A header is stored with each record.
- o File has prologue.

2.1 SEQUENTIAL FILES

In sequential file organization, records appear in the order in which they were written to the file.

First Record	Second Record	Third Record	Fourth Record	Fifth Record	Sixth Record

Figure 2-1 Sequential File Organization

Thus, you can only add records to a sequential file at the current end of the file.

Only sequential access can be used in a sequential file, and physical adjacency establishes the order in which RMS reads data records. In particular, if you sequentially access a record in the file, the next record is the physically following record.

When you update an existing record in a sequential file, you cannot change the length of the record.

2.2 RELATIVE FILES

Relative file organization consists of a series of fixed-length positions (or cells) that are consecutively numbered from 1 to n. Position 1 is at the beginning of the file. Position 2 is next, and so on. The number of a record is called its relative record number.

```
                      Relative Record Numbers
                                 |
                                 |
           ---------------------------------------------------
           |         |         |         |         |         |
           |         |         |         |         |         |
           1         2         3         4         5         6
           |         |         |         |         |         |
           |         |         |         |         |         |
           V         V         V         V         V         V
      -------------------------------------------------------------
      |         |         |         |         |         |         |
      | Third   | Empty   | Second  | First   | Fourth  | Fifth   |
      | Record  | Cell    | Record  | Record  | Record  | Record  |
      | Written |         | Written | Written | Written | Written |
      |         |         |         |         |         |         |
      -------------------------------------------------------------
```

Figure 2-2 Relative File Organization

This method of file organization is available on disk drives only.

Although each record in a relative file is assigned to a fixed-length cell, the actual size of an individual record can be smaller than the cell size. Different size records can be in the same file.

Relative file organization allows sequential and random access of records. The relative record number of a record in a relative file is the key value in random-access mode.

2.2.1 Sequential Access To Relative Files

When you use sequential-access mode in a relative file, physical order establishes the order in which RMS reads and writes data records.

RMS recognizes whether record cells are empty or contain records. When a program issues read requests in sequential-access mode for a relative file, RMS searches successive cells until it finds one containing a record.

When a program adds new records in sequential-access mode to a relative file, the records are written to ascending relative cell numbers. Each write operation causes RMS to place a record in the cell whose record number is one higher than the record number of the previous write operation. If the cell already contains a record, RMS rejects the write operation and returns an error.

2.2.2 Random Access To Relative Files

In random-access mode, your program, not the file organization, determines the order in which record processing occurs. Each program request for a record must include the key value of the particular record to be accessed.

When you use random-access mode in a relative file, you specify the relative record number of the object record. If no record exists in the specified cell on a retrieval request, RMS returns an error indicator to the requesting program. Similarly, a program can randomly store records in a relative file by identifying the cell in the file that a record is to occupy (see Figure 2-2). If a write operation specifies the relative record number of a cell that contains a record, RMS returns an error indicator to the program.

One method of keeping track of each record's cell is to store records based on a numeric value within the record. For example, an account number could be equivalent to the relative record number.

2.3 INDEXED FILES

An indexed file is a file in which a record can be randomly located using arbitrary value(s) in the record. A portion of a record defined for this purpose is called a key. An indexed file is distinguished from a hashed-key file by the fact that the records are logically sorted by each key as well.

An RMS indexed file has at least one key, called the primary key. Optionally up to 255 "secondary" keys can also be defined. This is referred to as a multikey indexed file. You describe keys to RMS just once, when the file is created. RMS places the key descriptions in the file's prologue, and thereafter uses the stored key descriptions to support usage of the keys.

2.3.1 Sequential Access To Indexed Files

When you use sequential-access mode in an indexed file, the key you choose establishes the order in which RMS presents data records to your program. If a series of sequential reads is done, each successive record contains a value in the specified key field that is equal to or greater than that of the previous record. The chosen key is called the "key-of-reference".

When you write records in an indexed file using sequential-access mode, the primary keys in the records must be presented in ASCII ascending order. If the keys are not in ascending order, an error is returned to your program.

2.3.2 Key Access To Indexed Files

To do a key access, you specify a key of reference and a key value. When RMS finds a matching key value in the specified index, it locates the associated user data record and passes the record to the user program. If there are multiple records containing the same key value, the key access always finds the one that was written first. To access the other records with the duplicate key value, you must do sequential accesses.

In contrast to read requests, program requests to write records in an indexed file do not require the separate specification of a key value or index. This is because all the record's keys must be inserted in their respective indexes. In this way, RMS insures that the record can later be retrieved by any of its key values.

2.3.2.1 Key Matches - The key value specified on a key access can be tailored to different circumstances. RMS supports the following four types of key matches:

1. Exact key match
2. Approximate key match
3. Generic key match
4. Approximate generic key match

Exact key match means that RMS returns an error unless there is a record in the file whose key precisely matches the value specified in your program.

The approximate match facility allows your program to select either of the following two relationships between the key of the record retrieved and the key value specified by your program:

1. Equal to or greater than

2. Greater than

The advantage of the first kind of match is that if the specified key value does not exist in any record of the file, RMS returns the record that contains the next higher key value. The second kind of match specifies that the record with the next higher key value is returned even if a record with the specified key value exists. You would use this type of match if you were sequentially processing the file and wanted to bypass the records whose key values equalled the one you have in hand.

Generic key match means that the program needs only to specify an initial portion of the key value. RMS returns to your program the first occurrence of a record whose key value begins with what you specified.

Approximate generic key match combines the second and third techniques. For example, if "AAAAA" and "ZZZZZ" are the keys in a file, specifying "BB" will locate the record that has "ZZZZZ" as a key.

2.3.2.2 Key Content - A key value in a record is a specific series of bytes within the record. A key must have the same byte size as the file, and an indexed file may have byte sizes of 6, 7, or 9. Key location and byte size are both established when an indexed file is created.

The series of bytes comprising a key value may be composed of up to 8 key-segments and may be as long as 255 characters. Multi-segment keys are useful when you wish to sort the records of a file upon more than one logical field. For example, suppose you wish to periodically print a file of car records sorted by make, model, and year of make. This can be done by a sequential scan if you have defined a 3-segment key whose key-segments are respectively make, model, and year of make.

2.3.3 Indexes

Buckets in an indexed file are linked together into trees. Such a tree is called an index. RMS maintains an index for each key you define for an indexed file (see Figure 2-4).

Each level of an index contains entries sorted on the index's key, and buckets further from the root contain key values that are closer together. This increase in detail continues to the leaf level where every key value in the file appears (see Figure 2-3).

```
                         Root Bucket
                       (Primary Index)

               -------------------------------
               |         |         |         |
               |  ADAM   |  * * *  |  JASON  |
               |         |         |         |
               -------------------------------
               |                   |
               |                   |
----------------                  --
|                                 |
|   Data Bucket 1                 |   Data Bucket n
V                                 V
-------------------------------   -------------------------------
|         |         |         |   |         |         |         |
|  ARRON  | * * * * |  ADAM   |   |  JOHNS  | * * * * |  JASON  |
| WEST ST |         | ELM ST  |   | EAST ST |         | MAIN ST |
|  01347  |         |  21000  |   |  30306  |         |  45591  |
|         |         |         |   |         |         |         |
-------------------------------   -------------------------------

<-------------------------DATA RECORDS--------------------------->
```

Figure 2-3 Storage of Keys in an Indexed File

When you do a key access, RMS moves from the root towards the leaf level. RMS starts by scanning the root bucket for the first entry whose key value is greater than or equal to the key value you want to access. When RMS finds this entry, RMS continues the scan at the next level. This process continues until the scan reaches the data level. Thus, the cost of a key access is proportional to the number of levels in an index rather than the number of records in the file.

```
                         ------------                       <-----
                         |          |                            |
                         |   ROOT   |                            |
                         |          |                            |
                         ------------                            |
                               |                                 |
                               |                                 |
                               |                                 |
               --------------------------------                  |-Index
               |                              |                  | Buckets
               |                              |                  |
               |                              |                  |
       -----------------              -----------------          |
       |               |              |               |          |
       | Intermediate  |              | Intermediate  |          |
       |     Index     |              |     Index     |          |
       |    Bucket     |              |    Bucket     |          |
       |               |              |               |          |
       -----------------              -----------------     <-----
               |                              |
               |                              |
               |                              |
       -----------------               -----------------
       |               |               |               |
       |               |               |               |
       |               |               |               |
 -------------   -------------   -------------   -------------  <--
 |           |   |           |   |           |   |           |    |
 |   LEAF    |   |   LEAF    |   |   LEAF    |   |   LEAF    |    |-Data
 |           |   |           |   |           |   |           |    | Buckets
 -------------   -------------   -------------   -------------  <--
```

Figure 2-4   Buckets in Indexed Files

NOTE

The root of the tree is "up," and the leaves of the tree are "down."

There are three types of buckets in indexed files:

1. Index buckets are the buckets that constitute all but the leaves of an index.

2. Secondary-data buckets are the buckets that are leaves for secondary key indexes. Entries in these buckets are called SIDRs.

3. Primary-data buckets are the buckets that contain your records. Thus, they are the leaves of the primary key's index.

2.3.4 Format Of A Record File Address (RFA) In An Indexed File

The physical location of a record in an indexed file is specified relative to the bucket that contains it. Specifically, a record's RFA is the starting page of the record's bucket combined with its ID within the bucket.

An indexed file RFA is graphically represented as p1/i1, where p1 is the starting page of the record's bucket and i1 is the record's ID. For example, the first record put in a file is often assigned the RFA 2/1 (because page 0 is the prologue and page 1 is the root bucket of the primary index).

2.4 INDEXED FILE EFFICIENCY

2.4.1 Key Access Efficiency

The efficiency of a key access in an indexed file is a function of how many levels there are in the index. Thus, there is a large change in random access time when the number of levels changes. For a given number of records, the number of levels is a function of how full the buckets are and the number of entries that fit per bucket. Specifically, the number of records that fit in an index of n levels is described in the following section.

2.4.1.1 Determining Number Of Levels - You can determine the number of records that fit in an index of L levels by using the following formula:

```
DF/R * ((IF/K) **L)
```

where:

D = Words per data bucket - 3 words of overhead

F = Average fullness of bucket (for example, 1/2)

If you do a lot of random writes, you can expect an average on the order of half-full. If you mostly retrieve from a file loaded from a sorted sequential file, you can expect an average that is close to full. Characteristics between these extremes lead to fullness factors between half-full and full.

