Global 16-bit Development System Data Management (DMAM) Manual Version 8.1

TABLE OF CONTENTS

APPENDICES

1. Introduction

1.1 The Global Cobol Data Management System

Designed as an extension to the Index-Sequential file structure (ISAM) available within Global System Manager, the Global Cobol Data Management system permits ISAM-like access to a number of sets of records contained within a single database file.

Each Data Management database file (also called a DMAM file) may contain up to 50 different sets of records. Each set of records may have up to 16 indexes, each of which will have its own method of constructing a key for the record. All the records of a set have the same length, and the same indexes apply to them, but distinct types of records may be used within a single record set.

Unlike earlier versions of the Global Cobol Data Management system, which only allowed DMAM databases to be held entirely within the Global System Manager directory structure (i.e. Global format DMAM databases), the V8.1 Global Cobol Data Management system allows DMAM databases to be held in any of the following formats:

- **Global format DMAM database** - held entirely within the Global System Manager directory structure and compatible with the pre-V8.1 Global Cobol Data Management system. This option is available on all Global System Manager implementations;

- **C-ISAM format DMAM database** - a single DMAM database is emulated by a series of Informix C-ISAM files. This option is only available on some Global System Manager (Unix) implementations - please consult your Configuration Notes for further information;

- **Btrieve format DMAM database** - a single DMAM database is emulated by a series of Btrieve files. This option is only available on Global System Manager (MS-DOS and Windows) and Global System Manager (Novell NetWare).

In addition to multiple indexes on records, Global format DMAM databases also have dynamically updated B*-tree indexes to hold the record keys. This means that the problems associated with long overflow chains in ISAM do not occur with DMAM.

A description of the concepts underlying the Global Cobol Data Management system can be found in chapter 2, and a detailed description of the internal structure of a DMAM database can be found in Appendix C.

1.2 Data Management Programming Support

The programming support for the Global Cobol Data Management system consists of the Data Management Access Method (DMAM), which permits you to read and write records to the database file, along with a number of system routines to obtain statistical information and perform certain kinds of special processing, such as rebuilding an index in the file. The Data Management Access Method (DMAM) itself is documented in chapter 3, and the various system routines are covered in chapter 4.

Creating new DMAM database files is achieved using the DBMAIN utility, which allows you to define record types and set up the various keys by

which they are to be accessed. You may also use DBMAIN to amend the key set-up on a DMAM database after it has been in use for a time. DBMAIN, together with other system utility programs, is documented in chapter 5.

1.3 Extensions for V8.1 Programming Software

Extensions to the Global Cobol Data Management system have been made as part of the V8.1 release of the Global Cobol Development software. The extensions have been made to optimise performance or to provide wholly new facilities for managing DMAM database files. The changes for V8.1 are as follows:

- A Memory Page version of DMAM is available (see chapter 3);
- A version of DMAM is now available which holds the data in either a series of Informix C-ISAM files (i.e. a C-ISAM format DMAM database), or a series of Btrieve files (i.e. a Btrieve format DMAM database) or as a Global format DMAM database (see chapter 3);
- A new routine DBKES$ is provided, to return index key segment data from a DMAM database so that the key structure can be examined by an application program (see chapter 4);
- A database structured dump and diagnostics utility, DMSDUMP, is available (see chapter 5).

Programs written to use the facilities of the V6.1 or V6.2 Global Cobol Data Management system will continue to work unchanged with the V8.1 Global Cobol Data Management system. However, programs using any new features of the V8.1 Global Cobol Data Management system must **only** be used with the V8.1 support utility programs documented in chapter 5, and with the V8.1 subroutines (documented in chapter 4), otherwise data corruption may occur.

In addition, programs using the Memory Page version of DMAM or if the database is held as a C-ISAM format DMAM database, or a Btrieve format DMAM database, may only be run under Global System Manager V8.1, or later.

2. Concepts Underlying Data Management

2.1 The Database File

Each Data Management file, conventionally termed a **database**, may contain a maximum of 50 separate **record sets**. A Global format DMAM database is allocated as a single file so far as ordinary Global System Manager directory handling processes are concerned. A C-ISAM format DMAM database is held as a single Global format **schema file** describing the C-ISAM file format together with a series of C-ISAM databases held in a Unix directory (see appendix C). A Btrieve format DMAM database is held as a single Global format **schema file** describing the Btrieve file format together with a series of Btrieve databases held in an MS-DOS (or Novell NetWare) directory (see appendix C).

2.2 The Record Set

A record set is a collection of records which all have the same fundamental structure, that is to say all have the same record length and key fields, even though they may have different record types, and may contain different kinds of information. A DMAM record set may be regarded as a logical data file contained within the DMAM database.

Although the maximum number of record sets which can be contained within a DMAM database is 50, when the database is created the actual maximum for that database may be set to a lower number. Each potential record set that a database may contain occupies about 400 bytes of space in the database header (about 3000 bytes for a schema file), so some space may be saved by reducing the number of record sets allowed.

2.3 Keys and Indexes

For each record set up to 16 different **keys** may be defined. Each key is built from a selection of fields within a data record, and is present in an **index**. Each index therefore permits access to the record set, ordering the records in a way dependent on how the key has been structured.

Each index has an identifying eight-character name which must be unique within the database, and it is this **index name** which is used by an application program to gain access to a record set for processing purposes. A maximum of 150 indexes may be defined in total in a single DMAM database, regardless of how many record sets may exist.

An index may be **unique**, in which case duplicate values of keys are not permitted, or **non-unique** where duplicates are allowed. For a non-unique index, keys which consist of all low-values, or all spaces, can be omitted from the index. This would normally be done to save index space and improve access speed where only a small proportion of records have other values, and only these records are of interest when processing the index. For C-ISAM format DMAM databases and Btrieve format DMAM databases, these records will be part of the C-ISAM, or Btrieve, index but will be omitted by DMAM when reading records.

Each unique index to a record set is functionally similar to an ISAM file with that key. DMAM is arranged in this way so as to facilitate the conversion of existing ISAM-based applications.

Important note: In the V6.2, and later, Global Cobol Data Management system the handling of ignored key values has been extended, so that a key may be ignored based on its entire value (as described above), or

on the value of either the first or last segment of the key. This extra facility is extremely useful when dealing with complex key structures. If this extended facility is used it is **vital** that keys are rebuilt using the V8.1, or later, IRBLD$ routine (or the V8.1, or later, DBREORG) otherwise data corruption may occur.

2.4 The Structure of Keys

Each key on a record may consist of up to 8 **segments**. A segment is a contiguous area of data within the record, and will typically be a single data field from the record. Each segment is concatenated with the other segments to form the total key.

All segments of a key must lie wholly within the first 255 bytes of the record. The total length of the key may not exceed 99 bytes. For Global format DMAM databases, the access method optimises REWRITE performance by avoiding any key checking if the record is unchanged up to the last byte used as a segment of any key. Thus, it is recommended that, so far as possible, fields to be used as part of keys are grouped at the start of the record, with any non-keyed data at the end.

2.5 Key Segments

Each segment of a key may be either a character field (collated in true ASCII sequence), a computational field (collated from negative numbers, through zero to positive numbers) or as a **translation** field. The segment may also be collated in normal (ascending) sequence, or reverse (descending) sequence.

2.6 Translation Segments

A translation segment is one which is translated via the character translation table set up in the database header before inclusion in the key. The translation mechanism, which is covered in more detail in Appendix D, permits foreign language keys to be collated in the correct sequence, and allows compressed, case and punctuation insensitive alpha keys for fields such as a customer name.

By default a database is set up with a translation table which ignores all characters in the field except the ASCII digits (0-9) and the upper and lower case letters (A-Z and a-z). The letters are collated without regard to case, and three characters are compacted into every 2 bytes used in the key.

This default table is intended to be used to create alpha index keys on long character fields within the record. In addition to the compression and compaction which occurs, the translation keys also truncate if the deduced key would be longer than the specified translation length (which may be shorter than the length of the original field) so that the key space used in the index may be kept to a manageable size.

Important note: Each translation segment specified as part of a key counts as **two** segments against the limit of eight segments from which a key may be constructed.

2.7 Data Allocation

Within a Global format DMAM database are a file header, the records belonging to each record set, and the index data blocks used to hold

the indexing information. File space is allocated dynamically between records and index data as records are added to the database.

For each record set, and for the index blocks, the database header contains an **allocation extent** which is set up using the database maintenance program, DBMAIN, documented in chapter 5. A new extent is allocated whenever a new record or index block needs to be added to the file, and the current extent is full. The database will therefore contain blocks of records, interspersed with blocks of index data, during normal operation.

When a record or index block is deleted it is added to a chain of such deleted elements. Elements from the deleted chain are reused in preference to allocating a new extent.

For a C-ISAM format DMAM database, the data is held in a series of C-ISAM files. A Global schema file is produced which holds the Global index data and the C-ISAM file information (e.g. Unix file and path name) and index data. The Global schema file and C-ISAM files are set up using the database maintenance program, DBMAIN, as documented in chapter 5.

For a Btrieve format DMAM database, the data is held in a series of Btrieve files. A Global schema file is produced which holds the Global index data and the Btrieve file information (e.g. MS-DOS file and path name) and index data. The Global schema file and Btrieve files are set up using the database maintenance program, DBMAIN, as documented in chapter 5.

For a more detailed description of the internal allocation mechanism refer to Appendix C.

2.8 File Recovery

One of the most important initial design criteria of the Global Cobol Data Management system was to ensure, so far as possible, that DMAM database files were robust to machine failure. To achieve this end an automatic recovery mechanism has been built into the Global Cobol DMAM access method (for Global format DMAM databases only).

Whenever a program WRITEs, REWRITEs or DELETEs a record, a copy of the pre-existing situation is preserved in the file header. Whenever the file is OPENed, this header area is checked, to see if a WRITE, REWRITE or DELETE had been in progress but not completed when the file was last used. If this is the case then the operation is completed by the OPEN process before control is returned to the user program.

The effect of this recovery processing is to guarantee data integrity even over machine failure **at the record level.** It is still the responsibility of the application program to deal with any cross-record data integrity checking which might need to be performed.

Any recovery processing for C-ISAM format DMAM databases, where the data is held in C-ISAM files will depend on the recovery processing available in C-ISAM. Similarly, any recovery processing for Btrieve format DMAM databases, where the data is held in Btrieve files will depend on the recovery processing available in Btrieve.

3. The Data Management File Organisation

3.1 Data Management Database Files

The general concepts used with DMAM database file management are explained in chapter 2. It is recommended that you familiarise yourself with the terminology used before reading the rest of the manual in detail. Additionally, Appendix C contains a detailed specification of the structure of a DMAM database file, which may be of interest to systems analysts and designers.

3.1.1 File Processing Statements

The file processing statements OPEN, READ, READ NEXT, READ PRIOR, READ FIRST, READ LAST, READ PHYSICAL, WRITE, REWRITE, DELETE, UNLOCK and CLOSE are used in conjunction with a file definition with ORGANISATION DMAM to process an existing DMAM database file:

OPEN must be executed prior to any other statement affecting the file;

READ is used to retrieve a record at random, given its record key;

READ NEXT retrieves the record whose key is next in collating sequence compared with the key last accessed. The operation can be used to process all or part of the record set in record key sequence;

READ PRIOR retrieves the record whose key immediately precedes in collating sequence the key last accessed. The operation can be used to process all or part of the record set in reverse record key sequence;

READ FIRST retrieves the first record whose key matches the partial key specified, or the first record in the record set if no partial key is specified;

READ LAST retrieves the last record whose key matches the partial key specified, or the last record in the record set if no partial key is specified;

READ PHYSICAL retrieves the record whose physical file address has been specified;

WRITE is used to add a new record and insert suitable keys into the appropriate indexes. It may **not** be used to update an existing record;

REWRITE is used to update the record last accessed;

DELETE is used to delete the record last accessed;

UNLOCK is used to release a lock obtained by a previous READ operation;

CLOSE must be issued to terminate file processing.

DMAM files can only be created and maintained on direct access devices.

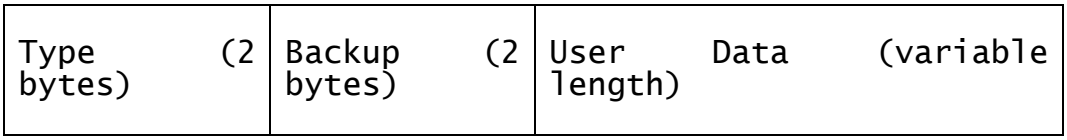
When any of the statements is executed a **file condition**, returned by exception condition 2, may arise as explained in chapter 1 of the Global Development File Management Manual. In addition, if OPTION ERROR is specified in the FD, exception condition 1 will be generated should an irrecoverable I/O error occur on an OPEN statement. Therefore it is usual to follow each file processing statement with an ON EXCEPTION statement. If you do not, and an exception condition arises, your program will be terminated in error. Due to the complexity of processing within DMAM it is not generally possible to pass back exceptions for serious error conditions which may arise during database processing. For this reason a centralised failure routine may be set up using the ON FAILURE statement documented in 3.2.10.

If an error occurs when accessing a C-ISAM format DMAM database or a Btrieve format DMAM database, a file condition will be returned and the C-ISAM UCI (or Btrieve UCI) error code will be returned in the system variable $$CRES. The UCI error codes are fully described in the Global File Converters Manual.

Note that DMAM is functionally similar to the IS access method (described in chapter 3 of the Global Development File Management Manual) when used to access a record set with a single unique key. Section 3.13 summarises the principle differences between DMAM and ISAM as an aid to upgrading existing programs.

3.1.2 Data Record Format

The data records of a DMAM database file consist of a two-character type code followed by a two-byte back-up data field and then any remaining user data, the key being built from within the first 256 bytes of the Global record:



The **type** is similar to that used by Global AutoClerk, where it distinguishes the different sorts of records that make up the record set. The first type character must not be an asterisk, since this is used to denote record deletion using REWRITE explained in 3.5.3. It is important not to confuse the record type field, which may vary from record to record within the record set, with the record-id used to identify the record set itself.

The **back-up** field is used to hold the back-up sequence of records in the database, and is reserved for future use for a database back-up utility. This field should not be used by the application programmer.

The back-up field is followed by any amount of **user data**. The first 256 bytes of the record are used to build the record keys for the various indexes defined for the record set. For Global format DMAM databases only, REWRITE handling is optimised to avoid any key checking if the record is unchanged up to the last byte of data used as part of any index. It is good practice, therefore, to group data fields which are to be part of any key at the start of the user data, and to follow them by non-key relevant data.

3.1.3 Creating a DMAM Database File

To create a DMAM database file you use the Database Maintenance utility (DBMAIN) which is documented in chapter 5. If you are converting data from existing RS or IS files, or copying data records from one DMAM file to another, you can use the data take-on utility to add the records to the database. Alternatively, you can add them yourself under program control using either conventional file processing statements, or the special data take-on routines documented in chapter 4.

When an empty file is created it consists of a header area defining the records and indexes used in the database and, for a Global format DMAM database, a series of empty indexes for the record types concerned. If any translation keys are used the database header will contain the **translation table** used to build them.

As records are added, they too are written to the file (Global, C-ISAM or Btrieve format, as appropriate) and the keys for them are inserted into the appropriate indexes.

You may convert an existing IS file directly to be one of the record sets of a DMAM database. If you wish to convert from an RS file it must contain records of the form described above. The contents of the back-up field are immaterial, but will be overwritten when the records are added to the database.

3.1.4 Specifying File Attributes

To process an existing DMAM database file you specify attributes such as its unit-id, volume-id and file-id in a file definition (FD) coded in the data division. Section 3.2 below describes those parts of the FD which are specific to the DMAM database file organisation, but the statements that are common to all organisations are defined in chapter 1 of the Global Development File Management Manual.

3.1.5 Deletion of Records

The DELETE statement is provided to allow you to delete records from a DMAM database file. However for consistency with ISAM the convention is adopted that records whose type code begins with an asterisk when they are REWRITTEN (or WRITTEN) are also deleted. Deleted records are reused by DMAM, and so they are not accessible to the user program. They are processed as if the ISAM OPTION IGNORE statement had been coded in the file definition.

3.1.6 Reorganising a DMAM Database File

As records are added to a DMAM database the indexes are updated in place, so that large numbers of updates can take place without causing significant performance degradation. However if deletions have taken place deleted records and index blocks may still physically occupy valuable file space. In such an eventuality, or if very large numbers of records have been added, it may prove necessary to reorganise the database to release the space occupied by such records and index blocks.

Reorganisation is accomplished either by using the database index reorganisation command (DBREORG) interactively, or by developing an application program to do part of the job. An application program can use the IRBLD$ system routine described in chapter 4 to rebuild either a single index or all the indexes for a record set, compacting the index blocks and freeing unused index space, but it may not remove

space occupied by deleted records. The DBRBLD utility creates a new Global format DMAM database if record compaction or conversion is required. **DBRBLD cannot be used on C-ISAM format or Btrieve format DMAM databases.**

Note that if you want to extend the space available for adding records and index blocks to Global format DMAM databases, this can be accomplished either by simply copying the file to a larger area - using either the file utility ($F) interactively, or by developing an application program using the COPY$ routine described in the Global Development File Management Manual - or by extending the database size using the $REORG utility documented in the Global Utilities Manual.

