PTS

programmer's guide disk file handling

M23A user library

```
Module Reference Number:          M023A—00A—12E
                                  ↑  ↑  ↑  ↑↑↑  ↑ ↑
                                  |  |  |  |||  | |
PTS ..............................|  |  |  |||  | |
Module Number .......................|  |  |||  | |
User Library ...........................|  |||  | |
Not Relevant ..............................|||  | |
Not Relevant ...............................||  | |
Complete Module .............................|  | |
Release ........................................| |
English Version ..................................|
```

A publication of

Philips Data Systems
SSS - Training & Documentation
Apeldoorn, The Netherlands

Order Number 5122 993 20432

PREFACE

This manual forms part of the documentation package to support TOSS Release 12. It describes the disk file handling functions supported by the data management packages supplied for TOSS.

In addition, it contains the information that is needed to understand the principles of data management in TOSS and the way TOSS disks are organized internally.

It is designed as reference material for application programmers. For information on the writing of CREDIT applications, refer to the other modules of the Programmers Guide, modules M21A - M25A.

The complete set of PTS documentation to support the user comprises the following modules. Modules related to the subject matter in this manual are marked with an asterisk.

M2A	A Programmers Introduction
*M4A	CREDIT Reference Manual
M5A	Device Drivers Reference Manual
*M8A	TOSS Utilities Reference Manual
M11A	DOS-PTS Reference Manual
*M21A	Programmers Guide - Elementary CREDIT
M22A	Programmers Guide - Workstation Handling
M23A	Programmers Guide - Disk File Handling
M24A	Programmers Guide - Data Communication
M25A	Programmers Guide - Workstation Management
M90A	PTS Reference Booklet
*M91A	CREDIT Reference Booklet

For an overview of the complete Training and Documentation package for PTS, please refer to the diagram on the following page.

PUBLICATION HISTORY

This version, published in June 1983, is based on the initial release of TOSS Release 12.

PTS Training and Documentation package

Training modules		Training manuals		Reference manuals	
M100	Introduction	M2A	Programmers Introduction		
M110 M111	Elementary CREDIT	M21A	Elementary CREDIT	M4A M91A M5A	CREDIT CREDIT Reference Card Device Drivers
M120 M121	Multitasking in CREDIT	(M21A	Elementary CREDIT)	(M4A (M91A	CREDIT) CREDIT)
M130 M131	DOS-PTS			M11A M90A	DOS-PTS PTS Reference Card
M150 M151 M152 M153	CREDIT Workstation handling	M22A	CREDIT Workstation handling	(M4A (M91A	CREDIT) CREDIT)
M160 M161	Disk file handling	M23A (M21A	Disk file handling Elementary CREDIT)	(M4A M8A (M90A	CREDIT) TOSS Utilities PTS Reference Card)
For basic DC training, refer to the Training Brochure					
M171	Data Communication in CREDIT	M24A	Data Communication	(M4A (M90A M15A	CREDIT) CREDIT) DC Drivers
M190 M191 M192 M193 M194	Workstation* Management	M25A	Workstation Management	(M4A (M90A	CREDIT) CREDIT)

Notes : Brackets indicate further use of a module already introduced.
Modules marked with an asterisk are not yet available.

CONTENTS

Chapter 1

INTRODUCTION

1.1 STRUCTURE OF THIS MANUAL

The manual is divided into two parts: chapters 2 up to 5 give general information related to disk file handling, while the remaining chapters contain information on the disk file handling instructions in CREDIT and the disk file handling functions supported by each package.

Chapter 2 introduces and defines the terms used in the remainder of the manual.

Chapter 3 contains information on disk formats and volume organisation of the disk types that can be used.

Chapter 4 introduces the different data management packages available in PTS and tries to give some hints on when to use these packages.

Chapter 5 is a full description of all file parameters used for the different file types handled by the data management packages.

Chapter 6 discusses the CREDIT instructions for disk file handling in detail.

Chapter 7 discusses the Abridged Data Management (ADM) package.

Chapter 8 discusses the Standard Data Management (SDM) package.

Chapter 9 discusses the Extended Data Management (EDM) package.

Chapter 10 lists and explains all the return parameters that can be obtained by each package.

Appendix A gives an estimate of the memory space needed for each package or version of a package.

Chapter 2

DATA MANAGEMENT

2.1 FILE ORGANIZATION

The information processed by a computer generally consists of large quantities of data which must be read, written and updated by an application.

Data management is concerned with the possibilities and methods of organizing data in such a way that they are accessible for different applications.

2.1.1 Files

Related data are grouped into files. To enable an application to retrieve data from the file, the information must be stored in the file according to a number of rules, defining the sequence of the data and the way of identifying them. These rules determine the file organization.

A data file need not be a contiguous area on the disk. Separate parts of the data file may reside on one or several volumes.

File Section

A file section is a continuation of the file on a different volume. File sections may reside on different disk types. Up to four file sections are allowed for a file.

File Extent

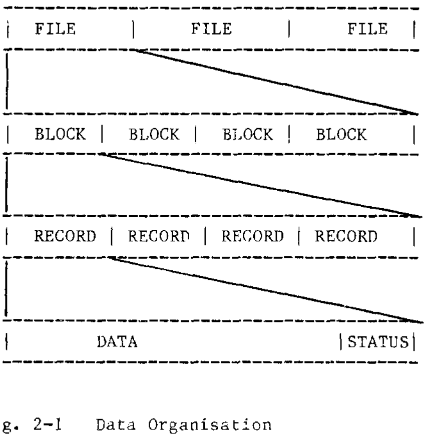
A file extent is a continuation of the file in a separate physical area on the same volume. Up to 64 extents of one file are allowed per volume. The logical sector number of the first sector of a file extent is always a multiple of 3, and the file extent length is also a multiple of 3 logical sectors and of the block length.