I = Words per index bucket - 3 words of overhead

K = Words per key + 1 word of overhead

L = Number of levels

R = Average number of words per record + O (overhead)

O = 2 (If the record format is fixed-length records)

O = 3 (If the record format is variable-length records)

NOTE

For a secondary key, R is approximately 2 + K + average number of write operations per record.

For example:

D = 509 (512 words/data bucket -3)

F = 0.8 (Average fullness of bucket)

I = 509 (512 words/index bucket -3)

K = 3 (for example, 10 character ASCII key plus overhead)

L = 2 (number of levels)

R = 50 (Average number of words/record plus overhead)

DF/R= 8 (Records/data bucket)

IF/K= 135 (Keys/index bucket)

MAX = 153000 (Maximum records that fit on 2 levels)

Thus, with 2 index levels, you can store 153000 records. If each bucket were half full (F = 0.5), the maximum number of records that will fit on 2 levels is 26244. If each bucket were full (F = 1), the maximum number of records that fit on 2 levels is 285610, over ten times greater than half full.

2.4.1.2 Shaping Your Index - Shaping your index effectively involves having certain data and making certain trade-offs. You need the following pieces of data:

1. The number of records that will eventually be in the file.

2. The number of records you plan to initially load.

3. The values of D, F, I, K, and R.

4. The frequency with which you do writes and deletes.

Aside from shrinking your records and keys, your primary tool for minimizing levels in an index is increasing F. You can improve F over time in two ways:

1. Sort the records in the initial load of the file, and

2. Periodically reload or UNCLUTTER a heavily updated file.

Bucket splitting occurs when you add more records than the bucket can hold. You can control bucket splitting in two ways:

1. During initial load of the file. Set data-fill and index-fill divided by bucket size equal to initial record count divided by eventual record count. Data-fill and index-fill are defined with the /DAFILL: and /IAFILL: switches in the RMSUTL DEFINE command. (Refer to Section 3.6.4 for a description of these two switches.)

2. Periodically UNCLUTTER or reload the file. The RMSUTL UNCLUTTER command is described in Section 3.6.13.

If your file is on a TOPS-10 and you pre-allocate the file, you can also make D or I greater than one page.

2.4.2 Write Access Efficiency

Write efficiency depends on three factors of decreasing importance.

1. The number of secondary keys. An additional secondary key linearly increases the cost of each write operation.

2. The number levels in an index. This has the same effect as for key accesses.

3. Likelihood of bucket splitting. A bucket split adds 2 bucket writes to a write operation.

CHAPTER 3

RMS FILE MANAGEMENT UTILITY

The RMS file management utility, RMSUTL, is an interactive utility for creating and manipulating RMS index files.

3.1 RULES OF OPERATION

To use RMSUTL, type RMSUTL (for TOPS-20) or R RMSUTL (for TOPS-10). RMSUTL responds with RMSUTL>, and you may now enter any of the commands described in section 3.6.

3.1.1 Command Format

Command formats consist of keywords, switches, names you have DEFINEd, file specifications, numbers, quoted strings, and guide words. Keywords and switches are names predefined by RMSUTL. Names you define may consist of letters, digits, hyphen, dollar-sign, period, and percent-sign, and may be as long as desired. However to avoid keyword conflicts, you should not create a name that is a prefix of BUCKET, DATAFIELD, FOUND-KEY-VALUE, HIGHEST, or PROLOGUE-DATA.

Command names and all other keywords need not be spelled out completely; any unique beginning is acceptable. For example, the DEFINE command can be DEF or DEFI. Commands are terminated by a carriage return - linefeed sequence.

See the TOPS-10 and TOPS-20 User Manuals for a description of file specification format. A quoted string is an arbitrary string of characters enclosed in double quotes. A guide word is a descriptive phrase enclosed in parentheses. You need never type in guide words. They are in RMSUTL to be typed out when you use recognition. You can type commands in either uppercase or lowercase. However, quoted strings are processed exactly as typed. Thus "ABC" is a different key value from "abc" (unless the current file is a SIXBIT file).

Type a ? to cause RMSUTL to display the list of available commands.

Question mark can also be used at any later point during command typein in order to see the alternatives available.

RMSUTL supports the usual input editing characters. Use ^U to delete the current command line. Use DELETE to backspace the cursor one position. Use ^R to redisplay the current command line. Use ^H to reestablish an aborted command line up to where the illegal typein occurred.

RMSUTL supports command recognition (use of ESCAPE), although not for file specifications. When you type ESCAPE, RMSUTL displays as much of the remainder of the command as it can. In particular, if you type the beginning characters of a keyword followed by ESCAPE, RMSUTL responds by typing the remainder of the keyword and any guide words following it. However, if you have not uniquely identified a keyword (for example, DE could be DELETE or DEFINE), RMSUTL sounds the bell.

3.1.2 Command Status

When necessary, RMSUTL displays errors, warnings, and informational messages.

An error message starts with ?, and there are the following cases.

1. ? message -- the command processor detected an input error, and command execution was not started.

2. ?UTLxxx -- RMSUTL detected an error during command execution. Usually this means that the whole command is cancelled. The exception occurs when the command performs more than one independent operation (for example, FIX RFA1, RFA2). In this case, only the operation affected by the error is cancelled; any preceding and succeeding operations are performed.

3. ? message / exit to EXEC -- an internal RMS error occurred. You should always report such errors to DIGITAL. You must rerun RMSUTL to perform more commands.

A warning message starts with % and indicates that something minor is in error, but RMSUTL is continuing execution of the operation. For example, you tried to name more than 16 datafields in a DISPLAY data-list command.

An informational message is enclosed in square brackets and is a comment about the command.

3.2 CREATING A FILE

The DEFINE FILE command enables you to create a file at a terminal rather than in a program. Unless you wish to create an indexed file, you simply give a single DEFINE FILE command, specifying the desired file attributes. If you wish to create an indexed file, you first DEFINE the areas and keys you wish to associate with the file. Then you give the DEFINE FILE command as before.

The following example creates an indexed file with a primary key and a secondary key, declares a large bucket size for the records of the file, and uses defaults for all other file attributes.

```
RMSUTL>DEFINE (OBJECT) AREA (NAMING IT) BIG (WITH BUCKET SIZE) 2 <RET>
RMSUTL>DEFINE (OBJECT) DATA (NAMING IT) EMPLOYEE-NAME (WITH TYPE) SIXBIT
       (WITH LENGTH) 30 (STARTING AT BYTE) 0 /DANAME:BIG <RET>
RMSUTL>DEFINE (OBJECT) DATA (NAMING IT) JOB-CLASS (WITH TYPE)
       SIXBIT (WITH LENGTH) 12 <RET>
RMSUTL>DEFINE (OBJECT) DATA (NAMING IT) SALARY (WITH TYPE) INTEGER <RET>
RMSUTL>DEFINE (OBJECT) KEY (NAMING IT) JCS (WITH SEGMENTS)
       JOB-CLASS,SALARY /CHANGES-ALLOWED /DUPLICATES-ALLOWED <RET>
RMSUTL>DEFINE (OBJECT) FILE (NAMING IT) EMPLOYEES.RMS
       (WITH ORGANIZATION) INDEXED (WITH KEYS) EMPLOYEE-NAME,JCS <RET>
```

3.3 FILE MANIPULATION

RMSUTL operates on one RMS file at a time. You establish this file with the OPEN RMS-FILE command. Similarly all output from the DISPLAY and scanning commands is written to a report file. The report file is an ASCII text file, and it is identified by the OPEN REPORT-FILE command. However, its use is optional. If no report file is open, data is output to your terminal.

When you are finished using a file, a CLOSE RMS-FILE command or CLOSE REPORT command must be done.

3.4 MANIPULATING DATA IN A FILE

RMSUTL enables you to symbolically manipulate control data and user data. It provides keywords for control field names and includes the DEFINE DATAFIELD command so you can name and describe your fields. The things you can do are DISPLAY fields, CHANGE fields, and DELETE entries.

You can access and change arbitrary fields in a prologue with DISPLAY PROLOG and CHANGE PROLOG. You can access, change, and expunge bucket entries with DISPLAY BUCKET, CHANGE BUCKET, and DELETE BUCKET. To use CHANGE BUCKET, CHANGE PROLOG, or DELETE BUCKET, you must open a file for PATCHING. The BUCKET and PROLOGUE options of the data manipulation commands constitute a physical view of the file. This view is normally needed only by system personnel. Its principle purpose is to facilitate diagnosing and patching corrupted files.

You can access, update, and delete records with DISPLAY data-list, CHANGE data-list, and DELETE RECORD. A data-list is one or more names created with the DEFINE DATAFIELD command. The record-level options of the data manipulation commands constitute a conceptual view of the file. They provide you at command-level with the same sort of capabilities you have in a program. Additionally, they allow you to scan a group of records and select all or some of them for processing. This control is provided by the records-to-use clause, which is described in section 3.7.

Another aspect of data manipulation is selecting what is to be processed. RMSUTL allows you to set:

- Current key tells RMSUTL which index to use when sequentially traversing the file.
- Current bucket tells RMSUTL which bucket to process when a CHANGE BUCKET, DISPLAY BUCKET, or DELETE BUCKET is done.
- Current record tells RMSUTL the base record that applies when relative record processing is done.

Currency indicators can be directly modified only by the SET command, which is described in section 3.6.11.

3.5 FILE SCANNING

Every file-scanning command directly accesses consecutive data buckets. These commands are designed so that it is easy to process an entire index or entire file in one command.

The file-scanning commands are SPACE, UNCLUTTER, and VERIFY. The VERIFY command is used to detect file corruption and out-of-sync-file control data. In update mode, it can be used to fix out-of-sync-file control data. The UNCLUTTER command is an updating VERIFY command that also removes deleted POINTER entries and deleted records from a file. The SPACE command simply reports on space usage in a file: how full buckets are and how much clutter there is in them. Output from each of these commands goes to the report file.

The OPEN RMS-FILE command for OUTPUT or PATCHING causes the file to be opened for exclusive update. Thus, an UNCLUTTER command or updating VERIFY command cannot be done while you are up for production. However, the impact of these restrictions has been minimized. First of all, you can do a read-only VERIFY. If any out-of-sync data is detected, you can then reopen the RMS file at a more convenient time and quickly correct the affected entries with the FIX command.