3.1.7 Programming Notes

The DMAM database access method described in this chapter is quite complex, to enable you to hold a variety of related data with the appropriate access keys defined on it. For Global format DMAM databases, there are no problems with insertions of keys into an index, but there is only a limited amount of recovery of index space freed by deleting records as this is an infrequent requirement for most applications. You are therefore advised either not to use DMAM for an application where the records have a very high volatility and a large range of possible keys, or to rebuild the indexes periodically on such records (e.g. after about every 3000 updates).

The maximum size key is 99 bytes. Normally keys should be made as small as is practical since this reduces the size and number of levels of the index, and hence the space it occupies.

For Global format DMAM databases, the index contains a number of 512-byte blocks arranged in levels. Each block contains a 500-byte key table capable of holding a number of key values. Call this number, which depends of course on the key length, n. Then the first level index contains one index block for every n records in the prime data area. The second level index contains one block for every n blocks of the first level index. Index levels are constructed one after another until the highest level index, consisting of just a single index block, is developed.

During processing the addition of records and the insertion of keys may cause an index block to become over full. In such a case it is split evenly and an extra entry is placed in the index block at the next higher level. This splitting propagates up the index tree as far as necessary, and it may cause a split of the highest level index block, causing the number of levels of index to be increased by one.

The structure of a Global format DMAM database is described in detail in Appendix C of this manual. For index allocation in C-ISAM format or Btrieve format DMAM databases please refer to the appropriate Informix C-ISAM or Btrieve manual.

3.1.8 Performance Guidelines

3.1.8.1 Global format DMAM databases

READ, READ FIRST and READ LAST need to access one index block from each level and then at least one data record.

READ NEXT and READ PRIOR retrieve a single data record using information saved in the access method or in a save area supplied in

an OPEN statement, and may need to read a single index block as well. If there is no information saved (which can occur immediately after an OPEN, after a WRITE, REWRITE or DELETE, or after a DMAM access on a separate FD) then READ NEXT and READ PRIOR require the same processing as a READ.

READ PHYSICAL retrieves a single data record from the file address supplied, but the address must have been determined from some other database operation previously.

WRITE adds a new record to the file, and then updates every index on that record. Each index update involves accessing one index block from each level, and then writing back one index block. If the updated index block overflows there is a further requirement to write a new index block, and to update a further existing index block (which may possibly in turn overflow).

REWRITE writes the data record previously accessed. Since the address of this record is remembered there is no need for REWRITE to search the index to find the record. It also needs to update the index for each non-unique key on the record (it is not permissible to change the unique keys). To optimise performance any index whose key is unchanged from its previous value is not updated.

DELETE deletes the data record previously accessed, and avoids the need to search the index in the same way as REWRITE. It does however need to update an index for each key on the record, removing the key from the index.

READ PHYSICAL is the most efficient operation, then READ NEXT and READ PRIOR, then READ, READ FIRST and READ LAST, then REWRITE and DELETE and finally WRITE. WRITE and REWRITE performance will be degraded if large amounts of index block splitting are required. The performance of READ NEXT and READ PRIOR can be improved by keeping the last index block used resident in memory. This will happen automatically if only a single FD is using DMAM for reading a database within a program. Any WRITE, REWRITE or DELETE operation may cause the index block to be lost, and any DMAM operation on a separate FD will certainly have such an effect. To avoid the performance impact of these situations the program can supply a 516-byte storage area for the index block in the OPEN statement.

3.1.8.2 C-ISAM format DMAM databases

For C-ISAM format DMAM databases the performance will depend almost entirely on the performance of the C-ISAM Access Method. You should be aware that a single DMAM operation may be mapped into several C-ISAM operations by the C-ISAM Universal Channel Interface (UCI).

3.1.8.3 Btrieve format DMAM databases

For Btrieve format DMAM databases the performance will depend almost entirely on the performance of the Btrieve Access Method. You should be aware that a single DMAM operation may be mapped into several Btrieve operations by the Btrieve Universal Channel Interface (UCI).

3.1.9 Bulk Additions

At times you will need to add large numbers of new records to a DMAM database file. In particular, this will occur during initial data take-on. If you simply add these records directly to the file,

performance will be degraded by the need to split index blocks at frequent intervals.

If you need to add many thousands of records to the database file you may benefit by writing the records using the DTWRT$ routine, and then rebuilding all the indexes on those records using the IRBLD$ routine. There is no performance benefit from the order in which the records are added.

3.1.10 File Locking

The DMAM database access method provides three forms of locking on the database records. The locks are specified by appending a lock-type (one of LOCK, PROTECT or DELETE-LOCK) to any of the READ, READ NEXT, READ PRIOR, READ FIRST, READ LAST, READ PHYSICAL statements. The effect of the various locking options for **Global format DMAM databases** is as follows:

LOCK — obtains an exclusive lock on the record (region = record address). This is intended to serve the purpose of an update lock, held on a record while it is in the process of being updated;

PROTECT — obtains a shared lock on the record, independent of the LOCK (region = record address + 1). This is intended to protect the record locked against deletion or some other major update, while some subordinate record is processed, and is provided to permit the application program to utilise its own master/servant record structures;

DELETE-LOCK — obtains an exclusive lock on the record which precludes all other locks (regions of record address and record address + 1). It is intended to secure a record while some catastrophic update is performed upon it, such as deleting it. Note that it is mandatory to obtain a DELETE-LOCK on a record before using the DELETE statement, but that this is not currently a requirement of the REWRITE deletion processing (in order to maintain consistency with ISAM).

Any further READ type operation on the FD will release any outstanding locks, and will of course obtain a new lock option if such is specified. A REWRITE or DELETE will also release any outstanding locks on the FD after processing is completed, and an UNLOCK or CLOSE will release any locks outstanding. Note that a WRITE leaves the locks unaffected, so that you may lock a header record and then write a series of subordinate records with security.

For **C-ISAM format DMAM databases**, all locking is performed on the Global schema file, within the Global System Manager directory structure, and no locking is attempted on the actual C-ISAM data files. **Therefore, no other non-Global application should attempt to access the C-ISAM data files while they are being accessed by an application that has been developed using the Global Cobol Data Management system.**

For LOCK and PROTECT LOCK operations the 4-byte lock region for the C-ISAM format DMAM database schema files is used as follows:

1st byte of lock region	2nd, 3rd & 4th bytes of lock region
record set	(record number * 2) - 1

For DELETE LOCK operations the 4-byte lock region for the C-ISAM format DMAM database schema files is used as follows:

1st byte of lock region	2nd, 3rd & 4th bytes of lock region
record set	record number * 2

Important note: If locking is to be used then the number of records in a single C-ISAM file must not exceed 8388607 (8Mb-1).

For **Btrieve format DMAM databases**, all locking is performed on the Global schema file, within the Global System Manager directory structure, and no locking is attempted on the actual Btrieve data files. **Therefore, no other non-Global application should attempt to access the Btrieve data files while they are being accessed by an application that has been developed using the Global Cobol Data Management system**.

For LOCK and PROTECT LOCK operations the 4-byte lock region for the Btrieve format DMAM database schema files is used as follows:

1st byte of lock region	2nd, 3rd & 4th bytes of lock region
record set	((record address / Global record number) * 2) - 1

For DELETE LOCK operations the 4-byte lock region for the Btrieve format DMAM database schema files is used as follows:

1st byte of lock region	2nd, 3rd & 4th bytes of lock region
record set	(record address / Global record number) * 2

Important note: If locking is to be used then the record address of records must not be such that record address/Global record length exceeds 8388607 (i.e. 8Mb-1).

3.2 The File Definition

The file definition for a DMAM database file is coded in either working storage or the linkage section as follows:

```
FD filename ORGANISATION DMAM
[ASSIGN TO UNIT unit-id FILE file-id [VOLUME volume-id]]
[INDEX NAME IS index-name]
[RECORD ADDRESS IS record-address]
[RECORD LENGTH IS length]
[SIZE IS size]
[OPTION ERROR]
[ON ERROR intercept]
[ON FAILURE fail-intercept]
```

The FD establishes a special group data item, 248 bytes in length, whose name is filename. The quantities unit-id, file-id, volume-id, index-name, record-address, length, size, intercept, and fail-

intercept appear as subordinate items within this group and can, if need be, be referred to by the application program.

If it is possible to specify unit-id, file-id, volume-id or index-name before the program executes then the quantity should be coded as a character string in quotes. If any of these quantities is not known until run-time then a symbol must be coded for the quantity. This symbol will then label a level 02 item which the user program is responsible for initialising.

3.2.1 The Filename

The filename must be a symbol. It serves to label the file definition, as explained in chapter 1 of the Global Development File Management Manual.

3.2.2 The ORGANISATION Clause

The organisation clause must be coded as shown, except that you may use the abbreviation ORG instead of ORGANISATION. It indicates that the file is a DMAM database.

3.2.3 The ASSIGN Statement

Use of the ASSIGN statement, which is the same for any file organisation, is explained in chapter 1 of the Global Development File Management Manual.

3.2.4 Optional Statement Placement

The INDEX NAME, RECORD ADDRESS, RECORD LENGTH, SIZE, ON ERROR and OPTION statements are optional and may appear in any order following the ASSIGN statement.

3.2.5 The INDEX NAME Statement

The INDEX NAME statement is required to identify the index used to read the DMAM database file. The index-name is coded either as a character string in quotes, or as a symbol. When a symbol is used the statement:

```
02 index-name PIC X(8)
```

is generated within the FD. The program must then place the index name to be used in the field before the file is opened.

3.2.6 The RECORD ADDRESS Statement

The RECORD ADDRESS statement is only required if you wish to use the READ PHYSICAL operation on the database. You code:

```
RECORD ADDRESS IS symbol
```

causing the statement:

```
02 symbol PIC 9(9) COMP
```

to be generated within the file definition. After any successful file processing statement (other than OPEN, CLOSE or UNLOCK) the record address of the record last processed will be returned to you in symbol. You must establish the correct record address in symbol before executing a READ PHYSICAL operation.

Important note: For C-ISAM format DMAM databases only, the value returned in the record address field will be the record number.

3.2.7 The RECORD LENGTH Statement

The RECORD LENGTH statement need only be supplied if you want to determine the record length of the record set at run-time. You code:

```
RECORD LENGTH IS symbol
```

causing the statement:

```
02 symbol PIC 9(4) COMP
```

to be generated within the file definition. When the OPEN operation terminates successfully the record length will be returned to you in the field named symbol.

3.2.8 The SIZE Statement

The SIZE statement is required only when you wish to determine the actual number of bytes allocated to the Global format DMAM database file. The size is coded as a symbol, causing the statement:

```
02 symbol PIC 9(9) COMP
```

to be generated within the FD. When the OPEN statement completes satisfactorily the actual number of bytes allocated to the file will be returned in the generated field.

For C-ISAM format and Btrieve format DMAM databases, the size is the number of bytes allocated to the schema file.

3.2.9 The OPTION and ON ERROR Statements

OPTION ERROR should be coded only if you wish your program to regain control following an irrecoverable I/O error. ON ERROR should be coded if you wish to handle certain I/O errors specially. The processing of these statements is common to all file organisations, and is described in the Global Development File Management Manual.

3.2.10 The ON FAILURE Statement

The ON FAILURE statement should be used if you wish to build a centralised failure handling routine into your program. Due to the complexity of database processing it is not practical to reflect irrecoverable I/O errors which arise during index updating back to the calling program. Instead, when such an error does arise the failure routine will be called, to permit your program to display a helpful message and to fail in a controlled way.

If you do not establish a failure routine, and a serious error arises, then your program will be terminated with a STOP CODE.

When the failure routine is called the accumulator contains the STOP CODE number (from Appendix A) which identifies the reason for the failure. If you wish to use this information to display an explanatory message to the operator, possibly suggesting a suitable course of recovery, then you should begin your failure routine with the statement:

```
$STORE fail-code
```

to store the failure code in the PIC 9(4) COMP variable fail-code.

For C-ISAM format DMAM databases, if a C-ISAM error is the primary cause of a STOP CODE then the C-ISAM UCI error code will be returned in the $$CRES PIC 9(9) COMP system variable. The C-ISAM UCI error codes are fully documented in the Global File Converters Manual.

For Btrieve format DMAM databases, if a Btrieve error is the primary cause of a STOP CODE then the Btrieve UCI error code will be returned in the $$CRES PIC 9(9) COMP system variable. The Btrieve UCI error codes are fully documented in the Global File Converters Manual.

Important note 1: Under no circumstances should control be returned from your failure routine to the Access Method. Your failure routine must terminate by issuing either a STOP RUN, CHAIN or RUN statement.

Important note 2: If your program is running as a user system request, where failing with a STOP CODE may damage the underlying application program, a special technique is available which may be used to return control to the system request handler in a controlled way after a DMAM failure has been returned. This technique is described in the Global Development System Subroutines Manual.

3.3 The OPEN Statement

The OPEN statement is coded:

```
OPEN type filename [index-area]
```

where type is the word OLD or SHARED and filename identifies the DMAM database file definition. The index-area is optional. When specified it is the name of a 516-byte work area in which the last index block used will be held. You supply this parameter if you wish to improve the performance of READ NEXT and READ PRIOR operations by eliminating index searching. The index area may be shared between several DMAM FDs, although it will only serve to optimise performance for the last FD used at any stage. It may not be used for any other purpose whilst the files using it remain open, otherwise unpredictable errors may occur.

The index area is only of use for Global format DMAM databases. It is of no value for C-ISAM format and Btrieve format DMAM databases.

An OPEN statement must be executed before any other file processing statement. If an OPEN is attempted but the FD is already open your program will be terminated in error.

OPEN NEW is **not** supported, because a DMAM database file is always created using DBMAIN as explained in 3.1.3. OPEN OLD obtains exclusive access to the file (allowing only one index to be used at a time) whilst OPEN SHARED allows co-operating jobs running under a multi-user system to share a DMAM database file. The features of the open operation which are common to all file organisations, such as volume-id checking, are described in detail in the Global Development File Management Manual.

3.3.1 File Conditions

The **file not found** ($$RES = "3") or **wrong type** ($$RES = "1") condition is returned if a file with DMAM database organisation and the same

file-id as that specified in the FD is not present on the direct access volume. The **index not present** ($$RES = "2") condition is returned if the index whose name was specified is not defined in the database file.

3.3.2 Exception Conditions
The **index needs rebuild** exception ($$COND = 3) is returned if any index associated with the selected record set needs to be rebuilt.

3.3.3 Successful Completion
Providing no file condition or irrecoverable I/O error occurs, the OPEN operation completes successfully. The index specified in the INDEX statement is found, and access to the record set to which it applies is made possible via the appropriate key. The file size and record length are returned in the FD and made available to the user program if SIZE and RECORD LENGTH statements with the symbol option were coded.

3.3.4 Programming Notes
For C-ISAM format DMAM databases, the Global schema file is opened as described above. The C-ISAM file for the specified record set will also be opened (or marked as opened) by the C-ISAM Universal Channel Interface (UCI). The C-ISAM file will not be opened exclusively even if an OPEN OLD is issued.

For Btrieve format DMAM databases, the Global schema file is opened as described above. The Btrieve file for the specified record set will also be opened (or marked as opened) by the Btrieve Universal Channel interface (UCI). The Btrieve file will not be opened exclusively even if an OPEN OLD is issued.

3.4 The WRITE Statement
WRITE is used to add a new record. It is coded:

```
WRITE filename FROM A
```

Here filename identifies the file definition and A is a simple or indexed variable, or a literal. If WRITE is attempted on an FD which is not already open the program will be terminated in error.

3.4.1 File Conditions
DMAM attempts to add the new record to the file and to insert entries for the various keys into the appropriate indexes. The **key already exists** condition ($$RES = "4") will be returned if any unique key on the record already exists in its index. Either a STOP CODE (see Appendix A) or the **file space exhausted** condition ($$RES = 5) will be returned if there is insufficient space in the file to contain the new record.

3.4.2 Successful Completion
Providing that no file condition or irrecoverable I/O error occurs, WRITE will add the new record to the database. The number of bytes transferred from the record at A to direct access storage will not depend on the picture clause of A but will be equal to the record length of the record set, defined when it was created.

3.4.3 Programming Notes

When a record which already exists is to be updated the REWRITE statement must be used. If some unique key has been changed then it must instead be deleted using the DELETE statement and then added using WRITE.

If a record to be added has a record type which begins with an asterisk, then the WRITE operation will complete successfully but no activity will take place on the database (the deleted record is ignored).

If the database becomes full when attempting to create a new index block, due to an index split having occurred, the failure routine will be entered if one has been established, otherwise the program will be terminated with a stop code.

3.5 The REWRITE Statement

REWRITE is used to update the record last accessed from a file. It is coded:

```
REWRITE filename FROM A
```

Here filename identifies the DMAM database file definition and A is a simple or indexed variable or literal. If REWRITE is attempted on an FD which is not already open the program will be terminated in error.

3.5.1 Key Value Checking

DMAM checks that the values of any unique keys supplied in A are the same as those of the record to be updated. If this is not the case the file operation will be suppressed and the **unique key changed** condition ($$COND = 4) is generated. Unique keys cannot be changed by a REWRITE operation: to modify unique keys the record must be DELETED, and then added with a WRITE operation.