2.1.2 Data-records

Data items holding information on the same subject (e.g. an account-holder, or an article) are grouped into records. A file contains records of the same type. Account-holders records will reside in an account-holder file, article records constitute an article file.
In a PTS system records on a file must all have the same length.

Status Byte

Data management adds a status byte to every record. This byte indicates if a record is "used" or "free".
When a file has been created and formatted, the file is preset with empty records with a status "free". New data records written to the file overwrite these free records and the status byte is set to "used".
When a record is deleted by the application its status is set to "free".
The status byte is not included in the record length for I/O, but it must be taken into account when calculating the blocking factor.



a disk volume contains one or more files.

a file contains blocks

one block contains n records

one record contains n bytes of data, plus one status byte.

Fig. 2-1 Data Organisation

2.2 RECORD-IDENTIFICATION

Data records on a file are identified by keys. There are two methods:

- Relative Key

  The records of a file are identified by their sequence number in the file. This is the position of the record relative to the beginning of the file, and is called the relative record number or relative key. The first record in the file has relative key 1.
  Every record can always be located by its relative key.

- Symbolic Key

  For the user it may be easier to identify the record by one or more of the data items on it. These are the symbolic keys or record keys. For example, the key of an account holder's record could be the surname, the account number or the user number, and the key of an article record could be the article name or code, or the name of the supplier.

2.2.1 Index File

When the application provides a symbolic key to identify the data record that must be accessed, this has to be converted to a relative key for the system. A table is built with one entry for every record in the data file, holding the symbolic key and the relative key of the data record. This table forms an index to the data file and is called the index file.

Index Levels

In the index file there is one index entry for every data record. For a large data file, this means a long search of the index file before the reference is found.

To reduce search time for a record, the index file may be divided into parts and for every part another index entry may be created. These index entries indicate the range of symbolic keys contained by each part of the index file. Together they constitute the "master index" or the level I index. When a record must be located via a symbolic key, the master index is searched first. The found entry points to the part of the index file to be searched and here the pointer to the data record will be found.

2.2.2 Prime Key

At least one of the keys must be unique for each data record. This is the prime key. The index containing the prime key must be defined as the first index when the file is created or opened.
The other keys are called alternate keys, and these need not be unique for one data record.

2.2.3 Duplicate Keys

For the alternate keys, duplicates are allowed: the key may have the same value in several records. For indexed accesses on these records, the first one is found by an indexed direct access and the others are then accessed by indexed sequential operations.
The Return Status "Duplicate Key" will inform the application that the next record has an identical symbolic key.

2.2.4 Curreny

For every task that opens a file, data management keeps a pointer to the current record for the task. The current record is the record last read. The currency is updated by read instructions and it is not affected by write instructions.

The currency allows the application to:
- read the next record
- rewrite the current record
- discard the current record

The currency of the data file is called the Current Record Number or CRN.

If the file is indexed, data management also keeps an index currency. This points to the current index entry: the index entry used for the last read instruction via this index.

2.3 RECORD ACCESS

2.3.1 Access Method

The access method is the way to find a record in the file. If the records are identified in more than one way, there exist several access methods for the same file.

Non Indexed Access

Access on a data file without indexes may be:

- Sequential
  Records are processed in sequence of the relative key. The next record is read or written. For Read Sequential instructions this is the record following the record last read. For Write Sequential instructions this is the first free record in the file, according to the file type. File types are explained in later chapters.

- Direct
  The record to be accessed is indicated by the relative key specified by the program. Records may be accessed directly in any (random) sequence.

- Current
  For Rewrite and Discard instructions the current record may be specified.

Indexed Access

Access on an indexed data file may be:

- Indexed sequential
  The records are read in the sequence in which they appear in the index file, that is, in sequence of the symbolic record key for that index.

- Indexed direct
  The record is identified by a symbolic key, either the prime key or an alternate key, specified by the program. For some instructions this has to be the prime key (see the instruction descriptions in Chapter 6).

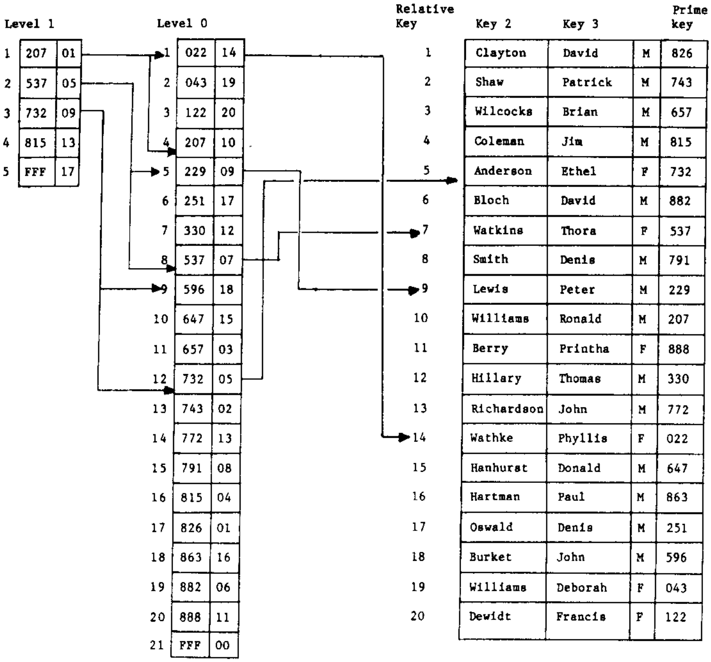
  Indexed direct read with a symbolic key specified for which duplicates exist in the file, will access the first record with that key occurring in the file, indicating "Duplicate Key" in the Return Status. The other records with this key may then be read with Read Indexed Sequential.