Additionally, a read-only VERIFY can be done from multiple jobs at the same time. This can be done because it is easy to specify mutually exclusive portions of the file within each job. Similarly, one can UNCLUTTER a file over a period of several runs.

3.6 COMMAND DESCRIPTIONS

The following conventions are used in the command descriptions that follow. Switches are shown alphabetically, but zero or more may be specified in any order. The symbol (|) represents a choice in arguments, parameters, or switches that can be used in a command. For example,

```
          /DISPLAY:| OCTAL   |
                   | DECIMAL |
```

means you can specify /DISPLAY:OCTAL or /DISPLAY:DECIMAL. If there are multiple options in a command that start with the same phrase, the guide words for the phrase are shown only with the first command option. Also guide words are shown for clarity only; it is never necessary to type in guide words.

The RMSUTL commands are:

- o CHANGE — changes the specified fields to the specified values
- o CLOSE — closes the currently opened RMS or REPORT file
- o DEFINE
  - AREA — creates and names an area description
  - DATA — defines data fields in a record
  - FILE — creates an RMS file with specified attributes
  - KEY — creates and names a key description
- o DELETE — deletes the specified entry or record
- o DISPLAY — outputs the specified fields to the report file
- o EXIT — returns to operating system command level (you can CONTINUE)
- o HELP — outputs a brief description of RMSUTL
- o INFORMATION — displays the specified portion of the current environment
- o OPEN — opens the specified RMS or REPORT file
- o REDEFINE — gives new attributes to a DEFINEd name

- o SET changes the currency indicators
- o SPACE outputs space usage statistics for the specified part of a file
- o TAKE executes the RMSUTL commands in the specified file
- o UNCLUTTER eliminates deleted POINTER entries and deleted records from a file
- o VERIFY determines if a file is internally consistent

3.6.1 CHANGE Command

Function

This command enables you to alter entries in your file. The file is checkpointed after an entry is changed.

Caution should be applied when using the CHANGE BUCKET and CHANGE PROLOGUE commands. Their purpose is to provide you with a symbolic, and therefore safer, way of correcting damaged control data. Conversely, they give you the power to damage a perfectly valid file.

Formats

The "to-list" argument in the formats below is of the form:

field-name (TO) value, field-name (TO) value, ... , etc.

```
                                        | FILE    |
CHANGE (VALUE OF) PROLOGUE-DATA (FOR) | AREA n1 | (SETTING) to-list
                                        | KEY  n2 |
```

Fields in the specified descriptor are altered. The RMS file must be open for patching.

N1 is an area number. N2 is a key of reference. If FILE is specified, the allowed fields are:

1. AREA-COUNT
2. AREA-OFFSET
3. BUCKET-SIZE
4. BYTE-SIZE
5. FORMAT
6. KEY-COUNT
7. KEY-OFFSET
8. MAXIMUM-RECORD-SIZE

9. ORGANIZATION

10. PAGES-IN-FILE

If KEY n2 is specified, the allowed fields are LEVELS, NEXT-KEY, ROOT-PAGE, and all the field-names of a KEY XAB.

The internal tables of RMS do not reflect the effects of a CHANGE PROLOG command until after the file is closed and reopened. Therefore RMSUTL does not allow you to perform any bucket or record operations until you reopen the file.

```
                                | ENTRY  n1 |
CHANGE (VALUE OF) BUCKET (AT) | HEADER    | (SETTING) to-list
                                | ID     n2 |
```

RMSUTL changes the specified entry within the current bucket. The RMS file must be open for patching. An entry can be identified by entry number or ID number. The field-names allowed in to-list are bucket and entry type dependent, see section 4.3.

If a RECORD entry of a primary-data bucket is specified, the to-list can also contain datafield names. This usage differs from CHANGE datafield-list in that there are no side effects or checks when a key value is changed.

RMSUTL changes the indicated fields in the header of the current bucket. The field-names allowed in to-list are:

1. AREA-NUMBER

2. ATTRIBUTES

3. LAST-ID

4. LEVEL

5. NEXT-BUCKET

6. NEXT-ID

7. TYPE

8. WORDS-IN-USE

CHANGE (VALUE OF) to-list records-to-use

The indicated datafields in the identified records are changed. At most 16 datafields can be specified. If the records-to-use clause is omitted, the current record is used. The file must be open for output or patching.

RMSUTL performs an RMS update operation (COBOL REWRITE verb, BASIC+2 UPDATE statement). Therefore, you have no more power than if you did the operation in a program. For example, if you tried to change the value of the primary key, RMS would not perform the update. See Section 3.7 for a description of the "records-to-use" clauses.

3.6.2 CLOSE Command

Function

This command closes the specified type of file, thereby permitting you to open another file of that type.

Format

```
CLOSE (FILE TYPE) | REPORT   |
                  | RMS-FILE |
```

This command closes either the currently opened report file or the currently opened RMS file.

3.6.3 DEFINE DATAFIELD Command

Function

The DEFINE DATAFIELD command is used to associate a name and data type with a consecutive group of bytes in a record. RMSUTL then can display or modify such a data segment when given its name. Additionally, you can place all your field definitions in a file and thereafter TAKE them whenever you run RMSUTL.

The DEFINE DATAFIELD command can be used to define a single segment key. You accomplish this by suffixing /KEY or any of the switches described for DEFINE KEY (see Section 3.6.4).

Formats

```
DEFINE (OBJECT) DATAFIELD (NAMING IT) user-name (WITH TYPE)

     | ASCII      |
     | FILE-BYTES | (WITH LENGTH) n1 (STARTING AT BYTE) n2
     | SIXBIT     |
```

This usage defines a string datafield with the specified internal representation. FILE-BYTES indicates that the representation is taken from the file byte size. N1 indicates the number of bytes in the datafield. N2 indicates the number of bytes preceding the starting byte of the datafield. If n2 is omitted, RMSUTL places the datafield immediately past the last defined datafield. However default positioning is allowed only when the last defined string has the same internal representation. The initial positioning defaults are ASCII at byte 0.

```
DEFINE (OBJECT) DATAFIELD (NAMING IT) user-name | FLOATING |
                                                | INTEGER  |

     (STARTING AT WORD) n3
```

FLOATING defines a numeric datafield that is represented internally as a single-precision floating-point number. N3 indicates the number of words preceding the datafield. If n3 is omitted, RMSUTL places the datafield at the first word boundary past the last defined datafield.

INTEGER defines a numeric datafield that is represented internally as a single-precision binary integer. N3 indicates the number of words preceding the datafield. If n3 is omitted, RMSUTL places the datafield at the first word boundary past the last defined datafield.

Switch

```
/DISPLAY: | OCTAL   |
          | DECIMAL |
```

The /DISPLAY: switch is applicable only when you DEFINE a DATAFIELD to be INTEGER. If OCTAL is specified, the field is DISPLAYed as an unsigned octal value. If DECIMAL is specified, the field is DISPLAYed as a signed decimal value. If /DISPLAY is not specified, DECIMAL is assumed.

3.6.4 Defining File Attributes And The DEFINE FILE Command

Function

The DEFINE FILE command is used to create an RMS file and assign it attributes. The DEFINE FILE command is the only way to make an indexed file contain multiple areas, allow duplicates for the primary key, or be loaded with partially full buckets (see Appendix A).

For indexed files, you must define at least one key field. You may also define one or more areas. An area must be DEFINEd before it is used in a DEFINE KEY or DEFINE DATAFIELD command. Similarly, keys must be DEFINEd before they are used in a DEFINE FILE command.

Format I

DEFINE (OBJECT) FILE (NAMING IT) filename (WITH ORGANIZATION)

INDEXED (WITH KEYS) key-name-list

Key-name-list is one or more key-names you have DEFINEd. It identifies the file's primary key, first secondary key, second secondary key, and so on. No segment of a key can have a string data type that disagrees with the file's byte size.

Switches

/BUCKET-SIZE:n

RMSUTL sets bucket size for the default area to n pages. If this switch is omitted, a bucket size of 1 page is assumed.

/BYTE-SIZE:n

RMSUTL sets the file's byte size to n bits. N can be 6, 7, or 9. If this switch is not specified, a byte size of 7 is assumed.

/RECORD-SIZE:n

N is a byte count. If record format is fixed, RMS uses n as the record size for all records in the file. Thus, this switch is required if record format is fixed. If record format is variable, RMS interprets n as the maximum record size. If this switch is not specified for a variable-record file, no maximum is established.

```
/FORMAT: | FIXED    |
         | VARIABLE |
```

RMSUTL gives the file the indicated record format. VARIABLE means that the file can contain records of different lengths. FIXED indicates that each record must have the size specified in the /RECORD-SIZE switch. If this switch is not specified, VARIABLE is assumed.

Format II

DEFINE (OBJECT) KEY (NAMING IT) key-name (WITH SEGMENTS) segment-name

This command creates a key description with the specified name and attributes. Segment-name identifies the datafields that comprise the key, up to eight can be specified. The datafields specified must already have been DEFINEd. The combined size of a key's segments may not exceed 255 bytes. The data type of a string segment must conform to the file's byte size. For example, a segment's data type can be SIXBIT only if the file's byte size is 6.

Switches

/CHANGES-ALLOWED

If specified, a program will be able to alter this key on an update. This switch can not be applied to a primary key.

/DANAME:area-name

Data buckets for this key will be allocated from the specified area, which must already have been DEFINEd. If this switch is not specified, area 0 is used.

/DAFILL:n

RMS will put at most n words in a data bucket for this key. This enables you to spread out records when initially loading a file. If this switch is not specified, n is set to 0 which means the entire bucket is used.

/DUPLICATES-ALLOWED

If specified, a program will be able to write a record that contains a key value that already exists in the file.

/IANAME:area-name

Index buckets for this key will be allocated from the specified area, which must already have been DEFINEd. If this switch is not specified, area 0 is used.

/IAFILL:n

RMS will put at most n words in an index bucket for this key. This enables you to spread out bucket entries when initially loading a file. If this switch is not specified, n is set to 0 which means the entire bucket is used.

Format III

DEFINE (OBJECT) AREA (NAMING IT) area-name (WITH BUCKET SIZE) n

This command creates an area description with the specified name and bucket size. N is the number of pages to be used for buckets of this area. An area-name can be referenced in 0 or more key descriptions.

3.6.5 DELETE Command

Function

This command is used to reclaim space in a file or to logically delete records.