3.5.2 Successful Completion

Providing no unique keys have been modified, and no irrecoverable I/O error occurs, REWRITE will update the record last accessed. The number of bytes transferred from the record at A to direct access storage will not depend on the picture clause of A but will be equal to the record length of the file, defined when it was created.

3.5.3 Programming Notes

REWRITE avoids updating any indexes whose keys have not been changed on the record processed. It also performs a quick check to determine if the record is unchanged up to the end of the last field used in any index, and avoids rebuilding all the key information if this is the case. Therefore in the interests of efficiency you are advised to group all fields used as part of keys at the start of the record data area, and to place the variable data after all such key information, wherever possible.

To maintain compatibility with ISAM, if the first character of the record type field is set to an asterisk (*) before issuing the REWRITE operation, then this is treated as if a DELETE operation had been issued as documented in section 3.6, except that it is not absolutely necessary to have obtained a DELETE-LOCK first. Nevertheless you are always advised to obtain a DELETE-LOCK before deleting a record, to ensure consistency of information in the database.

3.6 The DELETE Statement

The DELETE statement is used to delete the last record processed. You code:

```
DELETE filename
```

where filename identifies the DMAM database file definition. If the record you are deleting has not been locked with a DELETE-LOCK by a preceding READ type statement or if the FD is not already open then your program will be terminated in error.

3.6.1 File conditions

No file conditions can arise as a result of a DELETE statement. As the statement also requires that a DELETE-LOCK has been obtained on the record being deleted there should be no possibility of the record having been changed, but if this should occur then your program will be terminated in error. Such an error is only likely to arise if some program is deleting records using the ISAM compatible function of REWRITE (placing an asterisk in the first character of the record type) without any locking protecting the operation.

3.6.2 Successful completion

Providing that no irrecoverable I/O error occurs the record will be deleted from the database file, and the DELETE-LOCK will be released.

3.7 The READ Statement

READ is used to retrieve a record with a given key from the file. If the key is not present the operation attempts to retrieve the record whose key is immediately higher than the one specified.

The READ statement is coded:

```
READ filename INTO A [lock-type]
```

Here filename identifies the DMAM database file definition, A is a simple or indexed variable and lock-type is one of LOCK, PROTECT or DELETE-LOCK as specified in section 3.1.10. If READ is attempted on an FD which is not already open the program will be terminated in error.

3.7.1 Establishing the Key

The key is established by setting up the various elements of the key within the record area before the READ statement is executed. The key elements will be replaced by different values if the **record not found** file condition occurs.

3.7.2 File Conditions

The **record not found** file condition will be returned if a record with the key you have specified is not present on the file. In this case DMAM will retrieve the record whose key is next higher in collating sequence than the one you specified, if such a record exists.

If the key you supplied was greater than any key currently stored, the file condition will again be returned and the record area will be filled with high-values (to simulate ISAM dummy high record handling). Note that if the key you supply is present on the file, DMAM simply returns the appropriate record to you **without** returning a file condition.

If the lock-type clause was coded DMAM will attempt to obtain the appropriate lock for you (see section 3.1.10). If this is not possible, for example due to some other user already having a competing lock, then the **lock unavailable** condition ($$COND = 3) is returned. The record will still be retrieved, but will not have been locked.

Note that if the **record not found** condition has been returned then the record will not be locked, regardless of any lock option which might have been specified.

3.7.3 Successful Completion

Providing no permanent I/O error occurs and a key greater than or equal to the one you specified exists on the file, READ will transfer bytes from the record thus identified to A. The number of bytes transferred does not depend on the picture clause associated with A, but will be equal to the record length of the file defined when it was created.

3.8 The READ NEXT and READ PRIOR Statements

READ NEXT and READ PRIOR are used to process part or all of a file sequentially. READ NEXT processes keys in ascending sequence, and READ PRIOR processes keys in descending sequence. They are coded:

```
     READ NEXT filename [KEY LENGTH length] INTO A [lock-type]
and:
     READ PRIOR filename [KEY LENGTH length] INTO A [lock-type]
```

Here filename identifies the DMAM database file definition, length is an integer literal in the range 1 to 63, A is a simple or indexed variable and lock-type is one of LOCK, PROTECT or DELETE-LOCK as specified in section 3.1.10. If a READ NEXT or READ PRIOR is attempted on an FD which is not already open the program will be terminated in error.

3.8.1 The Record Retrieved

The record retrieved by READ NEXT or READ PRIOR depends on the previous file operation:

- A READ NEXT following OPEN will retrieve the first record of the record set. If there are no records in the record set **end of file** will be returned;

- A READ NEXT following a READ, READ NEXT, READ FIRST, READ LAST, READ PHYSICAL, WRITE, REWRITE or DELETE will retrieve the record immediately higher in collating sequence than the one just read processed. If there is no such record **end of file** will be returned;

- A READ PRIOR following a READ, READ NEXT, READ FIRST, READ LAST, READ PHYSICAL, WRITE, REWRITE or DELETE will retrieve the record immediately lower in collating sequence than the one just processed. If there is no such record **start of file** will be returned;

- A READ PRIOR following an OPEN will return **start of file** immediately.

Where a KEY LENGTH clause was specified, the partial key established serves to delimit the range of the READ NEXT or READ PRIOR operation. If the key of the record retrieved does not match the partial key established in the record area the **key match** condition will be returned.

Note that if the key length specified by the KEY LENGTH clause includes only part of a numeric key as the last segment addressed by the key length, then the record returned will be unpredictable.

3.8.2 File Conditions

The **end of file** condition ($$COND = 2) will be returned in response to READ NEXT if the record last accessed was the record with the highest key, or if the last file operation encountered end of file. When this condition occurs the record area is filled with high-values. The **start of file** condition ($$COND = 2) will be returned in response to READ PRIOR if the last record accessed was the record with the lowest key, or if the last file operation encountered start of file. Additionally, the **key check** condition ($$COND = 2) will be returned if a partial key was specified by the KEY LENGTH clause and the partial key established in the record area prior to the READ NEXT or READ PRIOR did not match the start of the key of the record retrieved. In this latter case the next record (in the case of READ NEXT) or previous record (in the case of READ PRIOR) will be retrieved.

If the lock-type clause was coded DMAM will attempt to obtain the appropriate lock for you (see section 3.1.10). If this is not possible, for example due to some other user already having a competing lock, then the **lock unavailable** condition ($$COND = 3) is returned. The record will still be retrieved, but will not have been locked.

Note that if the **key match** condition has been returned then the record will not be locked, regardless of any lock option which might have been specified.

3.8.3 Successful Completion

Providing no irrecoverable I/O error occurs, bytes will be transferred from the file to A. The number of bytes transferred will not depend on the picture clause associated with A, but will be equal to the record length of the file, defined when it was created.

3.9 The READ FIRST and READ LAST Statements

READ FIRST and READ LAST are used to retrieve the first or last record of a record set. They are coded:

```
     READ FIRST filename [KEY LENGTH length] INTO A [lock-type]
and:
     READ LAST filename [KEY LENGTH length] INTO A [lock-type]
```

Here filename identifies the DMAM database file definition, length is a numeric literal in the range 1 to 63, A is a simple or indexed variable and lock-type is one of LOCK, PROTECT or DELETE-LOCK as specified in section 3.1.10. If a READ FIRST or READ LAST is attempted on an FD which is not already open the program will be terminated in error.

3.9.1 The Record Retrieved

The record retrieved by READ FIRST or READ LAST depends on partial key specified by the KEY LENGTH clause. If no partial key is specified either the very first, or very last, record of the record set is retrieved. If a partial key is specified the first, or last, record in the record set whose key start matches the partial key is retrieved.

Note that if the key length specified by the KEY LENGTH clause includes only part of a numeric key as the last segment addressed by the key length, then the record returned will be unpredictable.

3.9.2 File Conditions

The **end of file** condition ($$COND = 2) will be returned in response to READ FIRST if there are no records in the record set. In this case the record area will be filled with high-values to simulate the dummy high record handling of ISAM. The **start of file** condition ($$COND = 2) will be returned in response to READ LAST if there are no records in the record set. Additionally the **key check** condition ($$COND = 2) will be returned if there is no record in the record set whose key start matches the partial key specified in the record area. In this case the record will **not** be returned, and the record area will remain unaffected by the operation.

If the lock-type clause was coded DMAM will attempt to obtain the appropriate lock for you (see section 3.1.10). If this is not possible, for example due to some other user already having a competing lock, then the **lock unavailable** condition ($$COND = 3) is returned. The record will still be retrieved, but will not have been locked.

3.9.3 Successful Completion

Providing no irrecoverable I/O error or key match condition occurs, bytes will be transferred from the file to A. The number of bytes transferred will not depend on the picture clause associated with A, but will be equal to the record length of the file, defined when it was created.

3.10 The READ PHYSICAL Statement

READ PHYSICAL is used to retrieve a record with a given record address (for Global format DMAM databases) or a record with a given record number (for C-ISAM format and Btrieve format DMAM databases) from the file. You code:

```
READ PHYSICAL filename INTO A [lock-type]
```

Here filename identifies the DMAM database file definition, A is a simple or indexed variable and lock-type is one of LOCK, PROTECT or DELETE-LOCK as specified in section 3.1.10. The record address (or record number) must have been established in the area defined in the RECORD ADDRESS clause prior to the READ PHYSICAL operation being issued. If READ PHYSICAL is attempted on an FD which is not already open the program will be terminated in error.

3.10.1 File conditions

If the READ PHYSICAL attempts to return a deleted record then the **record deleted** condition ($$COND = 2) is returned.

If the lock-type clause was coded DMAM will attempt to obtain the appropriate lock for you (see section 3.1.10). If this is not

possible, for example due to some other user already having a competing lock, then the **lock unavailable** condition ($$COND = 3) is returned. The record will still be retrieved, but will not have been locked.

3.10.2 Successful completion

Providing no irrecoverable I/O error occurs, bytes will be transferred from the file to A. The number of bytes transferred will not depend on the picture clause associated with A, but will be equal to the record length of the file, defined when it was created.

3.10.3 Programming notes

When a READ PHYSICAL operation is performed, the data is transferred from the physical record address established by the calling program. No checking is performed to ensure that this is in fact a record boundary, it being assumed that the calling program has saved a record address correctly. It is also the responsibility of the calling program to perform suitable validation checks on the record retrieved before processing it, as it may have been deleted, reused or updated since its address was saved.

Normally after a file processing statement has completed the RECORD ADDRESS field will contain the address (or record number) of the record last processed. However after a **start of file** condition has been returned (and immediately after an OPEN operation) the RECORD ADDRESS field will be zero, and after an **end of file** condition the RECORD ADDRESS field will contain -1. Programs should not attempt a READ PHYSICAL with either of these record addresses.

It is generally inappropriate to keep a physical record address from one run of a program to another, as any reorganisation of the database which might be performed between the two program runs would certainly invalidate the address.

An example of a situation where a READ PHYSICAL might be useful is in a sort. Records would be read from the database using READ NEXT, and the sort key constructed from them, with the record address appended as in a tag sort. When the sorted records are returned READ PHYSICAL is used to retrieve the records, thereby avoiding any index processing and speeding the whole process considerably.

It should be noted that a record may have been updated between these two accesses, or deleted, or even deleted and reused for a completely different record. It is therefore essential that either the entire process is carried out so that no such interference can occur (by having the file OPENed OLD, for example), or that the returned record is checked to ensure its validity (the sort ensuring that it still matches the selection criteria, and that the sort key in the record matches that in the tag sort, i.e. that none of the sort fields have been updated). In practice the former solution is normally impractical for various reasons. If a record has been updated so as to no longer be valid, you will probably have to ignore it.

3.11 The UNLOCK Statement

The UNLOCK statement is used to release any locks outstanding from a previous READ type operation using a lock-type clause. You code:

```
UNLOCK filename
```

where filename identifies the DMAM database file definition. If an UNLOCK is attempted on an FD which is not already open the program will be terminated in error.

3.11.1 Successful completion

Any outstanding lock on the file due to a previous READ type operation on the FD passed to the UNLOCK is released.

Note particularly that any lock gained through use of the LOCK verb is **not** released by this UNLOCK statement, but must be released by using an UNLOCK filename region statement as documented in the Global Development File Management Manual.

3.12 The CLOSE Statement

CLOSE must be used to terminate the processing of a file. It is coded:

```
CLOSE filename [TRUNCATE|DELETE]
```

where filename identifies the DMAM database file definition. If a CLOSE is attempted on an FD which is not already open the program will be terminated in error.

3.12.1 Standard Processing

CLOSE always completes any outstanding I/O operations on the file and returns the FD to the status it possessed prior to being opened. Following CLOSE, the FD can be re-opened for the same, or a different, DMAM database file using the same, or a different, index by a subsequent OPEN statement.

3.12.2 Truncation

If the TRUNCATE phrase is coded, DMAM will return the unused part of the overflow area, if any, to the volume so that it can be re-allocated.

3.12.3 Deletion

If the DELETE phrase is coded all the space the file occupies is returned to the volume and its file-id is erased from the directory. Following a CLOSE DELETE the file no longer exists.

Warning: A CLOSE DELETE will delete the entire database file. This option should be used with extreme caution.

3.12.4 Programming Note

WRITE, REWRITE and DELETE operations on a DMAM database file are effected immediately. Should a computer fail whilst attempting one of these operations a subsequent OPEN will notice this, and will undertake suitable recovery action to ensure data integrity and consistency. This feature, which is only available with Global format DMAM databases, is intended to protect files which are updated interactively from damage in event of machine failure.

The CLOSE statement on a DMAM database releases any locks outstanding on the particular FD which were gained through use of the DMAM LOCK option (see section 3.1.10) on a READ type statement. CLOSE does **not** release any other locks (including locks on other FDs, or locks gained through the use of the LOCK verb), unless the CLOSE releases the last

active channel on the database file. In practice this means you should exercise extreme caution in using the LOCK verb with a DMAM database. Always ensure that any locks you obtain using the LOCK verb are explicitly released using an UNLOCK verb.

3.13 ISAM Compatibility

One of the aims of the DMAM file organisation is to provide a straightforward upgrade path for an existing program using ISAM files. To this end the processing of a record set which has only a single unique key is as close as practical to that of ISAM. The following differences in the interface exist, and should be noted by program developers who are planning such an upgrade:

- The structure of the DMAM FD is quite different to that of ISAM. Most notably the index name is what would have been the file name with ISAM (and the database file name is probably the same for a whole program suite), so this must be set up differently. The DMAM FD is also larger than the ISAM FD (248 bytes compared to 96).

- The syntax of the OPEN operation is different, as the index area passed to optimise READ NEXT and PRIOR processing is 516 bytes long (rather than 256), and there is no particular advantage to passing this area for random READ and REWRITE processing.

- The WRITE statement cannot be used to update an existing record with DMAM, as it could with ISAM. This could be a problem with create/amend type programs which use common code to write the new or updated record to the file, but such processing is often the cause of serious program errors, and the decision to restrict the WRITE handling in DMAM has been taken to eliminate such problems.

- There is a new DELETE statement which ideally should be used to delete records, although the ISAM compatible REWRITE with asterisk in the record type is still supported. However, deleted records cannot be processed by the application. This is no problem if OPTION IGNORE or some similar processing was already being performed, but any program which was relying on its ability to read a deleted record will need to be redesigned.

- READ and READ NEXT operations which encounter end of file simulate the ISAM dummy high record processing by filling the record area with high-values. Although this is not precisely the same processing, it should be sufficiently close not to cause any serious problems.

- The DMAM access method is considerably larger than ISAM. Refer to Appendix B for details concerning the size of DMAM.

In practice anyone upgrading a program will almost certainly wish to take advantage of the extra file processing statements available with DMAM (READ PRIOR, READ FIRST, READ LAST and the lock options), and will very likely add extra non-unique keys to the records. Nevertheless the basic structure of existing programs can be easily modified to utilise DMAM.

3.14 Data Management Access Method Variations

The V8.1 Global Cobol Data Management system includes three variations of the Data Management Access Method (DMAM):

- Non Memory Page sub-routines for Global format DMAM databases only (see 3.14.1);
- Memory Page sub-routines for Global format DMAM databases only (see 3.14.2);
- Memory Page sub-routines for Global format, C-ISAM format and Btrieve format DMAM databases (see 3.14.3).

The variation of DMAM used by an application is determined at link-time.

3.14.1 Default DMAM

The version of DMAM linked by default into an application will be the non-Memory Page sub-routine AM$A which is only capable of accessing Global format DMAM databases.

This version includes all the DMAM code within the subroutine and will operate on all versions of Global System Manager from V7.0 onwards. If you wish to use any other variation of DMAM the subroutine must be linked explicitly (see sections 3.14.2 and 3.14.3).

3.14.2 Memory Page Global-only DMAM

Memory Page Global-only DMAM will only operate on Global System Manager V8.1, or later.

When this DMAM variation is linked into an application most of the access method code is held within a Global System Manager memory-page. Thus, the effective size of the sub-routine is very much reduced (see appendix B).

This DMAM variation is only capable of accessing Global format DMAM databases.

In order for your application to use Memory Page Global DMAM you must explicitly link module AM$Z into your code. This is simply achieved by specifying library C.$PAGES when linkage editing using $LINK. For example:

```
$44 LINK:DMAPPN  UNIT:210
$44 LINK:C.$PAGES  UNIT:202
$44 LINK:<CR>
```

If the additional libraries C.$MCOB or C.$APF are included in the linkage edit, they must be specified AFTER C.$PAGES. The C.$PAGES compilation library is distributed as part of the Global Cobol Development system.