2.3.2 Examples

Some examples of the different access methods for the file structure in fig. 2-2:

- Sequential Access
  Sequential access on the data file will access the records 'Clayton', 'Shaw', 'Wilcocks', and so on, in the order in which they appear in the data file.

- Indexed Sequential Access
  To access the record in nummeric order of the customer number, which is the prime key, they may be accessed via the index. The records will than be read in the following order: Phyllis Wathke 022; Deborah Williams 043; Francis Dewidt 122; Ronald Williams 207; and so on.



Level 1

1	207	01
2	537	05
3	732	09
4	815	13
5	FFF	17

Level 0

1	022	14
2	043	19
3	122	20
4	207	10
5	229	09
6	251	17
7	330	12
8	537	07
9	596	18
10	647	15
11	657	03
12	732	05
13	743	02
14	772	13
15	791	08
16	815	04
17	826	01
18	863	16
19	882	06
20	888	11
21	FFF	00

Relative Key	Key 2	Key 3		Prime key
1	Clayton	David	M	826
2	Shaw	Patrick	M	743
3	Wilcocks	Brian	M	657
4	Coleman	Jim	M	815
5	Anderson	Ethel	F	732
6	Bloch	David	M	882
7	Watkins	Thora	F	537
8	Smith	Denis	M	791
9	Lewis	Peter	M	229
10	Williams	Ronald	M	207
11	Berry	Printha	F	888
12	Hillary	Thomas	M	330
13	Richardson	John	M	772
14	Wathke	Phyllis	F	022
15	Hanhurst	Donald	M	647
16	Hartman	Paul	M	863
17	Oswald	Denis	M	251
18	Burket	John	M	596
19	Williams	Deborah	F	043
20	Dewidt	Francis	F	122

Fig. 2-2 Indexed File

2.4 BLOCKING

Blocking is grouping records into larger units that are transferred during one disk access, because transferring one record per disk access is in most cases not efficient. Transfer always starts on a logical sector boundary, so if the records are shorter or a little longer than 255 bytes, large areas remain unused.

For every transfer the read-write head is positioned at the required sector and transferring the records one by one means that a search time is needed for every record. This takes more time than is necessary, especially when the records are processed sequentially. Better use of time and disk space is made by "blocking" the records. A block is a number of records transferred during one disk access.

2.4.1 Blocking Factor

The number of records per block is the blocking factor. The blocking factor is chosen by the user when the file is created. To choose a blocking factor by which the most efficient use is made of the available disk space, it must be noted that:

- The system adds one status byte to every data record (except for L and X files, see chapter 7). When calculating the blocking factor, 1 must be added to the record length.
- Logical sector length is 256 bytes.
- Blocks always start on a sector boundary, but they may have a block length of several logical sectors.
- File extents always start on a sector with a logical sector number which is a multiple of three.
- The most effective disk access time is obtained when 3 logical sectors are read or written in one access, especially when 16+80 Mb disks are used in the PTS6000 system.
- For large blocks, large block buffers are needed in memory.

2.4.2 Examples

When the record length is 40 bytes a blocking factor of 6 uses 6x(40+1)=246 out of 256 bytes per logical sector.

When the record length is 150 bytes a blocking factor of 3 uses 3x(150+1)=453 bytes out of 512, and every block occupies 2 logical sectors. A blocking factor 5 uses 5x(150+1)=755 out of 768 bytes and every block occupies three logical sectors. The most efficient blocking factor in this case is 5.

2.5 DATA MANAGEMENT FUNCTIONS

2.5.1 File Handling Functions

Data management supports the following functions:

Create File

A new file may be created during runtime. The file must be opened for "Output only" and the application must supply the necessary information such as file name, volume where the file must reside, file size, record length, blocking factor, and the definition of symbolic keys if it is an indexed file. In that case the index filesare also created. As much space as is requested is reserved on the disk and formatted with "free" records.

The following functions can be executed for an existing file:

Open File

An Open file instruction is necessary to initiate a file for access by a task.

Disk files are held on a volume which may contain several different files. Also there may be more than one disk volume on-line at the time, and the data file may have index files to it that reside on a different volume. Therefore, an Open instruction must be executed to tell the Monitor which file is to be opened, on which volume(s) the file exist, how many indexes are to be used and on which volume the index files are found, before the records of a file can be used by the application.

Data Management checks if the task is allowed to open the file, and if there is space in memory for block- and record buffers, currency and protected-record administration.
If all requirements are met the file is opened for the task.

Read Record

Read Record is the instruction to read data from the file. The records are read into the application record buffer.

Rewrite Record

If in the course of a transaction some of the data in a record must be changed, this is done by the application in the application record buffer. The updated record is then rewritten to the file, where it overwrites the old one.

Discard Record

A record which is no longer needed can be discarded. The data is not physically removed from the disk but the status of the record is changed to "free". A free record can not be read by the application.

Write Record

A new data record may be written to the file from the application record buffer. A new record can only be written to a free record in the file.

Extend File

If a number of new records must be written to an existing standard file, the file may be opened for Extend. New records are written to the free part at the end of the file.

Close File

When the task no longer requires access to the file, it must close the file. The space reserved for buffers and administration within the Monitor becomes available for other tasks or other files to be opened. It is especially important to close files which were opened for exclusive access by the task as soon as possible, so that a second task may then open the file.