Caution should be applied when using DELETE BUCKET. Its purpose is to provide you with a symbolic, and therefore safer, way of correcting damaged control data. Therefore, it gives you the power to damage a perfectly valid file.

Formats

```
DELETE (OBJECT) BUCKET-ENTRY (IDENTIFIED BY) | ENTRY n1 |
                                             | ID    n2 |
```

The space occupied by the specified entry in the current bucket is reclaimed. ID n2 may be specified only for a data bucket. The RMS file must be open for patching.

RMSUTL tries to prevent accidental deletions. A SIDR entry space may not be reclaimed if any non-NIL values are in its RFA list. An index entry's space may not be reclaimed if its DOWN-POINTER identifies a valid nonempty bucket. A RECORD entrys space may not be reclaimed unless its DELETED attribute is on.

RECORD entries are special in that they are pointed at by other entries. If a RECORD entry points back to a POINTER entry, the POINTER entry is reclaimed as well. However, deletion of a RECORD entry does not affect any secondary data entries associated with the record.

```
                                       | KEY              |
DELETE (OBJECT) RECORD (IDENTIFIED BY) | LAST-ENTRY       |
                                       | RELATIVE-REC-NO  |
```

The records identified by KEY, LAST-ENTRY, or RELATIVE-REC-NO clause are deleted. These clauses are referred to as "records-to-use" clauses. If you do not specify one of these records-to-use clauses, only the current record is deleted. A delete operation in a program would have the exact same effect. See Section 3.7 for a description of the "records-to-use" clauses.

3.6.6 DISPLAY Command

Function

This command outputs values to the report file, which you identify with an OPEN REPORT command. However, if a report file is not open, the report is output to the user terminal. The data in the report is discussed in Section 3.8.1.

Formats

```
                                       | AREA n1            |
DISPLAY (VALUE OF) PROLOGUE-DATA (FOR) | FILE field-list    |
                                       | KEY n2 field-list  |
```

If the (FOR) clause is omitted, the entire prologue is displayed. If a field-list is omitted, the entire file, area, or key descriptor is displayed. N1 is an area number. N2 is a key of reference.

If FILE is specified, the allowed fields are:

1. AREA-COUNT
2. AREA-OFFSET
3. BUCKET-SIZE
4. BYTE-SIZE
5. KEY-COUNT
6. KEY-OFFSET
7. MAXIMUM-RECORD-SIZE
8. ORGANIZATION
9. PAGES-IN-FILE
10. RECORD-FORMAT

If KEY n2 is specified, the allowed fields are LEVELS, NEXT-KEY, ROOT-PAGE, and all the field-names of a KEY XAB (extended argument blocks).

```
                                    | ENTRY n-list       |
                                    | HEADER             |
DISPLAY (VALUE OF) BUCKET (AT)      | ID n-list          |
                                    | KEY-VALUE "string" |
                                    | LARGEST-ENTRY      |
```

The specified clause of the current bucket is displayed. A bucket consists of a header and 0 or more entries. If none of the above clauses are specified in the DISPLAY command, the entire bucket is displayed. The above clauses are defined as follows:

ENTRY n-list the specified entries are displayed.

HEADER the bucket's header is displayed.

ID n-list the entries with the specified IDs are displayed. The current bucket must be a data bucket.

KEY-VALUE "string"

the first entry with a key greater than or equal to "string" is displayed.

LARGEST-ENTRY the largest entry in the bucket is displayed.

N-list is a list of ranges separated by commas. A range is n1 (TO) n2. If n2 is omitted, n1 (TO) n1 is assumed. For example, DISPLAY BUCKET ENTRY 1 (TO) 3, 7 displays entry 1, entry 2, entry 3, and entry 7. A range may be sparse. For example, if no record in the current bucket has ID 4, DISPLAY BUCKET ID 3 (TO) 5 displays the other two entries regardless.

```
                                                    | KEY              |
DISPLAY (VALUE OF) DATA (FOR RECORDS IDENTIFIED BY) | LAST-ENTRY       |
                                                    | RELATIVE-REC-NO  |
```

The portion of the identified records indicated by the KEY, LAST-ENTRY, or RELATIVE-REC-NO clause is displayed. These clauses are referred to as "records-to-use" clauses. The entire record is treated as a field whose type is FILE-BYTES. A datafield-list is one or more datafield names separated by commas. If a datafield-list is specified, each field is displayed in accordance with its DEFINE command. A maximum of 16 datafields can be specified. If the above clauses are omitted, the current record is used. See section 3.7 for a description of the "records-to-use" clause.

3.6.7 FIX Command

Function

This command completes an aborted delete, put, or update for the specified entries. If the entry needs fixing, the same message as would occur during a VERIFY is output to the report file. No message is generated when you give a FIX command for an already valid entry.

FIX is used in conjunction with VERIFY. When VERIFY is run in NOFIX mode, it denotes inconsistencies that can be fixed later by suffixing [Fixable] to the message describing the inconsistency. See section 3.8.2 for a description of these messages and the exact FIX statement that applies to each of them.

Format

FIX (RECORD WITH RFA) fix-list

An entry in a fix-list is of the form "rfa1 (OF INDEX) n". If n is omitted, 0 is assumed. If an entry in a fix-list is invalid in some way, processing of subsequent entries in the list is not affected.

3.6.8 INFORMATION Command

Function

The INFORMATION command displays on your terminal the specified information, dependinng on the clause you use, the current status of RMSUTL, and the file you are maintaining.

Format

```
                         | ALL        |
                         | AREAS      |
INFORMATION (ABOUT)      | CONTEXT    |
                         | DATAFIELDS |
                         | KEYS       |
```

Specifying AREA, DATAFIELDS, or KEYS causes RMSUTL to display the attributes of the indicated DEFINEd names.

CONTEXT displays information about the currency indicators. It displays the current key of reference, the current record's RFA, and the current bucket's page number. It also tells you about the report file and RMS file.

ALL combines the output of the other options.

3.6.9 OPEN Command

Function

The OPEN command identifies and makes a file available to RMSUTL. There are two types of files, RMS files and report files. All data manipulation is with respect to the currently open RMS file. The output of DISPLAY and the file-scanning commands is directed to the currently open report file, if there is one. If no report file is open, reports are directed to TTY:. Until a file is CLOSEd, a new file of the same type cannot be identified.

Format I

```
                                                    | INPUT    |
OPEN (FILE TYPE) RMS-FILE (WITH NAME) filespec (FOR) | OUTPUT   |
                                                    | PATCHING |
```

Opens the specified RMS file for the indicated type of access. The default access mode is INPUT. CHANGE, DELETE, and UNCLUTTER may not be done if the file is open for input. CHANGE PROLOGUE, CHANGE BUCKET, and DELETE BUCKET may not be done unless the file is open for PATCHING.

If OUTPUT or PATCHING is specified, no one else can have the file open to write. If there is an access conflict with another user, the OPEN fails. If INPUT is specified, an access failure cannot occur. However there is a small chance that a later command may be affected by the output of another job.

OPEN RMS-FILE is a two operation command: OPEN followed by SET INDEX 0. If the second operation fails, the file "remains" open, with current bucket set to 1 and no current record set.

Switch

/MAXIMUM-RECORD-SIZE:n

If a file was created without a maximum record size, RMSUTL has no direction as to an appropriate size for its data record buffer. This switch tells RMSUTL to allocate a buffer of n file-bytes. If the switch is also omitted, RMSUTL allocates a buffer of 512 words.

Format II

OPEN (FILE TYPE) REPORT /APPEND

The OPEN REPORT command opens the specified file for output, creating it if necessary. If the file already exists, it is superseded unless the /APPEND switch is specified. The report file is a stream ASCII file and is checkpointed at the end of each RMSUTL command.

Switch

/APPEND

If a report file already exists, the /APPEND switch permits you to add stream ASCII data to the existing file. The existing report file will not be superseded.

3.6.10 REDEFINE Command

Function

This command gives new attributes to an already DEFINEd name.

Formats

```
                     | datafield |
REDEFINE (NAME)      | area      | remainder of DEFINE
                     | key       |
```

See section 3.6.3 for a description of DEFINE DATAFIELD. See section 3.6.4 for descriptions of DEFINE AREA and DEFINE KEY.

3.6.11 SET Command

Function

This command enables you to modify the currency indicators. The general rules governing currency indicators are as follows:

1. When you open an RMS file, RMSUTL simulates a SET INDEX 0.

2. If a SET command results in a ? message, the currency indicators are left unchanged.

3. The SET INDEX command sets all three currency indicators.

4. The SET BUCKET command sets only the current bucket.

5. The SET RECORD command sets just the current record unless you specify SET RECORD KEY new-key-of reference. In this case, it sets all three currency indicators.

Formats

SET (CURRENT) BUCKET (TO ONE IDENTIFIED BY) bucket-to-use

This command sets the current bucket to the one specified in the bucket-to-use clause. The bucket-to-use clause may be one of the following:

- DATA-LEVEL locate the leftmost data bucket under the current bucket. Leftmost means the data bucket with the lowest keys.

- DOWN n move to the bucket pointed to by the nth entry from the top of the current bucket. If n is omitted, move to the bucket pointed to by the first entry. The current bucket must be an index bucket.

- LAST-RECORD

  move to the bucket associated with the last record selected in a record-processing command. The record-processing commands are CHANGE data-list, DELETE RECORD, DISPLAY data-list, and SET RECORD. If the primary key is current, the primary data bucket containing the record is selected. If a secondary key is current, the secondary data bucket containing the SIDR that points at the record is selected.

- o NEXT — move to the bucket at the same level of the tree with the next higher group of key values. However, if the current bucket is rightmost, move to the leftmost bucket.

- o ROOT — position to the root bucket of the current index structure.

- o UP — move to the bucket whose entry points at the current bucket.

SET (CURRENT) INDEX (USING KEY OF REFERENCE) n1 (AND) where-in-index

This command sets the current key of reference to n1. It sets current bucket and current record using the where-in-index clause. The possibilities are:

BUCKET (AT PAGE) n2

RFA (BKT/ID) n3/n4

ROOT

If the where-in-index clause is omitted, ROOT is assumed.

ROOT causes the current bucket to be set to the root bucket of index n1. It causes the current record to be set to the record containing the lowest key value for key n1.