3.14.3 Memory Page Open DMAM

Memory Page Open DMAM will only operate on Global System Manager V8.1, or later.

When this DMAM variation is linked into an application most of the access method code is held within a Global System Manager memory-page. Thus, the effective size of the sub-routine is very much reduced (see appendix B).

This version of DMAM can be used to access Global format, C-ISAM format and Btrieve format DMAM databases. This is the only DMAM variation that is capable of accessing C-ISAM format or Btrieve format DMAM databases.

C-ISAM format DMAM databases are only supported on Global System Manager (Unix) (i.e. if the "Machine family code" as displayed by $S is "C2"). However, not all Global System Manager (Unix) configurations include the C-ISAM UCI. The Global Configuration Notes indicate whether the C-ISAM UCI is supported on a particular configuration.

Btrieve format DMAM databases are only supported on Global System Manager (MS-DOS and Windows) and Global System Manager (Novell NetWare) (i.e. if the "Machine family code" as displayed by $S is "JW"). However, not all Global System Manager (MS-DOS and Windows) or Global System Manager (Novell NetWare) configurations include the Btrieve UCI. The Global Configuration Notes indicate whether the Btrieve UCI is supported on a particular configuration.

All file access statements are as described earlier in this chapter. The creation and maintenance of C-ISAM format and Btrieve format DMAM databases is described in chapter 5.

In order for your application to use Memory Page Global DMAM you must explicitly link module AX$Z into your code. This is simply achieved by specifying module AX$Z in library C.$PAGES when linkage editing using $LINK. For example:

```
$44 LINK:DMAPPN  UNIT:210
$44 LINK:C.$PAGES/AX$Z  UNIT:202
$44 LINK:C.$PAGES  UNIT:202
$44 LINK:<CR>
```

If the additional libraries C.$MCOB or C.$APF are included in the linkage edit, they must be specified **AFTER** C.$PAGES. The C.$PAGES compilation library is distributed as part of the Global Cobol Development system.

4. Further Data Management File Handling

This chapter describes the various system routines which are available as part of the Global Cobol Data Management system.

Each system routine performs a specific function. If the functionality of a DMAM system routine is analogous to that of an ISAM system routine, the programming notes will explain the similarities.

4.1 Set up for use of DMAM, DBSET$

For Global System Manager V6.0, it is **absolutely essential** that the DBSET$ routine is called by a program, or suite of programs, before DMAM itself is used. It is responsible for ensuring that a suitable **key build routine**, which is used to create all keys used by DMAM, is established with its entry point in the Global System Manager SVC 63 pointer field.

4.1.1 Invocation

To set up for use of DMAM you code:

```
CALL DBSET$
```

4.1.2 Exceptions

Exception condition 1 is returned if DBSET$ is unable to attach a suitable key build routine. This will always occur on a V5.0 or earlier system. It will also occur on a V5.1, V5.2 or V6.0 system if DBSET$ is unable to load the assist file containing the key build routine. This file is conventionally named "%.a0V63 ", where a is the computer architecture code, and should be located on SYSRES. An attempt to load a Global Cobol version of the routine (named "%.00V63 ") is made if no assembler-specific component can be found. Failure to load the key build routine can occur either because it cannot be found, or because there is insufficient space in the user stack for it to be loaded.

4.1.3 Programming Notes

It is unnecessary to call DBSET$ on Global System Manager V6.1, or later, ($$VERS > 3) as the key build routine will already be present in the loaded nucleus. DBSET$ will however detect this situation and avoid loading a second copy of the routine.

If a copy of the key build routine is loaded into the system stack (by a Load Customisation for example) then DBSET$ will use this rather than loading another copy into the user stack.

Copies of the assist files are present on your Global Cobol Development distribution diskette (MKA). You should ensure that appropriate assist files are copied to a pre-V6.1 SYSRES before attempting to run a program which will use DMAM. Note, in particular, this copy procedure is NOT necessary for Global System Manager V8.1.

DMAM will not run on a V5.0, or earlier, system, so DBSET$ will always return an exception on such a system.

If you wish to unload the key build routine using UNLO$ you must first call it with no parameters to cause it to detach itself from Global System Manager.

4.2 The DMAM index rebuild routine, IRBLD$

The IRBLD$ routine is used to rebuild an index (or a series of indexes associated with a single record set) in situ on a DMAM database file.

4.2.1 Invocation

To rebuild an index, or group of indexes you use a CALL statement of the form:

```
CALL IRBLD$ USING filename sortfile [option]
```

The filename is the name of a DMAM file definition, which must be closed when the routine is called, and will remain closed when it returns control. The INDEX NAME must be established in the FD before IRBLD$ is called, to identify which index or record set is to be processed.

The sortfile is a sort work file suitable for use by the Global Cobol MSORT$ routine (documented in the Global Development File Management Manual), which is used by IRBLD$ when sorting the keys in the index. IRBLD$, when called on Global Cobol files, will pass to the sort a tag record consisting of the DMAM key and a 4-byte file address for each record in the set which does not contain an ignored key value. $CALC, or the algorithm for calculating sort sizes documented in the Global Development File Management Manual, may thus be used to estimate the memory and file size required for the sort (e.g. if an index contains 10,000 records with a total key length of 35 bytes, with 5K of free memory the sort would require 396K of work file).

The optional third parameter, option, is a PIC 9(4) COMP field or integer literal which should be non-zero if IRBLD$ is being used to rebuild all the indexes associated with a record set. If it is 1 all indexes which are marked as requiring rebuilding are rebuilt (an index is marked as requiring rebuilding whenever an index rebuild is initiated on it, in case the computer fails whilst it is in progress, and whenever the data take-on routines are used to add records without updating the index). If it is -1 all indexes are rebuilt. When the third parameter is omitted, or set to zero, then the index named in the INDEX NAME in the DMAM FD will be rebuilt.

4.2.2 Exception Conditions

Exception condition 1 will be returned if an irrecoverable I/O error occurs when attempting to OPEN the DMAM database file.

Exception condition 2 will be returned if the DMAM file cannot be found, or if the index specified by the INDEX NAME statement is not defined within the database.

Note that MSORT$ exceptions may also be returned from IRBLD$ (MSORT$ is documented in the Global Development File Management Manual).

4.2.3 Processing

If IRBLD$ is processing more than a single index, then each index is processed separately.

For the index being processed IRBLD$ first sets the flag in the index area to indicate that the index requires rebuilding, in case the reorganise should be interrupted for some reason.

For Global format DMAM databases all the index data blocks currently in use by the index are deleted. Each allocation extent of the file is examined, and if it contains records of the correct record set these are scanned and their keys are passed to the SORT. When the whole file has been processed the keys are sorted into order, and the index is then recreated from the returned key values. The index is rebuilt with an amount of spare space as determined from the index usage type set up for the index by the database maintenance utility, DBMAIN.

For C-ISAM format and Btrieve format DMAM databases the indexes are deleted and added again in effect doing a rebuild.

Once the index has been completely rebuilt, the flag in the index area is cleared and IRBLD$ either exits or determines the next index to process, as appropriate.

4.2.4 Programming Notes

Except under exceptional circumstances it should not be necessary to call IRBLD$ during normal working with a DMAM database. Large numbers of deletions, or extensive record addition may, however, make this desirable.

It is essential to call IRBLD$ to rebuild all indexes associated with a record after any records are added using the data take-on routines documented in section 4.5. If this is neglected, then any attempt to use the DMAM database will result in an exception from the OPEN statement, indicating that an index rebuild is required.

IRBLD$ provides some of the functions of CONV$ on an ISAM file, in that it reorganises the indexes. However IRBLD$ does not recover space occupied by deleted records or index data blocks, nor physically reorganise the layout of the records themselves.

4.3 The Database Statistics routines, DBSTA$ and DBCLC$

The database statistics routines are used to gain information about the data contained within a DMAM database. They each function on a single record set at a time, allowing a program to build up information about a series of record sets and hence to determine information about the database file as a whole.

Typically a program would begin by calling DBSTA$, to gain information about the current state of a record set. Then certain pieces of information, such as the number of records in use, would be updated and DBCLC$ would be called to determine the effect of the changes. This process would be repeated for each record set of interest, accumulating data on the effects so as to enable a sensible decision to be made at the conclusion of the processing.

4.3.1 The DBSTA$ Routine

The DBSTA$ routine is used to gain information about a record set within a DMAM database.

The DBSTA$ routine is invoked by a CALL statement of the form:

```
CALL DBSTA$ USING filename si
```

The filename is the name of a DMAM file definition, which must be open when the routine is called and will remain open when it returns control. The index specified by the INDEX NAME in the FD will determine for which record set information is returned.

The second parameter, si, defines a set information area in the format shown below, where the data will be returned.

4.3.2 The DBCLC$ Routine

The DBCLC$ routine is used in conjunction with DBSTA$ to make estimates of the amount of space required for a database. DBCLC$ is invoked by a similar call, of the form:

```
CALL DBCLC$ USING filename si
```

Again filename identifies a DMAM FD, and si a set information area, but for DBCLC$ the FD does not have to be open. DBCLC$ will recalculate the various file space fields in the SI block on the basis of the number of records set up, which will presumably have been altered by the calling program.

DBCLC$ makes use of certain fields within the DMAM FD, such as the record length, when calculating the file space occupied by a record set.

4.3.3 The SI Set Information Block

The set information block is laid out as follows:

```
01   SI
  02 SIFSIZ          PIC 9(9) COMP       * used file size
  02 SISID           PIC X(2)            * set identifier
  02 SISNAM    PIC X(30)           * set description
  02 SIALL           PIC 9(9) COMP       * allocation size
  02 SIRPA           PIC 9(9) COMP       * records/allocation
  02 SIIALL          PIC 9(9) COMP       * IDB allocation size
  02 SIIPA           PIC 9(9) COMP       * IDBs/allocation
  02 SINREC    PIC 9(9) COMP       * number/records in use
  02 SIFREC    PIC 9(6) COMP       * number of free records
  02 SINIDX    PIC 9(2) COMP       * number of indexes
  02 SITRSZ    PIC 9(9) COMP       * total record size
  02 SITNIB    PIC 9(6) COMP       * total number of IDBs
  02 SITISZ          PIC 9(9) COMP       * total index size
  02 SII OCCURS 16
    03    SIINAM    PIC X(8)            * index name
    03    SIITKL    PIC 9(2) COMP       * total key length
    03    SIIUNI    PIC 9 COMP              * unique flag
    03    SIIPCU    PIC 9(1,1) COMP         * amount used
    03    SIILEV    PIC 9(2) COMP       * index levels in use
    03    SIINIB    PIC 9(6) COMP       * no. IDBs in index
    03    SIISIZ         PIC 9(9) COMP       * index size
```

4.3.4 Information returned by DBSTA$

When DBSTA$ completes normally the routine returns the information in the SI block as indicated below. Fields marked with a [1] are returned by DBSTA$, and used by DBCLC$ in its calculations. Fields marked with a [2] are always calculated whenever DBSTA$ or DBCLC$ are called. Unmarked fields are returned by DBSTA$ and not used by DBCLC$.

SIFSIZ The total used size of the file (i.e. the amount actually allocated for use as records and index blocks). The free space in the file is therefore the total file size (from the SIZE statement in the FD) less SIFSIZ. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SISID The set identifier of the currently selected record set.

SISNAM The long description of the currently selected record set.

SIALL[1] The size of the allocation extent of the currently selected record set. In conjunction with SIRPA this is used to determine the file space occupied by the record set. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIRPA[2] The number of records which will fit into each extent allocation. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIIALL[1] The allocation extent used for index data blocks (IDBs). For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIIPA[2] The number of IDBs which will fit into each extent allocation. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SINREC[1] The number of records in use for the currently selected record set.

SIFREC[1] The number of records in the free record chain for the currently selected set, including any unused records from the last extent allocated. This many records may be added to the record set without requiring any further record extents to be allocated. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SINIDX[1] The number of indexes defined for the currently selected record set. There will be one occurrence of the SII data area for each index defined.

SITRSZ[2] The total size occupied by the records in the record set.

SITNIB[2] The number of IDBs in use by all the indexes on the record set if they were fully reorganised (sum of SIINIB for all indexes). For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SITISZ[2] The total size required by the number of IDBs given in SITNIB above. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

The following data fields occur once for each index on the record set, as specified in SINIDX. The data in occurrences greater than SINIDX is unpredictable.

SIINAM The name of the index.

SIITKL[1] The total length of the key which is contained in the index. For C-ISAM format and Btrieve format DMAM databases, the Global key length will be returned in this field.

SIIUNI[1] A flag indicating whether keys in the index must be unique or not. 0 indicates that keys must be unique. 1 indicates that keys may be non-unique. 2 indicates that keys may be non-unique and that keys of either spaces or low-values (or both) are omitted from the index.

SIIPCU[1] A field indicating how much of each index block is used when the index is reorganised. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIILEV The number of levels in the index (i.e. how many IDBs need to be read before a record may be retrieved when reading the file). For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIINIB[2] The number of IDBs that would be used by this index if it were reorganised. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

SIISIZ[2] The amount of file space required to hold the number of IDBs indicated by SIINIB. For C-ISAM format and Btrieve format DMAM databases, 0 will be returned in this field.

4.3.5 Exception Conditions

Exception condition 1 is returned if an overflow occurs while attempting to calculate the various file size fields. If such an exception does occur the contents of the SI block are unpredictable.

4.3.6 Programming Notes

DBSTA$ only returns information for a single record set, so if you are attempting to gather information for allocating a complete database you will need to call DBSTA$ for each record set to be contained within the database. Similar processing is required if you are considering a file reallocation using calls of DBCLC$.

Normally you should call DBSTA$ before calling DBCLC$, to establish the various parameters which are held within the database file. You will typically wish to change the value of SINREC before calling DBCLC$. If the value of SIFREC is large you should reduce this field by the same amount you increase SINREC by, to avoid overestimating the file space occupied by the records. Never set SIFREC to a value less than zero.

When DBCLC$ is called it assumes that the fields set up by DBSTA$ have been correctly established, and proceeds to calculate the other fields as detailed above. First the values of SIRPA and SIIPA are deduced from the values of SIALL and SIIALL. Then the value of SITRSZ is deduced from the values of SINREC and SIFREC, along with SIRPA and SIALL and the record length established in the DMAM FD. For each index the values of SIINIB and SIISIZ are deduced from SINREC, SIALL, SIIALL, SIITKL and SIIPCU, and as a result the value of SITNIB is determined. From this the value of SITISZ may be calculated.

When performing the calculation of the size of an index where some key values are omitted (SIIUNI = 2), DBCLC$ assumes that 20% of the records will appear in the index. If your application has information that renders this estimate inaccurate you must compensate by adjusting the calculated index size, and the total index size, by a suitable amount.

The file space fields returned by a call on DBSTA$ or DBCLC$ are **idealised** values, which would be accurate if the indexes were freshly reorganised and the allocation extent sizes had never been altered. It is entirely likely that the summation of the various record and index sizes for all record sets within a database would yield a different value than the total used file size, and any estimate of the current used file size for file reallocation purposes should use the greater of such a summation and the value of SIFSIZ.

Nevertheless, **differences** between the current size returned by DBSTA$ and the size in some other situation as determined by DBCLC$ should be sufficiently accurate to permit sensible estimation of file sizes for allocation or reallocation purposes.

When record and index sizes are calculated, they are rounded up to a whole number of allocation extents. The size of each individual index is rounded, but the total index size is a rounding of the space required for the total number of IDBs contained in all the indexes.

If you wish to make similar allowances in any revisions to the returned figures that you may make, you should consult Appendix C to determine how the values are calculated.

As the figures returned by DBCLC$ are idealised mid-range estimates of the file size required to hold the indicated number of estimates, they may not be the best basis on which determine the size of a new file which is intended to be large enough to hold a given number of records (DBCLC$ cannot return worst case estimates, as CALC$ does for ISAM files, because the size of file required to hold the data in the worst case might be over one hundred times greater than the mid-range estimate). To be reasonably sure that a file will be large enough to hold a given number of records, you should first calculate the record space required using the correct number of records, and then calculate the index space required with the number of records increased by 30%. The resulting figure will give a file size large enough to hold the indicated number of records over 99% of the time.

In any case, when using DBCLC$ to calculate file sizes, you should beware of sizes returned which are close to a boundary requiring an extra allocation extent for either IDBs or records. If the boundary is actually crossed when the file is used, then the resulting file will be larger than the estimated value (possibly much larger in the case of a file containing only a small number of records). It is generally

a good idea to allow a spare allocation extent for IDBs (at least) when estimating the size of a file which will contain less than a thousand records, regardless of whether the above estimating techniques have been used. For files with larger numbers of records, the above technique is recommended in any case, and will yield superior results to merely increasing the file size by a fixed amount.

4.4 The Index Key Segment subroutine, DBKES$

The DBKES$ subroutine is used to obtain details of the key segment structure for the indexes in a database. The subroutine functions on a single record set at a time, allowing the program to obtain key information about a series of record sets and hence determine the key information of the database as a whole.

4.4.1 Invocation

The DBKES$ routine is invoked by a CALL statement of the form:

```
CALL DBKES$ USING filename sg
```

The filename is the name of a DMAM file definition, which must be open when the routine is called and will remain open when it returns control. The index specified by the INDEX NAME in the FD will determine which record set information is returned.