Delete File

A file no longer needed may be deleted from the disk. The VTOC record for the file will then get the status "free" and the file can no longer be accessed. Only a file that has been opened for exclusive access by a task can be deleted by that task.

2.5.2 Sharability

A number of tasks may be using records of the same files at one time. There must be a protection against simultaneous updating of records by different tasks. Protection is possible on record level and on file level.

Record Protection

- Unprotected
  There is no restriction on concurrent use of the same records by other tasks. "Unprotected" is only allowed when the file has been opened for input only (the records can only be read, not updated or discarded).

- Protected
  When a file is opened protected, a task will hold the records it accesses under Protected Access. No other task can access the record. Other tasks may still access other records on the same file.

  The records are released when the task issues a transaction control instruction (see section 2.7) or closes the file, or when data management releases the records automatically to prevent a deadlock situation.

File Protection

- Exclusive
  Protection on file level means that the file is attached to the task, and no other task can access records of this file. A task can obtain exclusive access to a file by specifying sharability Exclusive when the file is opened. Exclusive access to a file is released by a Close instruction.

  Sharability Exclusive must be specified when the file is created or extended, and when it is to be deleted.

2.5.3 Data Set Declaration

Data management files to be accessed by an application must be defined by a DSET declaration in the data division, in the same way as other I/O devices. This is described in the CREDIT Programmer's Guide for Elementary CREDIT, module M21A.

The DSET declaration links the data set identifier used by the application to the TOSS file code specified during Monitor generation. Data Management file codes must be defined in Special Device Classes.

It is not possible to have common files in CREDIT applications.

2.6 FILE ENLARGEMENT

File enlargement is the addition of another file extent. Both non-indexed and indexed (not in ADM) files can be automatically enlarged during runtime. Automatic enlargement takes place when during Write instructions the last record of the file is written. For the details of automatic enlargement, which are different for each data management package, refer to chapters 7, 8 and 9.

2.6.1 Growth Factor

The size of the added file extent in the case of automatic enlargement is determined by the Growth Factor in the File Descriptor Block. The Growth Factor is set by the user when the file is opened. It represents a percentage of the size of the file when it is opened. From this, the number of records by which the file must be extended is calculated by data management. This number is then rounded upward to obtain a file extent length which is a multiple of three logical sectors and of the block length.

If after the file has been enlarged the end of file is reached again by Write instructions, the file is enlarged again by the same number of logical sectors, for it is the same percentage of the length of the file when it was opened.

The number of sectors by which the file will be enlarged is changed when the file is closed and opened again. A different Growth Factor may then be specified. However, if the Growth Factor remains the same, the percentage will be taken from the new file size and also result in a different file extent length.

Files are only extended if the Growth Factor specified is not zero.

If the file can not be enlarged, the message End of Medium is returned when the end of the file is reached by Write instructions. The file can not be enlarged if:

- The Growth Factor is zero
- There is no free VTOC record available for the new file extent
- The maximum number of file extents (64) on a volume has been reached and no next volume is available
- The maximum number of file extents and file sections has been reached

2.6.2 Example

A file with a size of 200 records is opened. The Growth Factor specified is 10. During Write instructions, the end of the file is reached. A new file extent is automatically created by data management, with a size of 10% of 200 records =20 records.
The sequential write operations are continued until the new end of file is reached. Another file extent is added, with a size of 10% of the original file size (200 records) so again 20 records.

Then the file is closed, and opened again. The size is now 240 records. The Growth Factor specified is still 10. Automatic enlargement will be by 10% of 240 =24 records.

If, when the file is opened for the second time, a Growth Factor of 5 is specified, automatic enlargement will be by 5% of 240 records = 12 records.

Note that this is only an example to explain the mechanism. In reality it would not be advisable to create such small file extents. Also, the size of the additional file extents will be rounded upwards to a multiple of three logical sectors.

2.7 TRANSACTION CONTROL

In a business environment, applications will mostly be designed to execute transactions. A transaction is an elementary business operation. A transaction may include one or more reads, writes and updates of data records in a number of files.

Before and after a transaction the data in the files are in accordance with each other and reflect a real situation. It is said that the files are in a state of integrity. While the transaction is in progress, the files are not in a state of integrity.

2.7.1 Integrity Unit

A series of record accesses on files, at the beginning and end of which the files are in a state of integrity, is called an integrity unit.

One transaction may consist of one or more integrity units.

Example

A data file contains account holders records. A certain amount of money must be withdrawn from the balance of Mr. X and paid into the bank account of Mr. Y.
Before the transaction Mr. X has the money and Mr. Y has not, which is a real situation. Halfway through the transaction, when the money has been withdrawn from the balance of Mr. X and not yet added to the balance of Mr. Y, the data in the file are not in a state of integrity as they do not reflect a real situation.
When the record for Mr. Y. has also been updated the file is again in a state of integrity and the transaction is completed.

The transaction in this example consists of one integrity unit.

2.7.2 Transaction Control Functions

The beginning and the end of a transaction are marked by transaction control functions (Commit). All accesses on the files executed between two transaction control functions belong to one transaction.

Several records of several files may have to be updated during one logical transaction. Before the transaction and after it, when all updates belonging to one logical transaction have been executed completely or not at all, the files are in a consistent state.

Points in the application program where files are in a consistent state are defined by the transaction control functions Commit and Rollback.

COMMIT terminates a transaction or subtransaction. The updates performed are "committed" to the data file, this means that it is no longer possible to undo the transaction. The records involved are released and may now be used by other tasks.

See also the detailed descriptions of transaction control functions for the different packages.

Chapter 3

VOLUME ORGANIZATION

3.1 DISK TYPES

A volume is a single physical unit capable of holding information. In this manual, disk volumes are considered and secondary storage media that are organised in the same way as a disk, the CMOS memory and the "simulated disk in primary memory".