BUCKET n2 sets the current bucket to n2. If this bucket is not part of index n1, RMSUTL may not be able to tell for sure. If [Page not start of ...] is displayed, the bucket is still made current because the problem may be that the bucket's header is clobbered rather than that you specified a bad page number. RMSUTL tries to set the current record to the record identified by the first entry in the selected bucket. However, RMSUTL cannot always do this. The full set of rules are:

1. If the bucket's header is clobbered, do not set a current record.

2. If the bucket at page n2 is not a data bucket, internally perform a SET BUCKET DATA-LEVEL.

3. If the bucket at the data-level is empty, do not set a current record.

4. For key 0, try to set current record to the first entry in this bucket.

5. For a secondary key, try to set current record to the first RFA in the first entry's RFA list. If the list is empty, do not set a current record.

6. Do a key access using the key value in the tentative current record. If the key access fails, do not set a current record.

7. Scan up to 100 duplicates until the tentative current record is found. If it is found, it has been made current. If it cannot be found (probably because it is a deleted entry or it is more than the 100th duplicate), "approximate" and set current record to the record located by the key access.

RFA n3/n4 tries to set the current record to the record located by ID n4 in the primary data bucket at page n3. The record or its POINTER entry may be at n3/n4. The operation may fail or become approximate in the same ways as BUCKET n2. For key 0 current bucket is set to the bucket containing the current record. For a secondary key, it is set to the bucket containing the current record's SIDR.

SET (CURRENT) RECORD (TO FIRST ONE IDENTIFIED BY) records-to-use

This command sets the current record to the first one selected by the records-to-use clause, which is described in section 3.7. If a new key of reference is specified in the records-to-use clause, it becomes current, and the current bucket is set to the data bucket containing the data record (or SIDR) selected.

3.6.12 SPACE Command

Function

This command accumulates space usage statistics for the data buckets in the specified key range. SPACE outputs status information and the results of its scan to the report file. This output is described in section 3.8.2. The fullness and clutter statistics it generates are a good indication of when to UNCLUTTER or reload a file.

Format

```
                             | ALL-KEYS           |
SPACE (USAGE OF FILE FROM)   | KEY (#) n1 range   |
                             | SECONDARY-KEYS     |
```

If ALL-KEYS is specified, all indexes of the file are scanned. If SECONDARY-KEYS is specified, the index of each secondary key is scanned. If KEY n1 is specified, the specified part of that key's data buckets is scanned. If a range is not specified, the entire index is scanned. If none of the above options are specified, ALL-KEYS is assumed.

Range is of the form (FROM) low-bound (TO) high-bound. Low-bound may be LOWEST or a quoted string. High-bound may be HIGHEST or a quoted string. The interpretation of a scanning range is consistent with the records-to-use clause, described in section 3.7.

3.6.13 UNCLUTTER Command

Function

This command eliminates POINTER entries and expunges DELETEd records within the specified range of keys. In the process of doing this, it does an updating VERIFY KEY 0 for the specified range. UNCLUTTER outputs status information and the results of its scan to the report file. This output is described in section 3.8.2.

Periodic use of this command reduces degradation of keyed accesses over time. There are two reasons for this. UNCLUTTER, by reclaiming space, reduces the likelihood of bucket splitting. Secondly, removing POINTER entries saves the file access needed to process the level of indirection POINTER entries imply.

Format

UNCLUTTER (INDEX FILE FROM) range

Range is of the form low-bound (TO) high-bound. Low-bound may be LOWEST or a quoted string. High-bound may be HIGHEST or a quoted string. If the range is omitted, LOWEST (TO) HIGHEST is assumed. The interpretation of a scanning range is consistent with the records-to-use clause, described in section 3.7.

Switch

/PROGRESS:n2

If the /PROGRESS switch is specified, RMSUTL outputs the UNCLUTTER's progress every n2 keys it scans. Progress is shown by displaying the highest key value scanned so far. If /PROGRESS is omitted, 10000 is assumed. RMSUTL also checkpoints the RMS file and the report file when it makes a progress report.

3.6.14 VERIFY Command

Function

This command verifies that records in the specified key range can be accessed sequentially and by key. VERIFY outputs status information and the results of its scan to the report file. This output is described in section 3.8.2. If the RMS file is open for output or patching, VERIFY will complete aborted deletes, puts, and updates, unless you specify otherwise.

As VERIFY scans an index structure, it does the following:

1. It verifies that the keys of each bucket are sorted in ascending sequence. And for secondary keys, it checks that each RFA in a SIDR points to a data record with the same key value as the SIDR.

2. It verifies that duplicates occur only when proper.

3. It verifies that each entry scanned can be accessed by key (by doing a key access for the highest key in each data bucket).

4. It checks the bucket header of each bucket it accesses.

Format

```
                              | ALL-KEYS           |
VERIFY (INDEX FILE UPON)      | KEY (#) n1 range   |
                              | SECONDARY-KEYS     |
```

If ALL-KEYS is specified, all indexes of the file are scanned. If SECONDARY-KEYS is specified, the index of each secondary key is scanned. If KEY n1 is specified, the specified part of that key's index is scanned. If a range is not specified, the entire index is scanned. If none of the above options are specified, ALL-KEYS is assumed.

Range is of the form (FROM) low-bound (TO) high-bound. Low-bound may be LOWEST or a quoted string. High-bound may be HIGHEST or a quoted string. The interpretation of a scanning range is consistent with the records-to-use clause, described in Section 3.7.

Switches

/NOACCESS

When a primary index is scanned, VERIFY accesses data records by each of their secondary keys unless this switch is specified. If the primary index is not being scanned, this switch is ignored. Accessing by secondary key is important but expensive. It is the only way to guarantee that a data record can be accessed by each of its secondary keys. However, a scan of the secondary indexes will discover most key inconsistencies caused by aborted updates (as opposed to aborted puts).

/NOFIX

When the RMS file is open for output or patching, VERIFY completes aborted deletes, puts, and updates unless this switch is specified.

/PROGRESS:n2

If the /PROGRESS switch is specified, RMSUTL outputs the VERIFY's progress every n2 keys it scans. Progress is shown by displaying the highest key value scanned so far. If /PROGRESS is omitted, 10000 is assumed. RMSUTL also checkpoints the RMS file and the report file when it makes a progress report.

3.6.15 Secondary Commands

Function

These commands (as shown in the formats below) allow you to perform other functions in the RMSUTL program.

Formats

EXIT (TO MONITOR)

Return to the operating system command level. Any open files are closed. You can type CONTINUE, but you have to (re)open the files you wish to process.

HELP (PLEASE)

Displays description of each RMSUTL command.

TAKE (COMMANDS FROM) file-spec /[NO]DISPLAY

Reads RMSUTL commands from the specified file. If the /DISPLAY switch is specified, TAKE outputs the commands to the terminal as they are processed. The default is /NODISPLAY.

3.7 RECORDS-TO-USE CLAUSE

The record-processing commands allow you to select all or part of a group of records for processing in a single command. You can select a range of records by key value or relative to the current record. Additionally you can select the current record by omitting the records-to-use clause, or you can select the last entry used in a bucket-processing command.

The facility for selecting part of a range is the value-test phrase. The value-test phrase is of the form datafield-name operator value3. When this optional phrase is specified, it causes RMSUTL to apply the indicated test to each record it locates. If the test is false, the located record is bypassed. Datafield-name must have been declared in the DEFINE DATAFIELD command. Value3 must conform to the data type of the specified datafield. Operator can be =, #, >, >=, <, or <= . These are respectively equal, not equal, greater than, greater than or equal, less than, and less than or equal.

Formats

KEY (#) n (FROM) low-bound (TO) high-bound (AND) value-test

RMSUTL locates each record that has a key value that is >= the value derived from low-bound and <= the value derived from high-bound. If high-bound is omitted, RMSUTL supplies a default. N identifies which key to use. Specifying 0 identifies the primary key. Omitting n causes RMSUTL to use the current key of reference.

Low-bound may be LOWEST or a quoted string. The value RMSUTL derives from LOWEST is a string of NULs. If low-bound is LOWEST and high-bound is omitted, high-bound is set to FOUND-KEY-VALUE. If low-bound is a quoted string and high-bound is omitted, high-bound is set to the same value specified in low-bound. Thus, if no records have a key value equal to the quoted string, no records are located. Conversely, if there are multiple records whose key is this value, each is located.

High-bound may be HIGHEST, FOUND-KEY-VALUE, or a quoted string. The value RMSUTL derives from HIGHEST is a string whose bits are all 1s. The value RMSUTL derives from FOUND-KEY-VALUE is the first key value in the file >= low-bound. If low-bound and high-bound are both quoted strings, high-bound must not be less than low-bound.

When the length of a quoted string is less than the defined length of the specified KEY, RMSUTL pads it to the defined length. For low-bounds, it pads with NULs. For high-bounds, it pads with 1 bits. Thus, if you specify DISPLAY DATA KEY 1 "A", RMSUTL locates each record whose first secondary key starts with "A". This is because RMSUTL interprets the KEY range as "A000..." (TO) "A111...", where 000... and 111... indicate padding to defined length with bytes that are all 0s and 1s respectively.

A range containing padded lengths is sometimes "too powerful." Consider a key that is a blank-padded person's name. It is not enough to specify DISPLAY DATA KEY "BROWN" to exclusively locate records whose key is "BROWN ". You must specify "BROWN " to prevent "BROWNxxxxx" from also being located.

LAST-ENTRY

If the primary key is current and a data bucket is current and the last accessed entry is an existing record, the last accessed entry is selected.

RELATIVE-REC-NO (FROM) n1 (TO) n2 (AND) value-test

RMSUTL locates the specified range of records from n1 through n2. Record 0 is the current record; record 1 is the next record; and so on. "Next" is with respect to the sequential ordering implied by the current key of reference.

If n2 is not specified, only the record identified by n1 is located and a value-test may not be specified. If n2 is specified, it must be greater than n1. If n2 is past the last record in the current index, no error results.

3.8 THE REPORT FILE

Output from DISPLAY, FIX, and the file-scanning commands is directed to the report file. If no report file is open, this output is directed to TTY: and is intermixed with command status messages. Also the command as typed does not appear in the report, as is usually the case.

3.8.1 DISPLAY OUTPUT

The format in which an entry is displayed depends on the type of bucket. The display of an index entry contains a key value and the page number of a bucket. The display of a secondary-data entry contains a key value and one or more RFAs. The display of a primary-data entry contains the entry's control data and primary key, unless the entry is a POINTER entry. DISPLAY BUCKET and DISPLAY PROLOG usually suppress output of 0-valued fields.