The second parameter, sg, defines the key information area defined as follows:

```
01   SG
  02 SGNIDX    PIC 9(2) COMP  * Number of used index in record
set
  02 SGINX OCCURS 16                * For each possible index
    03    SGINAM     PIC X(8)       * Index name
    03    SGITKL     PIC 9(2) COMP * Total key length
    03    SGKINX     PIC 9(2) COMP * Index into key table of the
first key segment
                                    * of this index
  02 SGKSEG OCCURS 64               * Key segment data
    03    SGKSTY     PIC X(2)       * Segment type
    03    SGKSOF     PIC 9(4) COMP * Key offset
    03    SGKSLE     PIC 9(2) COMP * Segment length
    03    SGKSTL     PIC 9(2) COMP *  Translation length; 0 = no
translation
```

4.4.2 Information returned by DBKES$

When DBKES$ returns without an exception the routine returns the information in the sg block as follows:

SGNIDX This is the number of used indexes in the record set, and hence the number of used entries in the SGINX table.

SGINAM The index name.

SGITKL The total length of the key which is contained in the index.

SGKINX The index into the key segment table, SGKSEG, of the first segment of this index.

SGKSTY The key segment type (e.g. "CA" computational ascending).

SFKSOF The offset from the start of the record of this key segment.

SGKSLE The length of this key segment.

SGKSTL The translation length of this key segment. This length will be 0 if there is no translation for this key segment.

4.4.3 Programming notes

DBKES$ only returns information for a single record set, so if you are attempting to gather information about the complete database you will need to call DBKES$ for each record set to be contained within the database.

Generally the DBKES$ routine will be used in verifying that the correct version of a particular database is being processed. This subroutine can also be used to check if any unauthorized amendment of the database has occurred.

4.5 The Data Take-on routines, DTOPN$, DTWRT$ and DTCLS$

The data take-on routines are provided to enable a program to load a large number of records into a DMAM database without the overhead of adding index entries and splitting index blocks. They are principally of interest in data conversion programs which transfer large amounts of data from an existing system into a DMAM database file.

To use the data take-on routines, you must use DTOPN$ to open the DMAM database, followed by repeated calls on DTWRT$ to add the required records. When all the records have been written you close the database using DTCLS$, and you would then normally use IRBLD$, documented in section 4.2 of this manual, to build the various indexes for the records added.

4.5.1 Open Database for Data Take-on, DTOPN$

The DTOPN$ routine is called to open the DMAM database for data take-on processing. You code:

```
CALL DTOPN$ USING filename
```

The filename is the name of a DMAM file definition, which must be closed when the routine is called and will be open when it returns successfully. The index specified by the INDEX NAME clause in the FD will determine which record set is made available for addition. The database is opened as if an OPEN OLD statement had been coded, so no other program can use the database file while data take-on is in progress.

4.5.2 Writing a record, DTWRT$

The DTWRT$ routine is used to write a record to the database without updating any indexes. You code:

```
CALL DTWRT$ USING filename record
```

The filename is the name of a DMAM file definition which must have been previously opened by a call of DTOPN$. The record identifies the record which is to be written to the database.

4.5.3 Terminating Data Take-on, DTCLS$

The DTCLS$ routine is used to indicate that you have finished data take-on, and want to close the database file so that it may be used for another purpose. You code:

```
CALL DTCLS$ USING filename
```

The filename is the name of a DMAM file definition which must have previously been opened by a call of DTOPN$, and which will be closed when the routine returns control.

4.5.4 Exception Conditions

DTOPN$ will return exception condition 1 if an irrecoverable I/O error occurs while attempting to open the database file. Exception condition 2 will be returned if the specified file cannot be found, or if it does not contain the index specified by the INDEX NAME clause of the FD.

DTWRT$ will return exception condition 2 if there is insufficient space in the database file to add the new record. Irrecoverable I/O errors will be passed to the failure routine defined in the ON FAILURE clause if they occur.

4.5.5 Programming Notes

When the record set is made available for data take-on, by a call on DTOPN$, all the indexes associated with it are marked as requiring rebuilding. Even if no records are added by calling DTWRT$ it will be necessary to call IRBLD$ to rebuild the indexes before the database may be used for normal processing.

While a database is open via DTOPN$ the FD may not be used for ordinary READ and WRITE statements, and any attempt to perform ordinary DMAM I/O statements via the FD will result in your program being terminated with a STOP code. Similarly if either DTWRT$ or DTCLS$ is attempted on an FD opened using an ordinary OPEN statement your program will be terminated with a STOP code.

After adding all the required records you must call IRBLD$ to rebuild **all** the indexes associated with the record set before attempting ordinary processing of the records. If you do not then any attempt to access the record set will result in an exception from the OPEN statement as indicated in section 3.3.

No checking of input key values takes place when DTWRT$ is called, so it is possible to write records into the database with conflicting unique key values. If this should occur IRBLD$ will terminate with a stop code during index rebuild, and it will be necessary to change the key definition to make the index non-unique (using DBMAIN), rebuild the index, and then delete offending duplicate entries before the index can be made unique again and rebuilt correctly. This is clearly an undesirable process to follow, so it is very important to ensure that new records being written to the database have distinct unique key values, both from themselves and from those of any other records in the database. Where records are converted from an existing ISAM application this will automatically be the case for the ISAM key

field. In other situations you should either take appropriate precautions to ensure key clashes do not occur, or not use DTWRT$ at all.

4.6 The Key Read routines

A number of special routines are provided to enable you to perform a key read on a DMAM database **without** retrieving the record identified. When the key read routine is called the conventional processing of the DMAM index takes place, but instead of passing back the record the routines exit leaving the record address in the RECORD ADDRESS field within the DMAM FD, so that it may be retrieved by a subsequent READ PHYSICAL operation on the DMAM database.

In addition, for Global format DMAM databases, the key read routines leave the record key in the extension area of the FD, so that it may be examined by the program.

Note that for C-ISAM format and Btrieve format DMAM databases ignore keys will not be honoured.

4.6.1 Invocation

Each key read routine is invoked by a call of the form:

```
CALL DBREK$ USING filename record
CALL DBRFK$ USING filename record [key-length]
CALL DBRLK$ USING filename record [key-length]
CALL DBRNK$ USING filename record [key-length]
CALL DBRPK$ USING filename record [key-length]
```

Where filename identifies an open DMAM FD which will remain open when the operation returns control and record identifies a record area from which the key value is determined (for DBREK$), or from which a partial key value is taken (for DBRFK$, DBRLK$, DBRNK$ and DBRPK$) when the optional key-length parameter is specified. The key-length (if specified) must be a PIC 9(4) COMP variable or integer literal specifying the partial key length. **It must not be specified for a call on DBREK$.**

4.6.2 The Available Routines

There are five key read routines, each one corresponding to a DMAM read operation. They are:

DBREK$ reads the key specified in the record area passed to the routine (as in a READ operation);

DBRFK$ reads the first key in the index, or first key matching the partial key specified in the record area passed to the routine (as in a READ FIRST operation);

DBRLK$ reads the last key in the index, or last key matching the partial key specified in the record area passed to the routine (as in a READ LAST operation);

DBRNK$ reads the next key in the index (as in a READ NEXT operation);

DBRPK$ reads the previous key in the index (as in a READ PRIOR operation);

All the routines, apart from DBREK$, may be passed a partial key which is used in the same way as in the equivalent DMAM read operation.

4.6.3 Exception Conditions

Each key read operation is capable of returning the same exception conditions as the equivalent DMAM read operation, for the same reason, and reference should be made to the appropriate sections of Chapter 3 for details.

Note that the key read routines perform no locking, and hence the lock unavailable exceptions cannot be returned by the routines.

The ON FAILURE processing performed by DMAM for conventional DMAM read operations also applies to the key read routines.

4.6.4 Programming Notes

For Global format DMAM databases, the actual key returned by the key read routine, can be obtained from the extension area within a DMAM FD by a redefinition of the FD as follows:

```
01   FILLER REDEFINES filename
  03 FILLER          PIC X(122)
  03 DBKEYL     PIC 9(2) COMP
  03 FILLER          PIC X(26)
  03 DBKEY      PIC X(99)
```

The DBKEYL field is the actual DMAM key length including the 4 byte addition which is appended to non-unique keys. The first 'DBKEYL' bytes of DBKEY are the key returned by the key read routine, and the remainder of DBKEY is undefined. Note that DBKEY field is returned in the format that it is held in the index block of the DMAM database. This is of particular relevance for translated and descending-order key fields.

There are two common uses of the key read routines. The first of these is when the data contained in the key permits the program to make a decision as to whether the record requires processing or not. Records which do not require processing are thus not read from disk, saving the access time involved. When a record is required for processing it will be retrieved by a READ PHYSICAL operation. Keys used for such processing are often a concatenation of a number of significant record flags, which may only be sensible as a record key when used in conjunction with the key read routines.

The second use of the key read routines is to permit a record to be retrieved with a partial key length which is decided at run-time. In such a case the appropriate DMAM read operation is simulated through a concatenation of a key read with the correct partial key length and a subsequent READ PHYSICAL to obtain the record concerned. Record locking is achieved through the use of lock options on the READ PHYSICAL itself. If a record locked condition arises, then the whole process (key read and READ PHYSICAL) should be repeated if it is retried, as the key values of the record may have been changed by the process which originally had it locked.

4.7 The Read Translation Table routine, DBRTT$

The Read Translation table routine is provided to enable you to retrieve the translation table from a DMAM database file, either to process it for your own purposes or, more likely, to use it in conjunction with the multi-phase sort, MSORT$ (documented in the Global Development File Management Manual), to govern the processing of translation sort keys in a way compatible with their handling in the DMAM database concerned.

4.7.1 Invocation

To retrieve the translation table you code a call of the form:

```
CALL DBRTT$ USING filename area
```

Here filename identifies an open DMAM FD which will remain open when the routine returns control, and area identifies a 260 byte area into which the translation table and its four byte header are to be read.

4.7.2 Exception conditions

Exception condition 1 will be returned if an irrecoverable I/O error occurs while attempting to retrieve the translation table from the database file.

If the FD passed to DBRTT$ is not open, or does not identify a DMAM database file, then your program will be terminated in error.

4.7.3 Successful completion

Assuming that no I/O error occurs, the translation table is taken from the database file, and placed in the 260 byte area.

4.7.4 Programming Notes

The structure of a DMAM translation table is described in Appendix D. Note that it may be read directly into the 260 byte area provided for translation tables inside MSORT$.

5. Data Management Utility Programs

This chapter describes in detail how to use each of the utility programs provided as part of the Global Cobol Data Management system.

Utility programs are provided to create and maintain DMAM database files, as well as to deal with some of the more frequent housekeeping activities such as index reorganisation. There are also some special purpose utilities associated with data take-on and file conversion.

5.1 Database Creation and Maintenance using DBMAIN

The DBMAIN utility is used either to create a new DMAM database file, or to amend the index arrangement of an existing DMAM database file. The database can be held in one of three ways: As a single Global format DMAM database, as a Global schema file describing the one or more C-ISAM databases that constitute a C-ISAM format DMAM database, or as a Global schema file describing the one or more Btrieve databases that constitute a Btrieve format DMAM database.

DBMAIN can also be used to obtain some statistical information regarding the utilisation of space within a DMAM database.

5.1.1 C-ISAM format and Btrieve format DMAM databases

Before using DBMAIN to create a C-ISAM format DMAM database or a Btrieve format DMAM database, you must first decide the record structures to be used. When the record structures have been designed, a set of tables which define the conversion of the Global format records to the C-ISAM or Btrieve format records must be created. This is achieved using the conversion table build utility, RCBUILD, as described in the Global Development File Management Manual. The conversion tables must be held in a single data library on the same unit as the DMAM database schema file. Each record of this data library must contain the conversion table for a single record set in the DMAM file and must have the same record name or set-id as the record set in the DMAM file (see section 5.1.5).

The conversion table from the data library is only used by DBMAIN when creating a new record set. When DBMAIN is used to amend an existing record set the conversion table is read from the schema file. However, **the data library must be present whenever DBMAIN is run.** It is possible to take on conversion tables again after exiting DBMAIN (see 5.1.11).

The RCBUILD utility is fully documented in the Global Development File Management Manual. However, the following points should be observed when building conversion tables for DMAM files:

- A single key segment, as defined by DBMAIN, should be equivalent to a single conversion field as defined in the conversion table;
- For performance reasons, it is STRONGLY recommended to describe all key segment fields first. This convention will also increase the "readability" of the conversion table. DBMAIN will pick up the first field at the required offset to build the C-ISAM or Btrieve key structure;
- For both C-ISAM format DMAM databases and Btrieve format DMAM databases, translated key fields must be converted to two separate

C-ISAM (or Btrieve) fields. The first (key) field must have the special 'TRANS' key word as described in the Global Development File Management Manual. This field will hold the translated key value (see appendix C). This field must be immediately followed by a second conversion field which will hold the original value of the translated key field.

- For C-ISAM format DMAM databases only, descending-order key fields must be converted to two separate C-ISAM fields. The first (key) field must have the special 'DESC' key word as described in the Global Development File Management Manual. This field will hold the descending-key Value (see appendix C). This must be immediately followed by a second conversion field which will hold the original value of the descending-order key field. Note that the requirement to hold two separate fields for descending-order keys is not necessary for Btrieve format DMAM databases (descending-order keys are supported by Btrieve).

5.1.2 Initial Menu

On entering DBMAIN an initial menu will be displayed which offers you the option of either creating or maintaining a Global format DMAM database, a C-ISAM format DMAM database or a Btrieve format DMAM database. If you select to run DBMAIN on a C-ISAM format DMAM database when the C-ISAM Universal Channel Interface (UCI) is not available (i.e. if DBMAIN is not being run on a Global System Manager (Unix) configuration with the appropriate BACNAT software), a 'NOT AVAILABLE' message will appear. If you select to run DBMAIN on a Btrieve format DMAM database when the Btrieve Universal Channel Interface (UCI) is not available (i.e. if DBMAIN is not being run on a Global System Manager (MS-DOS and Windows) or Global System Manager (Novell NetWare) configuration with the appropriate BACNAT software), a 'NOT AVAILABLE' message will appear. For Global System Manager (BOS) configurations, a Global format DMAM database will always be expected.

5.1.3 Initial Dialogue

When you run the appropriate menu entry you are prompted for the file name and unit of the DMAM database file (or schema file) you wish to process. If this database cannot be found you are asked to confirm that you wish to create a new database file and to specify its initial size. You will be asked to specify the name and unit of a template database to be used in creating the new file - if you do not wish to use one simply key <CR> to the template database prompt.

When a template database is used, details such as the record set and index information are copied from the template database into the new database (where it may be amended if necessary). No data records are transferred, so the newly created database will be empty, but with the same record set structure as the template database.

Only a Global format DMAM database file may be used as a template for a Global format DMAM database. Only a C-ISAM format DMAM database file may be used as a template for a C-ISAM format DMAM database. Only a Btrieve format DMAM database file may be used as a template for a Btrieve format DMAM database.

You are asked to specify the **maximum** number of record sets the new database may contain.

For C-ISAM format DMAM databases and Btrieve format DMAM databases you are also asked to specify the directory path for the set of C-ISAM (or Btrieve) files which will make up the DMAM database, and the name of the Global data library containing the conversion records.

Once the basic database has been created, for Global format DMAM databases only, you are shown a screen of statistics about the index allocation and used file space of the database. At this point you may either alter the extent used for allocating index data blocks (IDBs), or key <CR>.

After continuing from the index allocation prompt you are given the option to amend the translation table used by the database.

5.1.4 Specifying a Translation Table

If you choose to amend the translation table, you must specify the file-id and unit of a file from which the new translation table may be obtained. This should either be an existing DMAM database which contains a translation table you wish to copy, or a simple file in the form described in Appendix D. You should refer to Appendix D, which details the layout of a DMAM translation record and suggests methods by which you might set up a table of your own. Note that the translation table will have been taken from the template database, if one was specified.

Having amended the database translation, all record sets which have indexes which use translation will be marked as requiring rebuilding before they may be used.

After amending database translation, or if you elect not to do so, you will go on to examine the record sets defined for the database file.

5.1.5 Amending and Creating Record Sets

Each record set within the database file has a unique two-character identifier, called the set-id. It also has a 30-character long description intended to give a helpful description of the purpose of the record set. The record set list will also give further information about the set, such as the length of its records, the allocation extent associated with it, the number of records present within that record set (and the number of free entries available before a further extent needs to be allocated) and the number of indexes associated with that record set as appropriate.

To amend a record set you key its set-id to the baseline prompt. To create a new record set you key C to the baseline prompt, and then supply the set-id of the new record set (which must be unique over the database). You may also delete a record set from the database and print out the database and record set information. For Global format DMAM databases, a record set may only be deleted from the database providing no records have ever been written to that record set.

5.1.6 Processing a single record set

When you amend or create a record set a further screen is displayed, showing information about the Global index arrangement.

If you are creating a new record set you will be prompted for a record length. You will also be asked to supply an initial allocation extent for the record set for a Global format DMAM database or for the name of the C-ISAM (or Btrieve) file containing the record set for a C-ISAM

(or Btrieve) database. You will then be prompted to define indexes associated with the record set.