3.1.1 Disk Volumes

The following disk types are available for PTS systems:

PTS 6875 - 2.5 Mb fixed and cartridge disk
PTS 6876 - 5 Mb fixed and cartridge disk
PTS 6877 - 80 Mb fixed and cartridge disk
PTS 6961 - 16 Mb fixed and cartridge disk
PTS 6879 - 0.25 Mb flexible disk
PTS 6791 - 1 Mb flexible disk
PTS 6792 - 0.25 Mb flexible disk
PTS 6962 - 16 Mb + 80 Mb fixe and cartridge disk

3.1.2 CMOS Memory

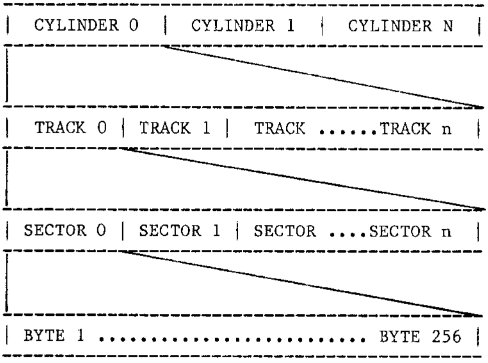
The CMOS memory is approximately 8kb of CMOS RAM, connected to the PTS6911 Workstation Controller (WS11). It can be considered as a peripheral with a very fast access time, and organised in the same way as a disk. The CMOS memory can only contain valid information when the power is on. A back-up battery maintains the contents of the memory during about 48 hours after a power failure. After that time the contents of the CMOS memory are undefined.

3.1.3 Simulated Disk in Primary Memory

A part of the memory may be reserved for use as a very fast access storage. The size is a multiple of 4k bytes and is defined during Monitor generation. Using the simulated disk in primary memory for example for transaction logging will improve the system performance.

3.2 DISK STRUCTURE

Each disk volume is divided into cylinders, each cylinder into tracks, and each track into sectors. This structure is transparent to the application, for the program only addresses records within a file. The programmer must be aware of this structure when constructing files, for it affects the blocking factor, number of file extents on the volume, or number of volumes required for one large file.



1 volume contains n cylinders

1 cylinder contains n tracks

1 track contains n logical sectors

1 logical sector = 256 bytes

3.2.1 Sectors

Disk sectors as seen by the hardware are different from those seen by the software. The hardware is concerned with physical sectors on the disk, while the software handles logical sectors.

- A physical sector is the unit of information transferred between the disk and the primary memory.

- A logical sector is the unit of information transferred between the disk driver and data management or the application. Depending on the disk type, the logical sector length may be different from the physical sector length.

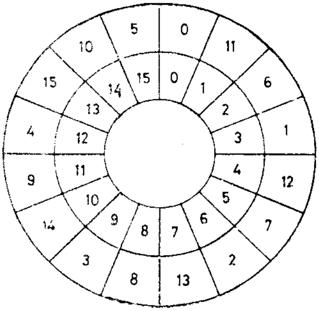
All the disk drivers handle multiple logical sector I/O, so it is possible to transfer more than one logical sector with one I/O request.

Physical and logical sector numbering may be different. This depends on the way in which the disk is premarked.

3.2.2. Interlacing

On many disk types the logical sectors are not numbered in physical sequence, but interlaced.
The figure below is an example. The inner ring of numbers represent the numbering of the physical sectors, the logical sectors are interlaced as shown by the outer ring of numbers.

Interlacing is done to save access time on the disk. After a sector has been transferred, the software needs some time to process it before the next sector can be transferred. During this time the disk keeps rotating and when the next transfer can be done the disk head will not be positioned at the adjacent sector, but further. Marking this as the next logical sector avoids waiting for almost one disk rotation.



Interlacing is fully transparent to the user, and sectors are always processed in sequence of logical sector number. The number and length of physical sectors, and the interlacing factor, depend on the disk type.

3.2.3 PTS6875 and PTS6876 Disk Versions

There are two format versions for the disks PTS6875 and PTS6876, the Packed and the Unpacked version. One of them is selected when the disk is formatted by the TOSS utility Create Volume (CRV).

Version 2 - Unpacked
There is one logical sector per physical sector, which provides lower average access times.

Version 3 - Packed
Two physical sectors contain three logical sectors. This version makes more efficient use of disk space. An intermediate buffer is used during transfers, unless both the number of the first logical sector and of the total number of sectors to be transferred are multiples of three. In that case the intermediate buffer is not needed, and access times are lower than for the unpacked version.

Which version is used for the disk need only be specified when the disk is formatted with the TOSS usitility CRV. The information is stored in the Volume Label and read into memory when the disk is on line.

3.3 SURVEY OF DISK CAPACITY

Disk Type	PTS6875		PTS6876	
Format Version	2	3	2	3
Number of cylinders	204	204	408	408
Tracks per cylinder	2	2	2	2
Physical sectors /track	16	16	16	16
Logical sectors/track	16	24	16	24
Bytes/physical sector	258	386	258	386
Bytes/logical sector	256	256	256	256
Maximum logical sector no	6323	9791	12647	19583
Maximum physical sector no	6527	6527	13055	13055

Disk Type Disk Version	PTS6879 flexible disk TOSS	 IBM	PTS6877 "80Mb" disk	PTS8863 mini fixed disk
Number of cylinders	77	77	822	255
Tracks per cylinder	1	1	5	2
Physical sectors/track	26	26	23	52
Logical sectors/track	13	26	69	52
Bytes/physical sector	128	128	768	128
Bytes/logical sector	256	128	256	256
Maximum logical sector no	1000	1932	283589	23399
Maximum physical sector no	2001	73026*	94529	23399

Disk Type Disk Version	PTS6961 "16+16" CMD disk	PTS6791 "1 Mb" flexible disk
Number of cylinders	822	77
Tracks per cylinder	1	2
Physical sectors/track	23	26
Logical sectors/track	69	13
Bytes/physical sector	768	256
Bytes/logical sector	256	256
Maximum logical sector no	56717	3990
Maximum physical sector no	18905	4003

* For the IBM format on the PTS6879 flexible disk, physical sector number has format 'tt0ss', where tt is track number, 0 is zero, and ss is the sector number. 73026 corresponds to logical sector number 1923.