Record displays are analogous to entry displays. In effect, control field-names are replaced by user field-names. Additionally, the record's RFA is included in the display. If the record has a POINTER entry, the POINTER entry's RFA is displayed as well. The RFA information is important when you are examining SIDRs.

3.8.2 File-Scanning Output

File-scanning output falls into two categories: scan status messages, which are enclosed in square brackets; and scan results.

There are the following types of scan status messages:

1. Completion messages occur when the scan of an index has been completed. For example, if you specify VERIFY ALL for a three key file, three completion messages are output. A normal completion message for VERIFY (or UNCLUTTER) contains a count of the records scanned. For the primary key, this is the number of existing user records. For secondary keys, it is the number of SIDRs. An abnormal completion message occurs if a loop is detected in the data buckets of an index.

2. Progress reports occur as a result of the /PROGRESS:n switch, where n refers to existing user records or SIDRs, as appropriate. When RMSUTL outputs a progress report, it also checkpoints the report file and RMS file. Thus, you are guaranteed that you can safely resume a scan with the key value output in the last progress report.

3. [Fixing] and [Fixable] are appended to scan results to indicate what RMSUTL took care of or can take care of for you. The two messages are mutually exclusive; an updating scan outputs the [Fixing], and a read-only scan outputs the [Fixable].

4. [Aborting scan of current bucket] occurs after the third uncorrectable inconsistency has been detected in a bucket. It is simply a means of bounding output when "garbage" is being scanned.

5. [Changing to /NOFIX ...] occurs after an UNCLUTTER or updating VERIFY has detected an inconsistency RMSUTL cannot correct. It is a precautionary measure to prevent RMSUTL from further clobbering your file.

6. [Empty RFA list ...] occurs when RMSUTL detects an SIDR whose RFA list contains all NILs. You need do nothing in response to this message, but you may reclaim some space by DELETing the specified entry if desired.

The scan-result messages are as follows. Except where noted, the messages relate to the VERIFY and UNCLUTTER commands.

For fixable conditions, the message description shows the FIX command you would later type in if the message occurred doing a read-only VERIFY. Most of the other conditions should never occur. For these, you will usually have to do most of the diagnosis yourself. Your primary tools are SET INDEX, SET BUCKET, and DISPLAY BUCKET.

Access by key n1 failed for rfa1

The record with RFA rfa1 could not be accessed by the indicated secondary key. This is normally caused by a put or update that aborted after the user record was updated but before all of its secondary keys were processed. This inconsistency is fixable; specify FIX rfa1. If the fix fails, "Could not insert key into secondary index" is output.

If the ID of one or more entries is clobbered, it is possible to get a cyclic fix. You fix key 0 and get "Access ... failed"; you fix a secondary key and get "No matching data record ..."; you fix key 0 and get "Access ... failed"; and so on.

Data bucket at page n1 points at page n2 but succeeding index entry does not

Normally consecutive data buckets are pointed at by successive index entries (or the same index entry in a horizontal search situation). This message can mean that a whole page of records has been accidentally bypassed in the data bucket chain. You will be able to access the affected records by key, but not sequentially. To locate the relevant index entries:

```
SET INDEX index-being-scanned BUCKET n1
DISPLAY BUCKET LAST
SET BUCKET UP
DISPLAY BUCKET KEY-VALUE "key of entry just DISPLAYed"
```

Data bucket clutter n%

This is output from a SPACE scan. It indicates the percentage of total data bucket space that is currently devoted to POINTER entries and records with the DELETEd attribute on. This message is output for the primary key only.

Data bucket fullness n%

This is output from a SPACE scan. It indicates the percentage of total data bucket space that is currently in use (including clutter).

Data record identified by back pointer for rfa1

The RFA field of the data record at rfa1 points at another data record rather than at a POINTER record.

Duplicate key encountered for rfa1

Duplicates are not allowed for the key being scanned, but the entry at rfa1 has the same key value as the entry that precedes it.

Key access aborted for rfa1

An unexpected error occurred in RMS when the record at rfa1 was accessed by key. This may indicate clobbered index buckets or a bug in RMS.

Key access failed for rfa1

The record at rfa1 could not be found by a key access. This may indicate clobbered index buckets. Other records earlier in the same bucket may also be inaccessible.

Key value out of order for rfa1

The entry at rfa1 has a lower key value than the entry that precedes it. If this message occurs, it is likely that "Key access failed ..." will occur for the last entry in the bucket. In the special case that the message occurs for a record with the DELETEd attribute on, simply DELETE BUCKET ID ID-of-rfa1 after making the bucket of rfa1 current.

No matching data record for RFA n1 (rfa1) of rfa2

The n1th RFA of the SIDR at rfa2 contains the pointer rfa1, and rfa1 does not identify an existing record with the same key value as appears in the SIDR. Normally this means that a delete or update aborted after the record's entry was updated but before all of the old secondary references were deleted. This inconsistency is fixable; specify FIX rfa2 index-of-scan.

If the match did not occur because the data record contained a different key value, RMSUTL also outputs "Access ... may fail ...". This is because VERIFY KEY 0 may lead to "Access ... failed for rfa1". Specifying FIX rfa1 will correct the problem if it does exist.

Page n1 not start of bucket OR ... clobbered OR not part of index n2

During the scan, RMSUTL tried to read a bucket at page n1, and it was bad in some way. You will have to diagnose why; start by specifying SET INDEX n2 BUCKET n1 and then displaying the bucket's header.

POINTER entry does not point at data record for rfa1

The data record at rfa1 pointed back at a POINTER entry that does not point at it. Normally this is caused when a put aborted after a bucket split but before the record's POINTER entry could be updated. This inconsistency is fixable; specify FIX rfa1.

POINTER entry not found for rfa1

The data record at rfa1 pointed back at an empty slot. This should not happen, but is fixable; specify FIX rfa1. This message may occur in conjunction with "Access ... failed ..." messages.

CHAPTER 4

FORMAT OF AN RMS RECORD FILE

An RMS record file consists of a prologue section and a data section. The prologue section is the repository of the file's attributes. The data section contains your records (and indexes if applicable).

4.1 PROLOGUE SECTION

The prologue section of an RMS file contains a file descriptor. A file descriptor contains all the permanent attributes you specified in the DEFINE FILE command or in your program (the DEFINE FILE command is described in Chapter 3). An indexed file also contains one or more key descriptors and one or more area descriptors, including the descriptor of area 0, which RMS creates. There is a second difference as well. In nonindexed files, the data section immediately follows the prologue. In indexed files, the data section starts at the next page boundary.

The two types of fields in descriptors are argument block fields and RMS-created fields. File attributes are specified to RMS when a file is created. They are specified in argument blocks called FABs and XABs (file and extended argument blocks).

4.1.1 File Descriptor For File Argument Block (FAB)

A file descriptor contains the following FAB information. The field-names recognized by RMSUTL are given.

- BUCKET-SIZE represents the unit of I/O for the file, and it is specified in terms of pages (1 page = 512 words).

- BYTE-SIZE is the number of bits per byte in records in this file. It must be 6, 7, or 9 for indexed files, 7 for stream files, and not greater than 36 for relative and sequential files.

- RECORD-SIZE — If the record format is variable, this field is the maximum number of bytes that can be in a record in the file. If you try to write a record whose size is larger than RECORD-SIZE, an error status is returned and the record is not written. However this check is bypassed if RECORD-SIZE is 0.

  If record format is fixed length, RECORD-SIZE is the number of bytes in each record in the file. If you perform an output operation and do not set record size to the value of RECORD-SIZE, RMS does not write the record and returns an error.

- ORGANIZATION — is SEQUENTIAL, RELATIVE, or INDEXED.

- FORMAT — is either FIXED or VARIABLE.

A file descriptor also contains:

- AREA-COUNT — is the number of areas defined for the file.

- AREA-OFFSET — is the number of words in the prologue preceding the first area descriptor.

- KEY-COUNT — is the number of keys defined for the file.

- KEY-OFFSET — is the number of words in the prologue preceding the first key descriptor.

- PAGES-IN-FILE — is the number of pages currently in the file.

4.1.2 Key Descriptor For Extended Argument Block (XAB)

A key descriptor contains all the field-names of a KEY XAB:

- DATA-AREA — This field must equal the ID field of some AREA descriptor. RMS assigns the data buckets for this key to that area and thereby sets their bucket size. Note that, for the primary key, the data buckets contain user data; and, for secondary keys, they contain the keys by themselves. If DATA-AREA is 0, the default area is used, and its bucket size is taken from the BUCKET-SIZE field of the file descriptor.

- DATA-FILL provides a means of spreading out the data in a file when you first load the file. This field is used to indicate the number of words that can be used in a data bucket when loading this key's data buckets. For example, if it is desirable that buckets be no more than half full and the bucket size is 1 (=512 words), this field would be set to 256. If DATA-FILL is 0, the entire bucket can be filled.

- DATA-TYPE determines the byte size RMS uses to compare keys. Therefore, it must be set so that key byte size matches the BYTE-SIZE field of the file descriptor. It should be SIXBIT for 6-bit bytes. It should be ASCII for 7-bit bytes. It should be EBCDIC for 9-bit bytes.

- ATTRIBUTES The ATTRIBUTES field controls the handling of keys during put and update operations. The "changeable" attribute means you can change the value of this key when you update a record. "Changeable" can not be specified for the primary key. The "duplicates" attribute means that multiple records in the file can contain the same value of this key. Records having duplicate keys are stored in the file so that sequential retrieval of them is in the order in which they were stored.

- INDEX-AREA must equal the ID field of some AREA descriptor. It assigns the index buckets for this key to that area and thereby sets their bucket size. If INDEX-AREA is 0, the default area is used, and its bucket size is taken from the BUCKET-SIZE field of the file descriptor.

- INDEX-FILL provides a means of spreading out the index data in a file when you first load the file. This field is used to indicate the number of words that can be used in an index bucket when loading this key's index buckets. For example, if it is desirable that index buckets be no more than half full and the bucket size is 1 (=512 words), this field would be set to 256. If INDEX-FILL is 0, the entire bucket can be filled.

- KEY-OF-REFERENCE

  contains a number that indicates which key this is: 0 for the primary key, 1 for the first secondary key, and so on.

- POSITIONs — define the starting byte number of each segment of this key. Each starting position is paired with the corresponding size value. A key has between one and eight segments. A size field containing 0 implies that the preceding segment was the last. Key segments are logically concatenated to form a particular key value.