5.1.7 Allocation Extents

Allocation extents are only of interest for Global format DMAM databases. Associated with each record set, and also with the allocation of IDBs, is a value termed the **allocation extent**. This value is used whenever it becomes necessary to add new records or IDBs to the database. Rather than make space for a single record or IDB, the file is extended by an allocation extent, which will comprise a number of records or IDBs.

The principle reason for allocating data space in extents is to improve the general performance of record addition, as well as to simplify the handling of index rebuilding and recovery. There is a small sacrifice in file space utilization, because records and IDBs do not span extents. As extents may only be allocated in units of 1K (1024 bytes) there will be unused space at the end of each extent. Provided that extents are not too small (more than 30 records or IDBs) this is not an important factor (a few per cent of file size).

5.1.8 Index Definition

Each index is identified by its unique index name. To amend an index associated with the selected record set you simply key its name (or number). To create a new index you key C to the baseline prompt and then provide the name of the new index. To avoid problems with duplication of index names we recommend that you use the two character set-id as the first two characters of the index name.

Having selected an index to maintain, the details of the **key segments** will be displayed on the screen. You may specify whether the index is unique or not (i.e. whether duplicate key values are permitted). A record may have more than one unique index (or even none at all), but all unique indexes for the record must be grouped together at the start of the indexes.

For a non-unique index you may specify that certain key values are to be omitted from read processing. For Global format DMAM databases it will mean that the record will be omitted from the index leaving the index smaller. For C-ISAM format DMAM databases and Btrieve format DMAM databases the record will still be present in the C-ISAM (or Btrieve) file index but will be ignored by a DMAM READ operation. You do this by specifying an **ignore code** for the index. The first character of the code indicates which values are to be ignored (N for none, S for spaces, L for low-values and B for both spaces and low-values), and the second character which element of the key is to be examined (A for all the key, F for the first segment only and L for the last segment only). So a type of LL, for example, would cause keys whose last segment was low-values to be omitted from the index.

Important note: Under V6.1 DBMAIN, only the whole key may be selected, and two separate prompts ask whether spaces or low-values are to be ignored. Database indexes established using V6.1 DBMAIN will thus appear as NA, SA, LA or BA under V8.1, depending on which values were selected to be ignored. **If you set up a database using V8.1 DBMAIN to use any ignore code other than these, you must ensure that indexes on the database are only rebuilt using V8.1 or later IRBLD$ or DMAM utility programs. Using pre-V8.1 index rebuild facilities on such a database will lead to data corruption.**

Finally, for Global format DMAM databases, you are asked to specify the **index type**. You may choose one of the following options:

AS Specifies a key which is **almost always** added in ascending sequence (i.e. new records are added **after** the last record in the index). There is no need for spare space to be left in the index blocks, and this is taken into account when adding records or reorganising indexes.

DS As above, except the key is added in descending sequence (i.e. records are added in front of the first record in the index).

AN Specifies a key where records are normally added in ascending sequence (i.e. after the last record), but there will be occasional out-of-sequence additions. Index blocks will be split leaving 20% free space and reorganised to leave 10% free space in them to avoid performance degradation on non-sequential additions.

DN As above, except that records are normally added in descending sequence (i.e. before the first record).

RI Specifies a key where keys are added randomly, but additions are infrequent. Index blocks will be split in half, and reorganisation will leave 10% free space in the indexes.

RF This is the default, and specifies a key where additions will occur at random and with reasonable frequency. Index blocks will be split in half and reorganised to leave 20% free space in them.

Care must be taken when specifying any index type other than RF, as an unwise choice of index type (specifying AS when RI would have been more appropriate, for example) can very seriously degrade the performance of the database. On the other hand, a sensible choice of index type can avoid considerable overheads in index space and depth. Index types should only be specified by someone who has a detailed knowledge of the internal functions of the software which will be using the database.

Index types of AS, DS and DN should only ever be used for unique keys, as non-unique keys have the record address appended to the key, and will typically not be in true key sequence.

The key length and number of segments are maintained automatically as you update the key segment information.

5.1.9 Specifying key segment information

Each key you define may be made up from up to eight segments of data from the record. Typically each segment will correspond to a data item in the record definition. There are six types of segments you may define:

XA defines a character ascending segment. Keys will be collated in strict ASCII sequence.

XD defines a character descending segment. Keys will be collated in reverse ASCII sequence.

CA defines a computational ascending segment. Keys will be collated in binary numeric sequence, starting with the largest negative number and passing through zero to end with the highest positive number.

CD defines a computational descending segment. Keys will be collated in reverse binary numeric sequence.

TA defines a translation ascending segment. Keys will be translated via the defined translation table and collated in ascending sequence of the result.

TD defines a translation descending segment. Keys will be translated and collated in reverse sequence.

When specifying segments remember that PIC DATE fields, containing an internal format date, should be treated as PIC 9(6) COMP fields.

For each segment you must supply a start offset within the record (which may not be less than 4) and a length. The total length of the key may not exceed 99 bytes (and non-unique keys have a four byte overhead for Global format DMAM databases, so the total length of the segments may not exceed 95 bytes), and no part of the key data may lie outside the first 255 bytes of the record (or the record itself if this is smaller).

For translation keys, which may permit compaction, the length merely serves to define the length of the input data item. You must also define a translation key length, which is the maximum number of characters which may result from the translation and compaction. This latter field is the actual length added to the key, and is subject to the restrictions mentioned above.

You may define up to eight segments for each key. Note that each translation segment requires two segments' worth of data to be held. You are also limited to a maximum of 64 segments' worth of data for all the indexes associated with a single record type.

5.1.10 Guide lines for setting up Indexes

To avoid wasted space you should ensure that keys are no longer than necessary for the function they are to provide, and that non-essential keys are avoided. Nevertheless it is valuable to define a key which will enable records to be read in an appropriate sequence where this avoids a frequent sort.

To simulate the current ISAM you should define a single, unique, character ascending (XA) key starting at offset 4 with a length equal to the ISAM key length.

Where several adjacent character data items comprise part (or all) of a key, it is sufficient to define a single character key to cover all of them. Similarly if a computational key is followed by a series of adjacent character data items a single computational key will serve for them all for a Global format DMAM database. For a C-ISAM format DMAM database or a Btrieve format DMAM database, computational fields should be defined as separate key segments. However, unlike sort keys,

there is little performance benefit in amalgamating segments in this way, and so it is probably better avoided (for reasons of clarity) unless you are short of segments to specify a key with.

Translation keys are commonly used to create **alpha index** keys, where only letters in the range A to Z are of interest and character case is unimportant. These are most efficiently held in a RAD-50 format (three characters in two bytes). Where a long character field is being compressed into an alpha index, it is not necessary to have the key long enough to cater for the longest possible alpha index key, merely that it is long enough for typical keys to be distinguishable. Thus, for example, a key based on a 30-character customer name would only need 15 significant characters typically, and these could be contained in 10 bytes of key when compressed (a two-thirds saving on key size).

Translation keys will also be of interest to those non-UK sites where the local language uses non-standard ASCII characters which must be collated in the correct sequence.

5.1.11 Finishing Database Maintenance

When you have completed updating the index layout of a record set key <ESC> to return to the record set list. From here you may amend or create further record set information, or key <ESC> to complete database processing.

If you have been updating a C-ISAM format DMAM database or a Btrieve format DMAM database you will be asked if you want to take on the conversion tables again. This allows the current conversion tables held in the schema file to be replaced by new conversion tables from the associated data library. If the conversion of key fields have been modified, however, it may be preferable to rebuild the database.

When you finish database processing, the updated information you have specified is written back to the database header. Any new or altered indexes for a record set which has existing records are marked as requiring rebuilding, and a rebuild overlay is invoked which will rebuild the required indexes one at a time. If, for some reason, the index rebuild should fail to complete then you will need to run the index reorganisation utility, DBREORG, to complete the index rebuild before attempting to use the database with any application.

5.2 Index Reorganisation using DBREORG

The DBREORG utility is used to reorganise one or more indexes within a DMAM database file. You would normally use it if a data take-on process had failed before reorganising the indexes, or in a situation where investigation had shown that an index had become sparsely filled, due to record deletions.

5.2.1 Initial Dialogue

When you run the DBREORG utility it asks you for the file name and unit of the DMAM database file you wish to process. Having selected a database you are shown a list of the indexes existing within that database, along with the record set on which they are kept and some other statistical information. Any indexes which currently require reorganisation (due to an incomplete data take-on for example) are shown highlighted.

You may select further indexes to be reorganised, and may further choose to reorganise all indexes on the record set thus identified. All reorganisations you specify are also shown highlighted in the list.

When you have finished your specification of indexes to be reorganised reply <ESC> to the baseline prompt, and DBREORG enters the rebuild phase.

5.2.2 The Rebuild Phase

When you have finished specifying a list of indexes to be rebuilt DBREORG enters the index rebuild phase. Each index in turn is rebuilt, and a message is displayed on the screen showing the start and end of each rebuild so that you may keep track of the progress.

When all the specified indexes have been rebuilt control is returned to the Global System Manager READY prompt or the main menu.

5.3 Data Take-on using DBTAKE

The DBTAKE program is used to transfer all the records of an existing ISAM file into a DMAM database. DBTAKE may also be used to transfer all the records of a record set in one DMAM database to a record set in another DMAM database. Thus, repeated use of DBTAKE (i.e. once per record set) will allow the entire contents of a DMAM database to be copied into another DMAM database. This technique may be used to transfer data, in any direction, between Global format DMAM databases, C-ISAM format DMAM databases and Btrieve format DMAM databases.

5.3.1 Specifying the files to be processed

You are first asked by DBTAKE whether you want to transfer records from an ISAM file to a DMAM database or from one DMAM database into another DMAM database. Reply with **I** or **D** as appropriate.

You are then prompted by DBTAKE for the file-id and unit of the ISAM or DMAM file which is to be converted. For a DMAM file you will also be asked for an index by which records of the required record set can be read. The records in this file or record set must not be longer than 4096 bytes. DBTAKE opens the input file, and then prompts you for the name, unit and index name of the DMAM database to which the input records are to be added file is to be converted. This database must have been previously defined using DBMAIN. It should have at least a primary key specified which matches the current ISAM key or an index for the DMAM record set, and it may have further keys defined to match other fields on the record which you wish to process as keys. It is not normally safe to define any other unique keys for the record unless you are **absolutely sure** that there are other unique data fields, or you are defining keys which are a super-set of the main unique key. Again the record length of the DMAM record set must not be greater than 4096 bytes.

5.3.2 Consistency Checking

DBTAKE will perform a number of consistency checks on the file or record set to be converted and the record set identified by your choice of DMAM file-id and index name. It will warn you if the DMAM record set has a record length smaller than that of the input record set, or if there are any existing records defined in the record set. Both of these situations are probably errors, but if you may choose to continue if you have deliberately organised things in this way.

If the records in the input file are larger than those in the DMAM record set then they will be truncated as they are copied. If they are smaller than those in the record set they will be extended with low-values up to the end of the new records.

If there are already records in the record set then unless you have not defined a unique primary key, or have taken care to ensure that there are no possible clashes in key value between the existing records and the new ones to be added, the data take-on will probably fail with an error when the indexes are built on the record set after take-on is complete.

5.3.3 Specifying the Sort Work File

When DBTAKE is satisfied that it is safe to proceed it will ask you for the name and unit of a sort work file, which will be used by the index rebuild process after the new records have been taken on.

5.3.4 Copying Records

Having obtained all the information it requires DBTAKE now proceeds to copy the records from the input file or record set to the DMAM database, using the data take-on routines described in section 4.5. As it copies the records it displays a count so that you may see how the process is proceeding.

When records are read from an ISAM file OPTION IGNORE is used, so any logically deleted records will be removed from the records present in the database.

5.3.5 Building Indexes

Once all the records have been copied from the input file, DBTAKE will invoke the index rebuild routine to rebuild all indexes on the record set.

When the indexes have been built DBTAKE returns control to the Global System Manager READY prompt or the main menu.

5.4 Database Rebuild and Reorganisation using DBRBLD

The DBRBLD program is used to perform a complete data rebuild of your database, removing deleted record and index space, and optionally reorganising the physical layout of the records.

Important note: DBRBLD will only operate on Global format DMAM databases.

5.4.1 Specifying the files to be processed

DBRBLD begins by asking you to specify the file name and unit of the database to be reorganised. You are shown certain information about the database (such as its size and the number of record sets it contains), and are then asked to specify the file name, unit and size of the output database file to be created. The output file must be large enough to contain all the data which will be passed to it (see later sections for an explanation of what affects the amount of data which will be passed to the output file).

Once input and output files have been specified you are asked to confirm that the information is correct before proceeding to record set selection.

5.4.2 Selecting record sets to be processed

Once you have chosen the files to be used during the rebuild, you are shown a screen with the first few record sets contained in the database on it. You may key <CR> to page forward and look at the next few record sets, cycling back to the beginning once you reach the end.

Initially all record sets in the file are selected to be copied to the output file, ordered in their current physical sequence (which is in principle random). You may choose the following options:

A to abandon the update, if you have made some serious mistake in record set selection or omission, and wish to go no further in case irreparable damage is done to the database.

O to omit a record set from the selection. Record sets which are omitted will not be copied to the output file when reorganisation is performed (so the entire record set will in effect be deleted from the output file). A record set which has been omitted is shown with a 'x' character against it in the list. If you omit a record set by mistake, and wish to re-introduce it, you should key 'O', and then key 'N' to the subsequent confirm prompt.

S to sequence a record set in a particular order when it is copied to the output file. You are asked to identify which record set you wish to sequence, and on which index you wish to sequence it (a reply of 0 causes the record set to be sequenced in its current physical order, the default). Sequencing a record set on an index which is frequently used to process records sequentially (using READ NEXT or READ PRIOR) may yield performance benefits, although the sequence will not be maintained if further records are added, or existing records are amended so the value of this on volatile data is probably slight.

If you select a currently omitted record set to sequence, then it will be automatically re-introduced to be copied to the output file. A record set selected for sequencing is shown with a '*' character against it in the list (unless it is still sequenced in physical order).

If an update to the database has failed leaving the index you are sequencing on in an inconsistent state, the following message will appear:

CANNOT SEQUENCE CORRUPT RECORD SET record-set

T to perform type checking on a record set. This option is only available in V8.1 or later. You are asked to identify which record set you want to type check and what the record type should be. A reply of <CTRL A> to the type will clear any checking on the record set. A record set which is to be type checked is shown with a '=' character against it in the list. DBRBLD will omit all record in the sets to be checked which do not have the correct type value in the type field.

If an update of the record set has failed leaving the record indexes in a corrupt state the following message will appear:

CANNOT TYPE CHECK CORRUPT RECORD SET record-set

Once you have sequenced and omitted the required record sets you should key <ESC> to proceed to select the reorganisation type.

5.4.3 Selecting the Reorganisation Option

If you have keyed <ESC> to the previous prompt by mistake, you can key <ESC> to the reorganisation option prompt to return to the record selection stage. Alternatively you should select one of the reorganisation options presented:

E to create an empty database, containing those record sets which you have chosen not to omit but none of the data records. This option is similar to using a template database with DBMAIN, except that record sets may be omitted from the new database created. This option is not offered if you have chosen any record sets for sequencing (as no record data is copied, record set sequencing is incompatible with creating an empty database).

R to rebuild the database, copying all record sets except those which have been omitted to the output file, sequencing appropriately any which have been chosen, and then rebuilding all indexes in the output file. The output file contains the fully reorganised database, which you might wish to copy back over the original input file (using $F for example).

D to rebuild the database, copying appropriate record sets in the correct sequence to the output file. The output file is then copied back over the original input file, deleting it, before indexes are rebuilt. This allows you to reorganise a database with a smaller work space then the Rebuild option allows (the output work file only needs to be large enough to contain all the data records without the indexes, which typically means it can be half the size), although admittedly with the small danger that a serious failure in the processing might occur compromising your data integrity. If a machine failure occurs it should be possible to complete the rebuild process by running DBREORG on the incomplete file, and rebuilding all the indexes.

Once you have selected a reorganisation option, the reorganise and rebuild you selected will take place. Creation of an empty database should only take a few seconds, but the other reorganisation options may take some time, especially if there are a large number of records involved.

Both rebuild options copy records from successive record sets of the input file to the output file (reading via an index if they are to be sequenced - this means that records cannot be sequenced on a corrupt or incomplete index). Once all records have been copied, each index in turn is rebuilt.

5.5 Inspecting and Amending a database using DBDUMP

The DBDUMP program is a programmer's tool, intended to allow you to inspect and amend records of a DMAM database file in a partially structured way. It enables you to examine records created by a program under test (or possibly in a live system), in order to check their validity. It uses DMAM to access records, and displays their contents on the screen in both hexadecimal and ASCII equivalents where appropriate, without regard to the field structure of the record.

5.5.1 Selecting the file and index to inspect

When you run DBDUMP, it asks you for the name and unit of the database to inspect, and the initial index you wish to inspect via. You may subsequently change index, even moving to a new record set if desired, but in all cases you must know the name of the index to be processed (DBDUMP does not provide this information to you, but it may easily be obtained from a print of the index structure using DBMAIN).