3.4 RESERVED AREAS

On cylinder zero, some sectors are reserved on track zero for use by the system software. If this track contains bad spots the volume can not be used.

Track zero on cylinder zero contains:

- Volume Label in sector zero; the identification of the volume.

- IPL loader in sectors starting in sector 1. The number of sectors occupied is device dependent.

- Free Space Administration Table (FSAT)
  When a file must be created or extended, the Free Space Administration Table is searched for free areas large enough to contain the required number of sectors.

- Volume Table of Contents (VTOC), containing the identification of all file extents on the volume.

Volume label and IPL are written by the utility Create Volume (CRV). After that, sectors are reserved for the VTOC and the FSAT. The number of sectors occupied by VTOC and FSAT depends on the number of VTOC entries specified when the utility is run.

The total number of sectors in the reserved area must be a multiple of three. For example, if Volume Label and IPL together occupy 4 sectors, the VTOC and FSAT will be a multiple of 3 sectors plus 2.
CRV creates an FSAT with at least the same number of entries ad the number of entries in the VTOC. The number of VTOC entries per sector is 6, the number of FSAT entries per sector is 32.

This results in the following number of entries in the FSAT:

No of VTOC entries	VTOC sectors	FSAT sectors	No of FSAT entries
1 - 6	1	1	32
7 - 12	2	3	96
13 - 18	3	2	64
19 - 24	4	1	32
25 - 30	5	3	96

The layout of Volume Label, Free Space Administration records and Volume Table of contents is described in the following sections.

3.4.1 Volume Label

The volume label is located on cylinder 00, track 00 sector 00 of all disk types.

The format of the volume label is shown below:

byte		
1 2 4	VOLUME NAME	
6	FSAT + VTOC LENGTH	
8	not used	
10	FSAT BASE	
12	VTOC RECORD LENGTH	
14 16 18 20 22 24	12 NC NUMBER	
26	NUMBER OF CYLINDERS	
28	NUMBER OF TRACKS	
30	NUMBER OF SECTORS / TRACK	
32 34 36 38 40 42 44	RELEASE NUMBER	
46	FSAT LENGTH	
48	FORMAT	DEVICE TYPE
100 - 139	IPL	

The fields have the following meaning:

VOLUME NAME
A string of 6 characters, left adjusted and padded with spaces. Spaces are not allowed within the volume name. Volume names must be unique within a system.

FSAT + VTOC LENGTH
Number of sectors occupied by the Free Space Administration plus Volume Table of Contents.

FSAT BASE
The address of the sector on which the FSAT starts.

VTOC RECORD LENGTH
The size of one VTOC record, in bytes. The status character is not included.

12 NC NUMBER
Reserved for the 12 NC number of the disk.

NUMBER OF CYLINDERS
The number of cylinders available for the user

NUMBER OF TRACKS
Number of tracks per cylinder. This is equal to the number of disk surfaces that are used.

NUMBER OF SECTORS PER TRACK
This is the number of logical sectors per track.

RELEASE NUMBER
This field contains the text "TOSS RELEASE xx.yy" where xx is the release number and yy is the level.

FSAT LENGTH
Number of sectors used for the administration of free areas on the volume.

FORMAT
Device dependent parameter indicating which interlacing pattern is used to map the sectors on the physical addresses.

DEVICE TYPE
An 8 bit integer indicating:

1 =	PTS 6875	2.5 Mb disk
2 =	PTS 6876	5 Mb disk
3 =	PTS 8863	6 Mb mini fixed disk
4 =	PTS 6877	80 Mb disk
5 =	PTS 6961	16 Mb disk
6 =		Simulated disk in CMOS memory
7 =	PTS 6872	0,25 Mb flexible disk
8 =	PTS 6879	0,25 Mb flexible disk
9 =	PTS 6791	1 Mb flexible disk
10 =		Simulated disk in primary memory

IPL
IPL is device dependent code used by the Initial Program Loader for that disk type.

All unused fields contain hexadecimal zeroes (X'00').

3.4.2 Free Space Administration Table

The Free Space Administration Table (FSAT) entries describe the start address and length of all free extents on the volume. The format of each entry is shown below. Unused entries contain all zeroes (X'00').

```
byte    ------------------------------
 1     |                              |
       |        EXTENT LENGTH         |
 2     |                              |
       |------------------------------|
 4     |                              |
       |         EXTENT BASE          |
 6     |                              |
        ------------------------------
```

The fields have the following meaning:

EXTENT LENGTH

The size of the free extent, in number of sectors.

EXTENT BASE

Logical sector number of the first sector in this free extent.

3.4.3 VTOC Records

Each VTOC record is 42 bytes long (41 bytes data + 1 status byte) and they are blocked 6 per sector. One record exists for every used extent in each file.

The VTOC is accessible only through Monitor routines. The Read File Parameter instruction will make the information about the file available to the application.