- SIZEs — define the number of bytes in each segment of this key. Each size value is paired with the corresponding starting position. A key has between one and eight segments. A size field containing 0 implies that the preceding segment was the last. Key segments are logically concatenated to form a particular key value. The sum of the specified segment sizes must be less than 256 characters.

A key descriptor also contains the following:

- LEVELS — is the number of levels in this key's index.

- NEXT-KEY — is the offset of the next key descriptor in the prologue.

- ROOT-PAGE — is the page number where the root bucket for this key's index is located.

An indexed file can have from 1 to 17 areas. Area 0 is implicitly defined by RMS. The bucket size of area 0 is taken from the BUCKET-SIZE field of the file descriptor. BUCKET-SIZE is the only field in an area descriptor.

4.2 DATA SECTION OF SEQUENTIAL AND RELATIVE FILES

Records may cross bucket boundaries in a sequential or relative file. The only control information in the data section of these file organizations is the record header. A sequential or relative record header consists of one word, and its format is:

- o ATTRIBUTES — DELETED and USED. DELETED is set if the record is deleted. USED is set when the record is created. RMS checks USED to determine if it is past the EOF (end-of-file) or scanning an empty cell in a relative file.
- o SIZE — is the number of bytes in the record.

4.3 DATA SECTION OF AN INDEXED FILE

Figure 4-1 displays the relationship between a primary index and a secondary index. Each SIDR points at the user data record that contains the same key value as in the SIDR.

```
                                PROLOGUE
                             ---------------
                             |             |
                             |     KEY     |
                             | DEFINITIONS |
                             |             |
                             ---------------
                                    |
                                    |
                                    |
                ---------------------------------------
                |                                     |
                |                                     |
                |                                     |
         ---------------                       ---------------
         |             |                       |             |
         |   PRIMARY   |                       |  SECONDARY  |
         |    INDEX    |                       |    INDEX    |
         |   BUCKETS   |                       |   BUCKETS   |
         |             |                       |             |
         ---------------                       ---------------
                |                                     |
                |                                     |
                |                                     |
         ---------------                       ---------------
         |             |                       |             |
         |     USER    |                       |    SIDR     |
         |     DATA    |<----------------------|             |
         |   BUCKETS   |                       |   BUCKETS   |
         |             |                       |             |
         ---------------                       ---------------

         <-------------------DATA BUCKETS-------------------->
```

Figure 4-1   RMS Record File Format

4.3.1 Bucket Headers

Each bucket in an indexed file contains a header describing its type and other properties. A header contains the following fields:

- AREA-NUMBER — is the number of the area to which the bucket belongs.
- ATTRIBUTES — can be RIGHTMOST and ROOT. ROOT identifies the bucket as the root of an index structure. RIGHTMOST indicates that the bucket contains the highest group of key values at its level in the index structure.
- LAST-ID — is the largest entry ID that this bucket can contain. It is normally 2**17 - 1.
- LEVEL — is the number of levels between the bucket's level and the data level. Thus data buckets are level 0.
- NEXT-BUCKET — is the page number of the next bucket at this level of the index structure, unless this is a rightmost bucket. In this case, it is the page number of the leftmost bucket at this level.
- NEXT-ID — is the ID that is to be assigned to the next entry stored in this bucket. An entry in an index bucket does not have an ID.
- TYPE — can be INDEX or DATA.
- WORDS-IN-USE — is the number of words preceding the first unused word in the bucket.

4.3.2 Entries in a Primary-Data Bucket

Most entries in a primary-data bucket are your records. An entry can also be an internal pointer to a moved record.

The common fields in an entry are:

- ID — is the "address" of an entry within its bucket.

- o ATTRIBUTES — can be any of the following normal cases. DELETED means that the record has been deleted and that RMS will expunge it when reclaiming space in the bucket. DELETED+KEEP means that the record has been deleted and that the only way to expunge it is with RMSUTL. NIL indicates an existing record. POINTER indicates an entry that points at a record that has been moved because of a bucket split.

The additional fields in a RECORD entry are:

- o DATA — is the contents of the record.
- o RFA — is the address in the file where the record was originally stored. The record can still be there, but if it is not, there is now a POINTER entry there.
- o SIZE — is the number of bytes of data in the record. This field is present only if record-format for the file is variable.

The additional fields in a POINTER entry are:

- o RFA — is the address in the file where the record was last moved to.

4.3.3 Entries in an Index Bucket

An entry in an index bucket contains the following fields:

- o ATTRIBUTES — are HIKEY or NIL. HIKEY is on if this entry points to the bucket containing the highest key value at its level in the index.
- o DOWN-POINTER — is the page number of the bucket whose keys are all less than or equal to the value in KEY-VALUE. Also, the lowest key value in the pointed-to bucket is always greater than the KEY-VALUE in the preceding entry.
- o KEY-VALUE — RMS uses the KEY-VALUE field to determine the DOWN-POINTER to follow down an index during a key access. Index entries are sorted in ascending order by KEY-VALUE, so RMS uses the DOWN-POINTER for the first KEY-VALUE field that is greater than or equal to the key value presented by the user.

4.3.4 Entries in a Secondary-Data Bucket

Secondary-data buckets maintain the sorted ordering of keys in a secondary index and are the link between secondary key values and the various records that contain them. An SIDR contains the following fields:

- ID — is the address of the entry within its bucket.
- KEY-VALUE — contains the key value upon which the entry is sorted.
- RFA 1 - RFA n — identify the records whose key values match the value in KEY-VALUE. These pointers are not modified when a record is moved during a bucket split. Thus, they may point at POINTER entries, which in turn point at the data records.

  When a record is deleted, RMS locates each SIDR that points at the record and sets the relevant RFA in each to NIL. When a secondary key value of a record is modified, RMS locates the SIDR containing the old key value and sets the relevant RFA to NIL.
- SIZE — is the number of words used to hold the KEY field and all the RFA fields.

CHAPTER 5

RMS STATUS CODES

Normally RMSUTL, the COBOL OTS, and BASOTS transform status codes returned by RMS into responses that do not involve telling you the actual code. However, it is possible for an unprogrammed-for condition to occur, in which case COBOL OTS and BASOTS display the underlying RMS status code.

There are two types of status codes which are returned by RMS:

1. ER$ - error codes

2. SU$ - success codes

Error codes are larger than success codes, and the minimum error code is ER$MIN. Thus, if RMS returns a code greater than or equal to ER$MIN, it must be an error code. All status codes are represented by a mnemonic symbol preceded by a two-character string indicating whether the code is a success or error code.

ER$MIN may change in the future, but it is currently 300000 octal. SU$MIN (success minimum code) may also change in the future, but it is currently 1000 octal. However, the relative value of status codes with respect to SU$MIN and ER$MIN will not change.

The following table names and describes each of the RMS status codes. If an STV (subsidary status value) value is associated with an error status, its meaning is also described. The VALUE field is the status code's offset in decimal (and octal) from xx$MIN.

Table 5-1

RMS Status Codes

NAME	VALUE	MEANING
SU$SUC	0	Operation was successful.
SU$IDX	1	Index could not be updated because an unexpected error (for example, I/O) occurred while updating the index. This is precisely the state an indexed file is in if an ill-timed crash occurs. The record is still accessible but efficiency can be affected, so you should consider reorganization (use the UNCLUTTER command or reload the file).
SU$REO	2	File should be reorganized because RMS needed to insert a record into a bucket that has no more record IDs available. This can occur only if a large number of deletes, updates, and puts are done to a bucket because there are 2**17-1 IDs available per bucket. This status is returned both on the $PUT that caused the status and when the file is closed.
SU$RRV	3	An internal record pointer could not be updated. This state can be reached as a result of a crash during $PUT and $UPDATE. This status code merely shows a case where the state has been reached. It shows that one or more records can not be accessible by its secondary key, or by RFA addressing. Reorganization is suggested, (use the UNCLUTTER command or reload the file).
SU$DUP	4	A record was $PUT or $UPDATE, and one or more of its key values was in a record already in the file. SU$DUP applies rather than ER$DUP because XB$DUP was set for each such key.

ER$AID	0	AID field in AREA XAB is not greater than AID field in preceding AREA XAB, or it is 0 or greater than 16. STV contains the address of the bad XAB.
ER$BKZ	4	BKZ in AREA XAB is 0 or greater than 7. STV contains the address of the bad XAB.
ER$BLN	5	BLN value in argblk is not correct for the specified BID value.
ER$BSZ	6	BSZ is not 7 for stream file or BSZ is 0 or greater than 36 for some other file type.
ER$BUG	7	Internal error detected in RMS. If the internal error was caused by a monitor detected condition, the STV field contains the monitor error code.
ER$CCF	8 (10)	Can't $CLOSE file. An unusual condition arose that prevented RMS from closing the file. Check the STV field for the monitor error code.
ER$CCR	9 (11)	Can't $CONNECT RAB because another RAB is already connected to the indicated file.
ER$CEF	11 (13)	Can't $ERASE file. RMS could not erase a file for an unknown reason. Check the STV field for the monitor error code.
ER$CGJ	12 (14)	Can't get a JFN for this file for an unknown reason. Check the STV field for the monitor error code.
ER$CHG	13 (15)	Illegal key value change. An $UPDATE was done in which the value of a particular key in the record was changed, and the key was defined without the XB$CHG attribute. STV contains the key of reference of the improper key value. If there is more than one improper key value, the lowest key of reference is placed in STV.
ER$COD	14 (16)	COD field in XAB not XA$ALL, XA$DAT, XA$KEY, or XA$SUM. STV contains the address of the bad XAB.
ER$COF	15 (17)	Can't open file. Caused by an unexpected error when RMS tried to open the file. Check the STV field for the monitor error code.