5.5.2 Inspecting and amending records

Initially DBDUMP reads and displays the very first record in the index selected. From the baseline prompt you may then select the following options:

A to amend the record. You are allowed to move through the displayed record contents over-typing existing data in either hexadecimal or ASCII (use the TAB key to change from hex to ASCII and vice versa). When you have finished amending data key <ESC>, and you will be asked if you wish to REWRITE the record with the altered data.

C to change index. Key the name of the new index you wish to use to access records from the file. If the index is on another record set you will be placed on the first record in that index. If it is on the same record set, then you will be placed on the same record within the new index, so that you may use Next and Prior to examine records using the new index.

D to delete the current record. You will be asked to confirm that you wish to do this.

F to move to the first record in the index.

L to move to the last record in the index.

N to move to the next record in the index.

P to move to the previous record in the index.

R to read a specific record from the record set. The various segments of the record key are highlighted on the display, and you are allowed to amend the record to establish the key value you require. When you key <ESC> the record is read (if no matching record is found a warning is produced, and when you key <CR> the record with the next higher key is read).

Keying <CR> has the same effect as keying 'N' if the last option you chose was C, F, N or R, and has the same effect as keying 'P' if the last option you chose was L or P.

5.5.3 Notes on using DBDUMP

When a record is displayed the record address is shown on the screen, along with the first 256 bytes of the record. If the record is longer than this you may examine the remainder of the record by keying A to amend the record, and moving the cursor off the bottom of the display (or the top to go back). You may also use the F4 key to page forward and F5 to page back.

During record amendment the current offset address within the record is shown (in hexadecimal), to help you in identifying fields within the record.

When you have finished inspecting and amending records, key <ESC> to the baseline prompt to exit back to the monitor.

5.6 Producing a Structured Dump of a Database using DMSDUMP

The DMSDUMP program produces a structured dump of a Global format DMAM database or the Global schema file for C-ISAM format and Btrieve format DMAM databases. This dump is intended as a programmer's tool and shows information regarding the present state of the database as an aid to program debugging. The report includes:

- Database statistics;
- Translation table information;
- Record set and index information;
- UCI conversion table information (C-ISAM format and Btrieve format DMAM database only);
- Information on the database extents (Global format DMAM databases only);
- Deleted record chains (Global format DMAM databases only)

DMSDUMP allows you to dump a single record set or all the record sets in the DMAM database.

For example, the following dialogue runs DMSDUMP to dump all the record sets for the $$MAIL database on unit $ML. The report will be created on the $PR unit:

```
GSM READY:DMSDUMP
DUMP DMAM FILE:$$MAIL  UNIT:$ML
PRINT UNIT ($PR):<CR>
Key record set number, <CR> for all:<CR>
```

Appendix A - DMAM STOP and EXIT Codes

This appendix describes the various stop and exit codes which can be produced by the programs and subroutines documented in this manual.

As is normally the case, stop codes identify fatal error conditions which prevent the program continuing, and exit codes are terminations caused by the absence of an ON EXCEPTION statement immediately following the statement which generates the exception condition.

The following notes are referred to, as appropriate, in the following pages:

Note 1 When a stop or exit code is generated by a file processing statement, the type of file processing statement along with the affected file-id can be found from the diagnostic report.

Note 2 When a stop or exit code is the result of an I/O error during file processing the dialogue preceding the program check information will fully explain the events leading up to the termination.

Note 3 Some of the codes described here are included for completeness only. They should never occur. If they do, suspect program corruption or try reloading Global System Manager.

Note 4 This stop code will cause a failure routine, set up using the ON FAILURE clause of a DMAM file definition, to be entered if one has been defined. The value of the indicated stop code will be in the accumulator when the failure routine is entered, so that it may perform discriminatory processing on the stop code. If no failure routine has been specified the stop code will appear on the screen as part of a TERMINATED - STOP message.

A.1 Stop Codes

STOP 8801 An irrecoverable I/O error has arisen while reading or writing data within a DMAM database file during file processing. The database will need to be restored from back-up. See notes 1, 2 and 4.

STOP 8802 An unexpected error condition has arisen whilst processing a DMAM file. See notes 1, 3 and 4.

STOP 8804 An attempt to delete a key from an index has failed because the old key value cannot be found. The index is corrupt. Either rebuild the index or restore the database from back-up. See notes 1 and 4.

STOP 8805 A READ PHYSICAL has been attempted on an illegal record address (most likely the address of 0 or -1 left by a start of file or end of file condition).

STOP 8806 An attempt to recover the last operation on a DMAM file has failed because of a key clash with a non-unique key. See notes 1, 3 and 4.

STOP 8807 An illegal operation (e.g. OPEN NEW) was attempted using the Data Management access method. See note 1.

STOP 8808 An attempt has been made to open a C-ISAM format DMAM database or a Btrieve format DMAM database on a version of Global System Manager where a Universal Channel Interface (UCI) is not available.

STOP 8809 An attempt has been made to open a DMAM database using a read-only version of DMAM, but there was an outstanding write operation noted in the header (due to some earlier process failing). The database must be opened with a full version of DMAM so that the pending operation may be completed, before it may be processed with a read-only version. See note 4.

STOP 8810 An attempt to recover the last operation on a DMAM database has failed because of an illegal key alteration. See notes 1, 3 and 4.

STOP 8811 An attempt has been made to open an index on a DMAM file, but the space available in the FD to hold the key was too small. See notes 1, 3 and 4.

STOP 8812 The key of the last record read from a DMAM database does not match the key contained in the record on the file when a subsequent REWRITE takes place. This is most likely due to inadequate security surrounding the updating of records on the database. See note 1.

STOP 8813 An attempt has been made to REWRITE or DELETE a record on a DMAM database without a successful preceding READ operation. See note 1.

STOP 8814 An attempt has been made to DELETE a record from a DMAM database without first obtaining a DELETE-LOCK on the record. See note 1.

STOP 8820 An attempt has been made to process a DMAM database using a conventional file processing statement (READ, WRITE etc.) after it was opened by a call of DTOPN$ for data take-on.

STOP 8821 An index block in a DMAM database has become corrupt. The database should be restored from back-up. See notes 1 and 4.

STOP 8822 An attempt to add a key to an index has resulted in an index split which would cause the index to contain more than 20 levels. Rebuilding the index might cure this problem. See notes 1 and 4.

STOP 8823 A DMAM database has become full when attempting to allocate a new index block. Some database indexes may be inconsistent as the record has been written but not all the relevant indexes have been updated. The database must be increased in size and opened again so that data recovery can occur before any further processing takes place. See notes 1 and 4.

Because DMAM attempts to update the index after writing the data record, on opening the file again recovery will take

place and complete the WRITE operation (cf. STOP 8824, see below).

STOP 8824 A DMAM database has become full when attempting to allocate a new record. The database must be increased in size before any further processing takes place. The last record has not been written. See notes 1 and 4.

Because DMAM attempts to write the data record before updating the index there is no need for recovery to take place when re-opening the file (cf. STOP 8823, see above).

STOP 8825 An invalid deleted chain pointer has been detected during a record or IDB addition on a DMAM database file. The database is corrupt and must either be rebuilt or restored from back-up. See note 4.

STOP 8826 An index split during index rebuilding using IRBLD$ has cascaded past the root IDB. See note 3.

STOP 8827 The deleted chain index in the DMAM database has been corrupted. The database is corrupt and must either be rebuilt or restored from back-up. See note 3.

STOP 8830 A duplicate key has been detected when performing an index rebuild on a unique index. Either duplicate records have been added using data take-on procedures, or the database has become corrupt. Before any further processing may be performed on the database it will be necessary to re-define the offending index as non-unique and to repeat the index rebuild process, or to restore from back-up. See note 4.

STOP 8831 The record information was not available in the DMAM database when attempting to recover the last DMAM operation. The database should be restored from back-up. See note 4.

STOP 8833 The DMAM Access Method is unable to produce a unique lock for this record. The C-ISAM or Btrieve file has become too large. Try deleting unwanted records and re-organising the file.

STOP 8863 The IRBLD$ routine has been called on a Global schema file but the Memory Page Open version of DMAM (i.e. AX$Z) has not been linked into the program.

STOP 21101 The DBSTA$ routine has been called but the DMAM FD passed to it was not open.

STOP 21102 The DBSTA$ or DBCLC$ routine has been called on a Global schema file but the Memory Page Open version of DMAM (i.e. AX$Z) has not been linked into the program.

STOP 21201 The DTWRT$ or DTCLS$ routine has been called, but the FD passed was previously opened using a conventional OPEN statement, not a call on DTOPN$.

STOP 21202 The DTOPN$, DTWRT$ or DTCLS$ routine has been called on a Global schema file but the Memory Page Open version of DMAM (i.e. AX$Z) has not been linked into the program.

STOP 21301 An invalid key length (negative or greater than 63) has been specified to the DBRFK$, DBRLK$, DBRNK$ or DBRPK$ routine.

Alternatively, a third parameter has been specified for the DBREK$ routine.

STOP 21302 The DBRFK$, DBRLK$, DBRNK$, DBRPK$ or DBREK$ routine has been called on a Global schema file but the Memory Page Open version of DMAM (i.e. AX$Z) has not been linked into the program.

STOP 21701 The DMAM FD passed to the DBRTT$ routine was not open.

STOP 21702 The FD passed to the DBRTT$ routine was not a DMAM FD.

A.2 Exit Codes

EXIT 8801 An internal error has arisen in the Data Management access method during file processing. See notes 1 and 3.

EXIT 8802 A file condition has arisen while processing a DMAM database file but the user program has failed to trap the resulting exception. See note 1.

EXIT 8803 A file condition has arisen while processing a DMAM database file but the user program has failed to trap the resulting exception. This will be a **key requires rebuild** exception from an OPEN operation, or a **lock unavailable** exception from a READ. See note 1.

EXIT 8804 A file condition has arisen while processing a DMAM database file but the user program has failed to trap the resulting exception. If it is a REWRITE operation, an attempt has been made to REWRITE a record on a DMAM database having **altered a unique key** which is not permitted. See note 1.

EXIT 8899 An unexpected failure has occurred in a DMAM memory page. See notes 1 and 3.

EXIT 21001 The DBSET$ routine has determined that it cannot load the key build routine, either because it is running on a V5.0 system, or because it cannot find a key build routine on SYSRES (routines are named "%.a0V63 ", where a is the architecture code for the computer), and the user program has failed to trap the resulting exception.

EXIT 21101 Overflow has occurred during the calculations on database size performed by DBSTA$ or DBCLC$, and the user program has failed to trap the resulting exception.

Appendix B – Included Routines

Table B below shows the program names of the particular subroutines included from the system libraries when various language constructs described in this manual are coded. The SIZE column indicates the approximate size of each routine in bytes, rounded up to the nearest 0.1K (K = 1024 bytes).

If you require a more accurate estimate you should compile and link a program containing a GLOBAL statement of each of the required routines and file organisations. The link map will then give the total size of the included routines.

Global Cobol statement	Program name of the subroutine included	Size (Kb)
CALL IRBLD$	OC$A, AM$A, BZ$A, CA$A, EC$A, EI$A, ER$A, IX$A	16.7
CALL DBREK$ CALL DBRFK$ CALL DBRLK$ CALL DBRNK$ CALL DBRPK$	OG$A, AM$A, CA$A, EC$A, ER$A	9.7
CALL DBRTT$	OH$A	0.1
CALL DBSET$	OD$A, IE$A, IF$A	3.2
CALL DBSTA$ CALL DBCLC$	OE$A, AM$A, CA$A, EC$A, ER$A	10.6
CALL DTOPN$ CALL DTWRT$ CALL DTCLS$	OF$A, AM$A, CA$A, EC$A, ER$A	9.8
CALL DBKES$	OP$A, AM$A, CA$A, EC$A, ER$A	10.2
FD .. ORG DMAM (normal)	AM$A, CA$A, EC$A, ER$A	9.7
FD .. ORG DMAM (memory page Global)	AM$Z, CA$Z, EC$A	3.5
FD .. ORG DMAM (memory page open)	AX$Z, CA$Z, EC$A	5.6

Table B – Included Routines

Appendix C - DMAM Database Structure

C.1 Global Format DMAM Databases

A Global format DMAM database is divided into two parts, a header area and a data area. The data area is further sub-divided into **extents** each of which contains one or more data records or index blocks. The area of the database beyond the last extent is unused, and extents will be allocated from it as necessary when records are added to the file.

C.1.1 The Header Area

The header area is always at the start of the database file. It consists of the following sections:

- A 256 byte recovery area, for holding the start of an updated record for recovery during REWRITE and DELETE processing;
- 768 bytes of recovery and status information used during database update and recovery;
- 1024 bytes of static database information, including a list of set-ids, which is only updated when using DBMAIN to update the data layout;
- 512 bytes containing the translation table and various other translation information;
- 1536 bytes containing the list of index names, along with associated index types and record set numbers;
- For each potential record set which the database may contain, a 372 byte area defining the index construction and data usage which also contains the 30 character set description.

Space is left for a number of record set data blocks equal to the maximum number of record sets that the database may contain (as specified using DBMAIN).

C.1.2 The Data Area

The data area starts 8 bytes before a 512 byte boundary in the file, as soon as possible after the last record set data block. File space between the last record set data block and the start of the data area is unused.

The data area consists of a series of extents, each of which contains either records from a single record set, or index data blocks (IDBs). The size of each extent must be a multiple of 1024 bytes, but extents may not all be of the same size (indeed the extent size is specified independently for each record set, and for the IDBs, and each extent may be altered during the lifetime of the database).

C.1.3 The Extent Areas

The first 8 bytes of each extent contain information detailing the nature of the included records (whether they are IDBs, or to which record set they belong). They also contain the length of the extent, so that it is possible to read through the file sequentially by processing the extent information. The last 3 bytes of the extent

header are a recognisable value which is uncommon in standard data, and is used by file recovery in the event of data corruption or an irrecoverable I/O error.

As the first 8 bytes of each extent are used by DMAM, and the extents are a multiple of 512 bytes long, the part of the extent used for holding data records, or IDBs, will always begin on a 512 byte boundary in the file. Since an IDB is 512 bytes long this ensures that all IDBs begin on a 512 byte boundary in the file.

C.1.4 Data Records

The first two bytes of each record are a record type. The next two bytes of the record contain an incremental backup sequence, and are reserved for use by the database backup utility. The remainder of the record contains user data.

If the first byte of the record type is an asterisk (*, #2A) then the record has been deleted. All deleted records from the same record set are chained together so that they may be reused. In the recovery area at the start of the file there is a pointer to the last record deleted. The four bytes after the backup sequence field of the deleted record contain a pointer to the next-to-last record deleted, etc. The last record in the deleted chain (or the pointer in the recovery area if there are no deleted records) has a negative pointer field. This field indicates the next position in the last extent allocated for that record set which will be used when a new record is added.

C.1.5 Index Data Blocks (IDBs)

Each IDB is 512 bytes long. IDBs are allocated in separate extents to data records, but IDBs for different indexes may, and indeed normally will, occupy the same extent.

The indexes in a DMAM database are organised in a B*-tree structure, and are dynamically updated as records are added to and deleted from the database.

Each index is arranged conventionally in a number of levels. The location of the highest level (root) index block for an index is indicated in the 372 byte record set control block. All index levels apart from the lowest (leaf) level contain keys and file pointers indicating the location of the index blocks for the next lower level of the index. The leaf index blocks contain keys and pointers to the appropriate records. Leaf index blocks are chained forwards and backwards to improve performance on sequential read operations (READ NEXT and READ PRIOR).

Deleted IDBs are chained together in the same way as data records.

C.1.6 Structure of an IDB

The data within an IDB is laid out as follows:

- The first byte of the IDB is a PIC 9(2) COMP field which contains the level of this IDB. The root will be the highest level, and the leaf IDBs have a level of zero. The level of the root may not be greater than 20 due to memory limitations imposed by DMAM. If the IDB is deleted then this field is set to 42 (#2A, *);

- The next two bytes of the IDB are used to contain the backup sequence information;

- The next byte of the IDB is a PIC 9(2) COMP field containing the number of used keys in the body of the IDB. This may only be zero for a non-leaf IDB (and will only occur in the root of an empty index, or when a large proportion of the keys in a part of the index have been deleted);

- Following the initial four bytes is a PIC 9(9) COMP field which is only used for deleted IDBs and leaf IDBs. In a deleted IDB it is the pointer to the next IDB in the deleted chain, as for the deleted record chains. In a leaf IDB it is the pointer to the next IDB in the leaf chain, or zero if this is the last IDB in the index.

- The next four bytes are a PIC 9(9) COMP field, used in a leaf IDB as a pointer to the previous IDB in the leaf chain, or zero if this is the first IDB in the index. In a non-leaf IDB this is in fact the first index pointer to the next lower level of IDBs.

- In a leaf IDB the previous IDB pointer is followed by a series of pairs of fields making up the body of the IDB. The first field of each pair is a key (of the appropriate length) and the second is a PIC 9(9) COMP pointer to the record which has that key value.

- In a non-leaf IDB key values and file pointers are laid out as above, but the key values are merely separators used to indicate which is the correct IDB from the next lower level of the index. Keys up to and including the separator value are located in the IDB indicated by the preceding file pointer. Keys greater than the last separator value are located in the IDB indicated by the file pointer succeeding the last separator in the IDB.

The structure of a DMAM index is illustrated by figure C.1 overleaf. This shows a stylised index arrangement, with a maximum of three keys per IDB. Note that the separator values in the non-leaf IDBs do not have to correspond to any actual key held at the leaf level.