VTOC records have the following layout:

byte		
1 2 4	FILE NAME	
8	FILE SECTION NUMBER	
10	FILE EXTENT NUMBER	
12 14	FILE EXTENT LENGTH	
16 18	FILE EXTENT BASE	
20 22	LAST RECORD NUMBER	
24	RECORD LENGTH	
26	BLOCKING FCT	FILE ORG
28 30 32	CREATION DATE	
34	RETENTION PERIOD	
36		NO OF INDEXES
38	KEY ADDRESS	
40	NO OF EXTENTS	STATUS

The fields have the following meaning:

FILE NAME
A string of 8 characters, left-adjusted and padded with spaces. This field must be set to spaces (X'20') for unused entries. No spaces are allowed within the File Name. Some file names are reserved for use by the system, see section 3.6
On one volume, all VTOC entries with the same file name are regarded as describing extents of the same data file.

FILE SECTION NUMBER
A binary value numbering the file sections. File section numbering starts from zero.

FILE EXTENT NUMBER
A binary value numbering the extents within each file-section. File extent numbering starts from zero.

FILE EXTENT LENGTH
A binary value representing the number of sectors in the file extent.

FILE EXTENT BASE
A binary value representing the logical sector number of the fist sector in the extent.

LAST RECORD NUMBER (LRN)
For Standard files, LRN is a binary value which is the relative key of the last used record in the file written by Write Sequential instructions. There may be "free" and "used" records in the file between the LRN and the end of the physical file area.

For indexed files of S-type, LRN is the relative key of the last data record written by Write Indexed Direct instructions.

For an index file of S-type, LRN is the relative key of the last index record in the last used partition.

For a master index file of S-type, LRN ids the relative key of the last used master index record.

For EDM files (both D and I files), LRN is the relative key of the first record in the free record chain.

For L and X files, LRN is a binary value representing the number of used sectors.

RECORD LENGTH
A binary value representing the number of bytes per record. This is a fixed value for all records in the file and does not include the status byte. For L-files the record length is always 256, for X files it is a multiple of 256.

BLOCKING FACTOR
A binary value representing the number of records per block. For I, L and X files the blocking factor is always 1.

FILE ORGANISATION
One character indicating the file type:
  S - File of S-type
  D - the data part of an EDM (E) file
  I - for the index part of an EDM (E) file
  L - library file or load file
  B - bad spot file
  X - non-standard file

NUMBER OF INDEXES
For the indexed files of S-type this is a binary value representing the number of index files belonging to this data file.
For other file types the field contains zero.

KEY ADDRESS
A binary value representing the first character position of the symbolic key in the data record. This field is only used for index files of S-type and is set to zero for other file types.

NUMBER OF EXTENTS
A binary value representing the number of extents of the file that contained on this volume. This number is only set in the VTOC record of the first extent.

STATUS
A single character indicating whether this VTOC record is used (X'FF') or free (X'00'). This character is not included in the VTOC RECORD LENGTH defined in the volume label.

CREATION DATE
A string of 6 ISO-7 characters representing the creation date of the file. Recommended format can be either YYMMDD or YYDDD, left adjusted, where
 YY = last two digits of the year
 MM = month of the year
 DD = day of the month
DDD = day of the year

The format is not checked by the system software.

RETENTION PERIOD
A string of 3 ISO-7 characters representing the number of days that this file is to be retained.
The contents of this field is not checked or used by the system software.

3.5 VOLUME CREATION

A disk volume to be used on the PTS system must be initialised and formatted by the TOSS utility Create Volume (CRV). This utility is described in the TOSS Utilities Reference Manual, module M8A.

The utility writes a Volume Label and an empty VTOC, and FSAT and an IPL on track zero. Defective sectors are assigned to a badspot file.

3.5.1 BADSPOT file (B-file)

A Badspot file is a dummy file which includes all defective sectors on a volume, so that these are not used for the real files. When a new volume is formatted with the TOSS utility Create Volume, the quality of each sector is checked. Unusable sectors are included in a B file, registered as such in the Volume Table of Contents and withdrawn from the Free Space Administration Table.
The BADSPOT file can contain up to 18 extents. If the disk contains too mny defective sectors, it can not be used.

3.6 RESERVED FILE NAMES

File names exists of up to 8 ISO-7 characters. The rules for the file names of indexed file structures in SDM and EDM and the rules for L file names are described below.
Within TOSS systems, some file names are reserved.

3.6.1 Standard Data Management Files

The file name of the data file is specified when the file is created. It may consist of up to 8 characters, the first 6 are significant and must be unique. The file names of the index files consist of "In" followed by the first 6 characters of the data file name, where "n" is the number of the index. The file names of the master files consist of "Mn" followed by the first 6 characters of the data file name, where "n" is the number of the index.

3.6.2 Extended Data Management Files

The file name of the D file is specified when the file is created. It may consist of up to 8 characters, the first 6 are significant and must be unique. The file name of the I file consists of "I$" followed by the first 6 characters of the D file name.

When EDM is used, the following file names are reserved: "TLOGFILE", "FLOGFILE", "I$0000" and file names starting with "$$$$".

3.6.3 System File Names

File names with the format "$XXXX:nn" are reserved for load files and configuration files and must not be used for TOSS files.

"XXXX" represents up to four ISO-7 characters and "nn" are two numeric ISO-7 characters.

Chapter 4

TOSS DATA MANAGEMENT PACKAGES

Three data management packages are available for PTS:

- Extended Data Management (EDM)
- Standard Data Management (SDM)
- Abridged Data Management (ADM)

A system may contain either EDM or SDM. ADM may be included on its own or together with EDM or SDM, to handle the file types that EDM or SDM cannot handle (library files and undefined files).

The data management package required is selected during Monitor generation.