ER$CUR	16 (20)	No current record. Caused by a $UPDATE, $TRUNCATE, or $DELETE that was not preceded by a successful $FIND or $GET operation.
ER$DAN	17 (21)	DAN field of KEY XAB contained value greater than your highest area ID. STV contains the address of the bad XAB.
ER$DEL	18 (22)	A record accessed with RFA addressing has been deleted since you saved the RFA.
ER$DEV	19 (23)	Device not disk (RMS-10), device not disk or terminal (RMS-20), or device not disk on $CREATE.
ER$DME	22 (26)	Dynamic memory exhausted. RMS could not allocate temporary storage for buffers, and so on. If you set MBF to a large value on your $CONNECTs, you might try using a smaller value.
ER$DTP	23 (27)	DTP field in KEY XAB is not XB$SIX, XB$STG, or XB$EBC, or BSZ in FAB is not respectively 6, 7, or 9. STV contains the address of the bad XAB.
ER$DUP	24 (30)	A record was $PUT or $UPDATE, and one or more of its key values was in a record already in the file. ER$DUP applies rather than SU$DUP because XB$DUP was not set for some of them.
ER$EDQ	25 (31)	ENQ/DEQ monitor error. An attempt to (un)lock a record (or file) resulted in an unexpected error from the monitor. Check the STV field for the monitor error code.
ER$EOF	26 (32)	You sequentially read ($FIND or $GET) past the last record in the file or index.
ER$FAB	27 (33)	BID field of FAB did not contain FA$TYP.
ER$FAC	28 (34)	The file access field in the FAB is not compatible with what you did. 1. Occurs if FB$PUT not set on a $CREATE. 2. Occurs if a record operation ($GET, $PUT, ...) is attempted with the corresponding FAC bit not set.

ER$FEX	29 (35)	You tried to $CREATE an existing file, and FB$CIF and FB$SUP were both off.
ER$FLG	30 (36)	XB$CHG was set for primary key. STV contains the address of the bad XAB.
ER$FLK	31 (37)	File already locked. Someone else has already opened and locked the file with an access incompatible with the access you requested.
ER$FNC	33 (41)	You cannot $ERASE a file when anyone else has it open. Just wait and try again.
ER$FNF	34 (42)	File not found. The file name you specified by FNA did not identify any file.
ER$FSI	96 (140)	File spec on $CREATE, $ERASE, or $OPEN contained illegal syntax of some sort (for example, two directory specs).
ER$FUL	37 (45)	An RMS file can be no larger than 256K pages.
ER$IAL	38 (46)	Illegal argument on call.
ER$IAN	39 (47)	IAN field of KEY XAB contained value greater than your highest area ID. STV contains the address of the bad XAB.
ER$IFI	43 (53)	The IFI field did not identify an internal file block. This is normally caused by a $CLOSE or $DISPLAY on a FAB that has not been correctly opened.
ER$IMX	45 (55)	Multiple SUMMARY or DATE XABS appeared in an XAB chain on $OPEN or $DISPLAY. STV contains the address of the bad XAB.
ER$ISI	48 (60)	The ISI field did not identify an internal record block. This is normally caused by a record operation on a RAB that has not been correctly connected.
ER$JFN	49 (61)	On $OPEN or $CREATE or $ERASE, you specified a nonzero JFN to RMS-10, or you specified a JFN that TOPS-20 did not recognize, or you specified the JFN of an open file.
ER$KBF	50 (62)	You did a $FIND or $GET with RAC equal to RB$KEY, but did not set KBF.

ER$KEY	51 (63)	KBF on a key access to a relative file pointed to a zero or a number greater than the MRN value for the file.
ER$KRF	52 (64)	You specified an incorrect key of reference for an indexed file. This can happen on a key $GET, key $FIND, or $CONNECT that specified a KRF greater than the highest key defined for the file.
ER$KSZ	53 (65)	You specified a KSZ value on a random $GET or random $FIND that is greater than the defined length of the key.
ER$MRS	56 (70)	Invalid MRS value on a $CREATE, caused by a value of zero when RFM is FB$FIX or ORG is FB$REL.
ER$NEF	57 (71)	Not at end of file. Caused by an attempt to $PUT into the middle of a sequential file.
ER$NPK	59 (73)	No primary key. Caused by an attempt to $CREATE an indexed file without specification of a primary key. In other words, first KEY XAB in chain had a nonzero REF field.
ER$NXT	60 (74)	Incorrect value of NXT field and XAB, caused by an address in the range 1-17. STV contains the address of the bad XAB.
ER$ORD	61 (75)	Either KEY or AREA XABS are not in ascending order with respect to their ID field (respectively REF and AID). STV contains the address of the bad XAB.
ER$ORG	62 (76)	ORG was not FB$SEQ, FB$REL, or FB$IDX on a $CREATE.
ER$PEF	63 (77)	Can't position to EOF. Caused by RB$EOF being set on $CONNECT of nonsequential file.
ER$PRV	66 (102)	Privilege Violation. Caused by an attempt to open file for access for which you don't have the rights.
ER$RAB	68 (104)	BID field of RAB did not contain RA$TYP.
ER$RAC	69 (105)	RAC was not RB$SEQ, RB$KEY, or RB$RFA on a $FIND or $GET, or RAC was not RB$SEQ or RB$KEY on a $PUT.

ER$RAT	70 (106)	You specified an invalid RAT value on a $CREATE. RAT was nonzero and not FB$BLK, or it was FB$BLK for a stream file.
ER$RBF	71 (107)	RBF was not set up on a $PUT or an $UPDATE.
ER$REF	72 (110)	REF field in KEY XAB is not greater than REF field in preceding KEY XAB, or it is greater than 255. STV contains the address of the bad XAB.
ER$RFA	75 (113)	The RFA field on an RFA access contained zero or identified a never used cell.
ER$RFM	76 (114)	On a $CREATE, RFM was FB$LSA or FB$STM and ORG was not FB$SEQ, or RFM was not FB$LSA, FB$STM, FB$FIX, or FB$VAR.
ER$RLK	77 (115)	Record locked. Caused by an attempt to access a record that is currently locked by another process.
ER$RNF	78 (116)	Record not found occurs in the following situations: 1. Key $FIND or key $GET to cell in relative file that is empty or contains a deleted record. 2. Exact key access. There is no record with the specified key value. 3. Generic key access There is no record whose key starts with the specified key value. 4. (Generic) approximate key access is attempted. The specified key value is greater than (RB$KGE) or greater than or equal to (RB$KGT) for any key in the specified index.

ER$RSZ	84 (124)	RSZ not equal to size of record on $UPDATE, or RSZ was not valid on $PUT: 1. RFM is FB$FIX and RSZ is not equal to MRS. 2. RSZ is greater than MRS. 3. ORG is FB$IDX and RSZ is greater than the number of bytes in a data-record bucket.
ER$RTB	85 (125)	Move mode applied on a $GET, when the record was larger than the number of words specified in USZ. Note that RMS does fill your buffer. Check the STV field for the number of bytes actually in the record.
ER$SEQ	86 (126)	Keys out of sequence. Caused by a sequential $PUT to an indexed file in which the primary key value is not greater than the key value on the previous $PUT. This restriction is enforced only if the last operation was also a sequential $PUT.
ER$SIZ	87 (127)	The number of bytes in a key is 0 or greater than 255. STV contains the address of the bad XAB.
ER$UBF	90 (132)	UBF was not set up on a $CONNECT or a $GET.
ER$UDF	91 (133)	The file is in an undefined state and should be reorganized. STV contains a code that further explains the state of the file (see Table 5-2).
ER$XAB	94 (136)	BID field of XAB did not contain XA$TYP. STV contains the address of the bad XAB.

The following table describes the STV values for ER$UDF. The VALUE field is an offset from ER$MIN.

Table 5-2

STV Values for ER$UDF

NAME	VALUE	MEANING
ER$RRV	81 (121)	Bad internal record pointer encountered in file.
FE$BEM	320 (500)	Empty index bucket detected.
FE$BFC	321 (501)	Bad file class found in file FDB.
FE$BHE	322 (502)	Bucket header has bad format.
FE$HNF	327 (507)	File prologue header was not found.
FE$NOA	328 (510)	No area descriptors were found in the file.
FE$NOI	329 (511)	No index descriptors were found in the file.
FE$NOU	332 (512)	No data record found for RFA in secondary index.

APPENDIX A

USAGE OF AN RMS FILE FROM BOTH BASIC+2 AND COBOL-74

(TOPS-20 ONLY)

When you create an RMS file to be used by both BASIC+2 and COBOL-74, you should observe the following rules:

1. The file must be indexed and its keys must be ASCII.

2. A record may contain only:

COBOL	BASIC
DISPLAY-7	NAME$
COMPUTATIONAL	NAME%
INDEX	NAME%
COMPUTATIONAL-1	NAME

3. A COBOL record may not contain a group-level field that has an OCCURS clause.

4. There must be no usage of the JUSTIFY or SYNC keywords.

5. You must declare fields in the same order, set string lengths the same, and set array bounds the same.

All the file parameters established by COBOL can be processed in BASIC+2 Version 2.1 or later.

All the indexed file parameters established by BASIC+2 can be processed by COBOL. However, full compatibility is achieved only when:

- o The file is defined with duplicates not allowed for the primary key.

- o Changes are allowed for all secondary keys.

If the primary key is defined such that duplicates are allowed, you must have a DECLARATIVE procedure to intercept the error COBOL generates for this condition. For example,

```
        ENVIRONMENT DIVISION.
        FILE-CONTROL.
                SELECT RMS-FILE ASSIGN TO DSK;
                        .
                        .
                        .
                FILE STATUS IS FS-1, FS-2, ACTION-CODE.
                        .
                        .
                        .
        DATA DIVISION.
                .
                .
                .
        01      FS-1    PIC 99.
        01      FS-2    PIC 9(10).
        01      FS-2-IMAGE      REDEFINES FS-2.
                02      FILLER  PIC X(7).
                02      ERR-CODE        PIC X(3).
        01      ACTION-CODE     INDEX.
                .
                .
                .
        PROCEDURE DIVISION.
        DECLARATIVES.
        OPEN-ERROR SECTION.
                USE AFTER STANDARD ERROR PROCEDURE ON RMS-FILE.
        OPEN-ERROR-1.
                IF ERR-CODE = "523"
                        MOVE 1 TO ACTION-CODE.
        OPEN-ERROR-EXIT.
                EXIT.
        END DECLARATIVES.
                .
                .
                .
        Paragraph-name.  or  Section-name SECTION.
                remainder of PROCEDURE DIVISION
```

If a file defined in BASIC+2 has a secondary key defined without changes allowed, you can use the file freely in COBOL with one exception. To detect an attempt (by a terminal operator for example) to update the key, you must have a DECLARATIVE procedure that checks for an error code (ERR-CODE) of 506. Alternatively you can eliminate the problem by using RMSUTL to make all such keys changeable. See the description of the CHANGE PROLOG command in section 3.6.1.

INDEX