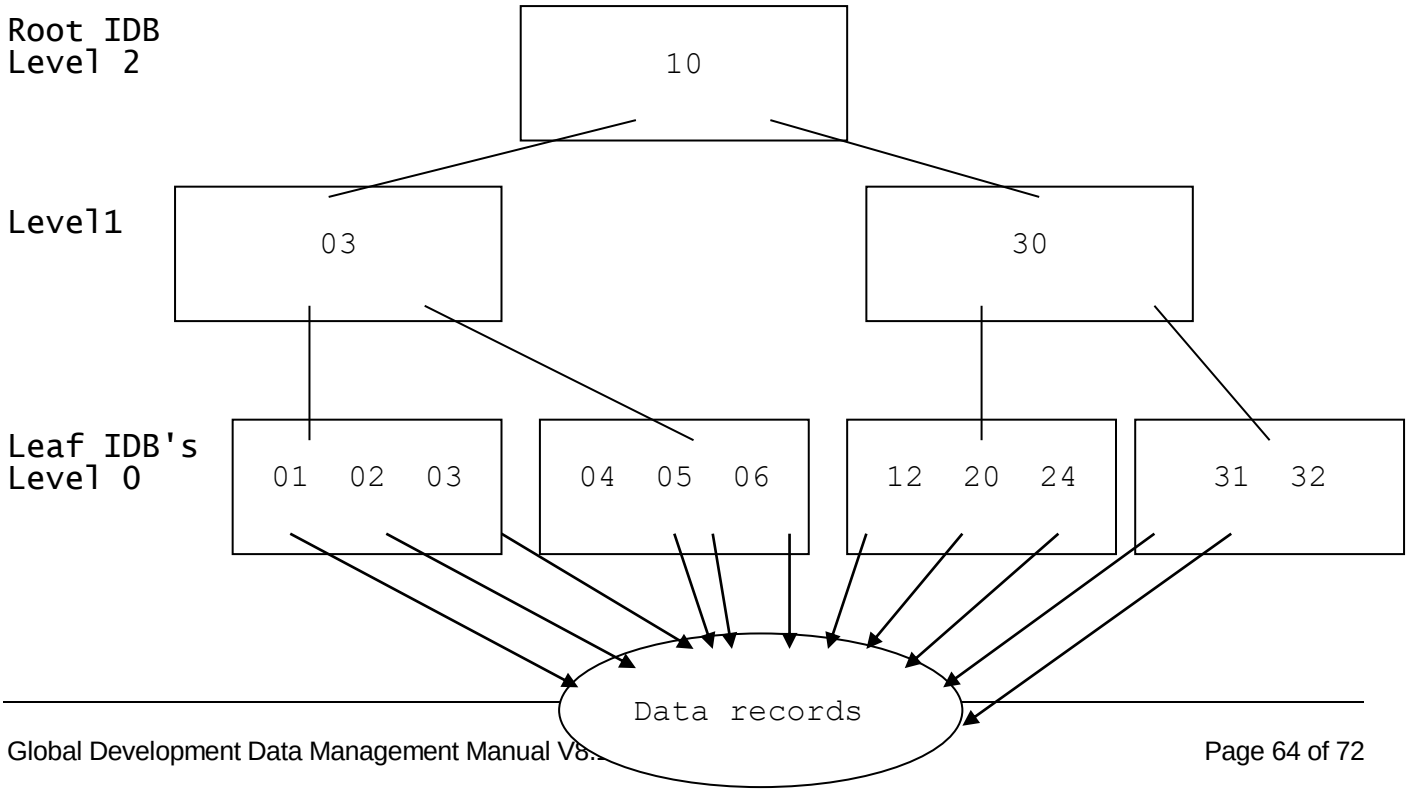


Figure C.1 - Index Layout

When a key is added to an index it is inserted into the correct place in a leaf IDB. If there is insufficient room for this, then the leaf IDB is split (the precise details of the split depend on the index type), the new key is added to the correct leaf IDB, and a new separator is placed in the IDB of the next higher level in the index. This may result in a further split of this index block, which may potentially cascade up to cause a split of the root IDB, and the consequent deepening of the index tree.

C.1.7 Key Format

All keys held in an index are processed by DMAM so that they may be held in simple ASCII sequence. This means that various manipulations are carried out on the data used as part of computational and descending sequence key segments, as well as those done as part of a translation key.

Computational key segments have the most significant bit of the field inverted, so that collation runs from negative to zero to positive.

Translation key segments will be substituted, byte by byte, as indicated by the translation table in the database. They will also be compacted as indicated by the table, either by bit shifting or RAD-50 compression.

Descending key segments have each of the bits in the field flipped (after any computational or translation replacement), to reverse the collating sequence.

C.2 C-ISAM Format DMAM Databases

C-ISAM format DMAM databases consist of a Global schema file and one or more C-ISAM databases. The schema file, within a Global directory, holds information about the C-ISAM databases that contain the data records. The C-ISAM databases are held within the Unix filing system. A separate C-ISAM database is required for each record set in the C-ISAM format DMAM database.

C.2.1 The Schema File

The schema file consists of the following sections:

- A 256 byte area containing the name of the Unix directory which holds the C-ISAM database(s), and the name of the data library containing the conversion tables;
- 768 bytes of spare space to maintain compatibility with the header information in a Global format DMAM database;
- 1024 bytes of static database information, including a list of set-ids, which is only updated when using DBMAIN to update the data layout;
- 512 bytes containing the translation table and various other translation information;

- 1536 bytes containing the list of index names, along with associated index types and record set numbers;

- For each potential record set which the database may contain: A 386 byte area defining the Global index construction and data usage which also contains the 30 character set description as well as the 14 character Unix file name for the C-ISAM database that holds the record set;

- For each potential record set which the database may contain: A 2642 byte area defining the C-ISAM index construction and the conversion record.

Space is left for a number of record set data blocks equal to the maximum number of records sets that the database may contain (as specified by DBMAIN).

C.2.2 C-ISAM Database Structure

For details of the C-ISAM database structure please consult your Informix C-ISAM documentation.

All fields within the C-ISAM database(s) are processed by the C-ISAM Universal Channel Interface (UCI). In addition to providing the raw file access to the C-ISAM databases the UCI translates each Global record to a C-ISAM record, and vice-versa, using the UCI record conversion tables (1 per record type) and DMAM translation table (1 per DMAM database) held within the schema file.

C.2.3 Key and Field Formats for Translation and Descending Keys

Translation and Descending key fields must each be mapped to two C-ISAM fields. The first field holds the translated or descending value and is used as a C-ISAM key field. The second field holds the 'unmodified' version of the data which is NOT part of the C-ISAM key structure. When the UCI converts from Global record format to C-ISAM record format both fields in the C-ISAM record are produced. However, when the UCI converts from C-ISAM record format to Global record format only the 'unmodified' version is converted.

In the C-ISAM record, translation key segments will be substituted, byte by byte, as indicated by the translation table in the database. They will also be compacted as indicated by the table, either by bit shifting or RAD-50 compression (see appendix D).

To overcome a shortcoming in C-ISAM (i.e. it does not allow descending keys), the key-values for DMAM descending sequences are held in a special format within the C-ISAM database so that descending key fields are ordered in the correct sequence. The value of the data within the fields specified as a descending keys is modified. The actual modification depends upon the type of field and is described in the following sections.

C.2.3.1 Character fields in Descending Keys

A character string in a descending key is modified by one's complementing each byte before storing into the key area of the C-ISAM record.

Character strings can also be modified via a translation table to provide a simple character-to-character substitution, or a more complex data compression technique (as described in appendix D). A descending key field may also include translation in which case, the data is translated and then inverted as described above.

C.2.3.2 Long/Short Integer fields in Descending Keys
An integer field in a descending key is modified by one's complementing each byte of the integer, before storing into the key area of the C-ISAM record.

C.2.3.3 Float/Double fields in Descending Keys
A floating point field in a descending key is converted into an 8-byte quantity before being inverted by one's complementing each byte. The result is converted back to a floating point number before storing into the key area of the C-ISAM record.

C.2.3.4 C-ISAM Decimal fields in Descending Keys
A C-ISAM decimal field in a descending key is converted into an 8-byte quantity before being inverted by one's complementing each byte. The result is converted back to a C-ISAM decimal number before storing into the key area of the C-ISAM record.

C.2.3.5 Unix Date fields in Descending Keys
A Unix date field in a descending key is converted into a long integer variable (4 bytes) which contains the number of days since 31-Dec-1899. The result is then subtracted from 0 before storing into the key area of the C-ISAM record.

C.2.4 C-ISAM Record Locking
The C-ISAM databases that hold the data records for a C-ISAM format DMAM database may be accessed via the C-ISAM indexes created by DMAM or through any other index on the databases. However, DMAM does NOT perform any C-ISAM file or record locking on the C-ISAM databases. Consequently, these files should not be accessed by non-Global, Unix applications while they are being accessed by a Global DMAM application otherwise data corruption may occur.

Furthermore, any C-ISAM application that accesses the C-ISAM databases that constitute a C-ISAM format DMAM database must be aware of any translation and/or descending keys. If a C-ISAM application writes to a C-ISAM database that is part of a C-ISAM format DMAM database it must produce both the descending and/or translated key values as well as the original data where necessary.

C.3 Btrieve Format DMAM Databases
Btrieve format DMAM databases consist of a Global schema file and one or more Btrieve databases. The schema file, within a Global directory, holds information about the Btrieve databases that contain the data records. The Btrieve databases are held within the MS-DOS filing system. A separate Btrieve database is required for each record set in the Btrieve format DMAM database.

C.3.1 The Schema File
The schema file consists of the following sections:

- A 256 byte area containing the name of the MS-DOS directory which holds the Btrieve database(s), and the name of the data library containing the conversion tables;

- 768 bytes of spare space to maintain compatibility with the header information in a Global format DMAM database;

- 1024 bytes of static database information, including a list of set-ids, which is only updated when using DBMAIN to update the data layout;

- 512 bytes containing the translation table and various other translation information;

- 1536 bytes containing the list of index names, along with associated index types and record set numbers;

- For each potential record set which the database may contain: A 386 byte area defining the Global index construction and data usage which also contains the 30 character set description and the 12 character MS-DOS file name for the Btrieve database that holds the record set;

- For each potential record set which the database may contain: A 2642 byte area defining the Btrieve index construction and the conversion record.

Space is left for a number of record set data blocks equal to the maximum number of records sets that the database may contain (as specified by DBMAIN).

C.3.2 Btrieve Database Structure

For details of the Btrieve database structure please consult your Btrieve documentation.

All fields within the Btrieve database(s) are processed by the Btrieve Universal Channel Interface (UCI). In addition to providing the raw file access to the Btrieve databases the UCI translates each Global record to a Btrieve record, and vice-versa, using the UCI record conversion tables (1 per record type) and DMAM translation table (1 per DMAM database) held within the schema file.

C.3.3 Key and Field Formats for Translation and Descending Keys

Translation key fields must each be mapped to two Btrieve fields. The first field holds the translated value and is used as a Btrieve key field. The second field holds the 'unmodified' version of the data which is NOT part of the Btrieve key structure. When the UCI converts from Global record format to Btrieve record format both fields in the Btrieve record are produced. However, when the UCI converts from Btrieve record format to Global record format only the 'unmodified' version is converted.

In the Btrieve record, translation key segments will be substituted, byte by byte, as indicated by the translation table in the database. They will also be compacted as indicated by the table, either by bit shifting or RAD-50 compression (see appendix D).

Note that Btrieve, unlike C-ISAM (see section C.2.3, above) allows "DMAM-like" descending keys so the special field conversion algorithms within the C-ISAM UCI are not required for the Btrieve UCI.

C.3.4 Btrieve Record Locking

The Btrieve databases that hold the data records for a Btrieve format DMAM database may be accessed via the Btrieve indexes created by DMAM or through any other index on the databases. However, DMAM does NOT perform any Btrieve file or record locking on the Btrieve databases. Consequently, these files should not be accessed by non-Global, MS-DOS applications while they are being accessed by a Global DMAM application otherwise data corruption may occur.

Furthermore, any Btrieve application that accesses the Btrieve databases that constitute a Btrieve format DMAM database must be aware of any translation keys. If a Btrieve application writes to a Btrieve database that is part of a Btrieve format DMAM database it must produce both the translated key values as well as the original data where necessary.

Appendix D - DMAM Translation Tables

A translation table is held within a DMAM database file (in the header) and is used in constructing all translation keys used by any index. There is only a single translation table in a DMAM database, so if you have a requirement for more than one kind of translation you must split the record sets affected between two separate database files.

A translation table is laid out as follows:

```
01   TT                          * Translation table
  03 TTBITS    PIC 9 COMP           * type indicator
  03 FILLER         PIC X(3)        * reserved
     VALUE     LOW-VALUES
  03 TTCHAR OCCURS 256 PIC X   * translated chars
```

D.1 Translation Types

The TTBITS field indicates the type of translation to be performed. It may have the following values:

-1 which indicates that translation is to be on a byte for byte basis, with no special replacement or compaction processing. Each character in the original string is replaced by the appropriate character from the TTCHAR table (e.g. space, #20, is the 33rd element of the table and capital a, #41, is the 66th element of the table). This value is used when using the translation to re-sequence characters, as for example when collating a non-UK character set in the correct sequence;

0 which indicates that a form of RAD-50 compaction is to be used. Each character in the original string is substituted by the appropriate character from the TTCHAR table. A substituted value of #FF means that character is to be ignored. Otherwise the substituted value must be in the range #00 to #27 (0 to 39 decimal). Three substituted values are held together in a single word (2 bytes) as a number of the form 1600*A + 40*B + C, for substituted values of A, B, and C. This form of translation is used by the default table established by DBMAIN;

N a positive number, in the range 1 to 8, which indicates that bit compression is to be used. Each character in the original string is substituted by the appropriate character from the TTCHAR table. A substituted value of #FF means that character is to be ignored. Otherwise the indicated number of bits from the substituted value are appended to the values so far substituted.

In the latter two cases special processing is performed for Characters substituted as the value #00. Such characters are treated as separators, and only the first of a sequence of such characters encountered is placed in the output key value. Subsequent separator characters are ignored, until after a non-ignored, non-separator character is encountered.

In all cases where less than a full 8-bit character value results from a substitution (TTBITS is zero, or in the range 1 to 7) the

substituted characters must be either #FF, to indicate that they are to be ignored, or must contain a value no larger than indicated by the value of TTBITS (not greater than #27 for RAD-50 compaction, no more bits set than the value of TTBITS otherwise).

D.2 The Default Translation Table

DBMAIN establishes a default translation table in a DMAM database file when it is created. This table is laid out as follows:

- The value of TTBITS is zero, indicating RAD-50 compression is to be used;
- The TTCHAR characters corresponding to the ASCII numerals (elements 49 to 58 inclusive) are set to the values from #01 to #0A;
- The TTCHAR characters corresponding to the upper case letters (elements 66 to 91 inclusive) are set to the values from #0B to #24;
- The TTCHAR characters corresponding to the lower case letters (elements 98 to 123 inclusive) are also set to the values from #0B to #24;
- All other characters are set to #FF.

The effect of this table is that only ASCII letters (A-Z, a-z) and numerics (0-9) are significant in the translated key. There are no special separator characters, and letters are collated without regard to case. The collating sequence for those characters which significance is the same as it is in ASCII (i.e. 0-9 and then A-Z in increasing order).

As an example, suppose the string "14-17 Salston Av., Herts" were translated via this table. The final key value would be determined as if the string were in fact "1417SALSTONAVHERTS" (i.e. with all letters in upper case and non-significant characters removed). The key would also only occupy 12 bytes, as each three adjacent characters are compressed into a 2-byte word.

The default translation table is used by the Global 2000 applications when constructing case-insensitive 'alpha match' keys for use in name searching.

D.3 Setting up your own Translation Table

If you wish to set up your own translation table, you must arrange to create an RS file, containing a single record of length 260 bytes which is a translation table as defined above. You might wish to do this by writing a small program to create the table, or alternatively by using $PATCH on a suitable file to establish the correct values.

Before establishing a translation table you should consider what function you wish it to perform. Typically you will either wish to create some special compressed key format, or to reorder the collating sequence of characters for some non-UK language.

In the former case you must identify the subset of the character range in which you are interested. Consider carefully the question of separators, and whether any are required. The number of discrete

values required will indicate the value which should go into TTBITS, and hence the range of values which the substituted characters in TTCHAR will have. For example, if you required 25 discrete values (with or without separators) then the minimum value for TTBITS would be 5 (5 bits permits 31 values and a separator). You might wish to use a larger value of TTBITS to increase the number of possible values in the table and enable future expansion of the valid characters.

In the case of reordering the collating sequence, for a non-UK language for example, then you would arrange substitution to achieve this. You might also wish to compact the resulting keys, using the same analysis of discrete values as above.

Note that it is quite permissible for two (or more) different characters to translate to the same value (as is the case in the default table) and hence although there is only one separator value a number of characters may all translate to it.

Note also the difference between a value of 8 and -1 in TTBITS. A value of -1 does byte for byte substitution, translating all characters. A value of 8 also does byte for byte translation, but any characters translating to #00 are treated as separators and any characters translating to #FF are ignored.