4.1 EXTENDED DATA MANAGEMENT

Extended Data Management is a data management package for indexed file handling. EDM supports transaction control and file recovery by means of function logging.

EDM supports the following features:

- Standard files and indexed files of E-type are handled.
- Up to 10 indexes may be defined for one data file
- Symbolic record keys consisting of up to 16 separate items (concatenated keys)
- Conditional indexing
- Re-use of deleted records in indexed files
- Transaction control functions COMMIT and ROLLBACK to take care of file consistency
- Transaction logging
- Automatic transaction rollback in the case of deadlock or fatal I/O errors.
- Function logging
- The possibility to create, delete and extend files

4.1.1 Versions of EDM

Three versions of EDM are available:

- Version 1
  A complete EDM package, disk resident, which includes all the functions listed above.

- Version 2
  A complete EDM package, primary memory resident.

- Version 3
  A primary memory resident subset, not segmented, which does not include transaction logging and function logging.

4.2 STANDARD DATA MANAGEMENT

Standard Data Management (SDM) is a separate data management package. It is primary memory resident. Indexed files are supported but index handling is less powerful than in the EDM package.

SDM supports the following features:

- Standard files and indexed files of S-type are handled
- Up to four indexes are allowed per data file
- Keys consist of one data item
- Two levels of indexing
- The possibility to create and extend indexed and non-indexed data files
- The possibility to delete files.

SDM is upward compatible with EDM in that the instruction set supported by SDM is a subset of the instructions available in EDM.

An indexed file structure for SDM, however, can not be handled by EDM, unless it is converted into an EDM file structure with the TOSS utility Copy File to File, copying the data file of S-type to a previously defined E file. The index part of the E file will be built according to the definition of a I file.

4.3 ABRIDGED DATA MANAGEMENT

Abridged Data Management is a data management package for file handling on logical sector level.
Record length + 1 must be a multiple of 256.
In ADM the following features are implemented:

- Standard files, L files and X files can be handled
- Files can be opened for exclusive access
- Files can be created, extended and deleted.
- Direct access on files.
- Sequential access only for Write instructions on Standard files.

The instructions available for ADM are a subset of the instruction set for SDM.

4.4 COMPARISON

Which data management package is used depends on the application requirements, memory space available, and the organisation of the files to be processed.

SDM - EDM

The main reasons for using EDM are:
- Logging and recovery functions
- Powerful index handling
- Support of large indexed files and a high updating frequency without the need for reorganizing the files.

SDM may be preferred to EDM when rather static files with up to four indexes are handled, for performance reasons: it may be faster and requires less memory space.

The CPU load of EDM is approximately 3 times as much as the CPU load of SDM for similar functions. The number of disk accesses is about the same for SDM and EDM without logging functions. However, EDM needs more disk accesses for direct access instructions because it does not have a common block buffer pool.

The logging functions of EDM increase the number of disk accesses. Transaction logging on the simulated disk in primary memory will improve the performance of EDM.

SDM - ADM

ADM is used for standard files when the application itself does the record handling and access on logical sector level is sufficient. ADM must be used to handle L and X files.

Compared to SDM, Abridged Data Management has the following restrictions:

Not supported are:

- Physical I/O. Only basic write, without checking, is implemented.
- Sequential access, except Write Sequential for Standard files.
- Current access. ADM does not maintain a currency.
- The POSIT instruction to position the currency.
- Indexed files.
- Record protection.
  File protection however is supported, it is possible to open a file with sharability Exclusive.
- The Delay option

- Transaction control functions. However, it is allowed to use dummy COMMIT instructions for compatibility with SDM.
- No file recovery is possible after a disk failure or a system halt.

The aproximate memory space requirements of each package are found in Appendix A.

4.5 SUMMARY

This diagram gives an overview of the file types and functions supported by ADM, SDM and EDM.

Y = supported
- = not supported.

	ADM	SDM	EDM1,2	EDM 3
FILE TYPES				
E file	-	-	Y	Y
indexed fileof S-type	-	Y	-	-
Standard file	Y	Y	Y	Y
L file	Y	-	-	-
X file	Y	-	-	-
FILE HANDLING INSTRUCTIONS				
OPEN FILE				
for: input	Y	Y	Y	Y
input-output	Y	Y	Y	Y
output sequential	Y	Y	Y	Y
output direct	Y	Y	Y	Y
output extend	Y	Y	Y	Y
SHARABILITY				
unprotected	Y	Y	Y	Y
protected	-	Y	Y	Y
exclusive	Y	Y	Y	Y
CLOSE FILE				
lock	Y	Y	Y	Y
discard	Y	Y	Y	Y
POSITION currency				
direct	-	Y	Y	Y
indexed direct	-	Y	Y	Y
COMMIT	-	Y	Y	Y
protected	-	-	Y	Y
with release	-	Y	Y	Y
ROLLBACK	-	-	Y	Y
release to prevent deadlock	-	Y	Y	Y
LOGGING				
transaction	-	-	Y	
function	-	-	Y	

	ADM	SDM	EDM1,2	EDM 3
RECORD HANDLING INSTRUCTIONS				
READ				
sequential	-	Y	Y	Y
direct	Y	Y	Y	Y
indexed sequential	-	Y	Y	Y
indexed direct	-	Y	Y	Y
WRITE				
sequential	Y	Y	Y	Y
direct	Y	Y	Y	Y
indexed sequential	-	-	Y	Y
indexed direct	-	Y	Y	Y
REWRITE				
current	-	Y	Y	Y
direct	Y	Y	Y	Y
indexed direct	-	Y	Y	Y
DISCARD				
current	-	Y	Y	Y
direct	Y	Y	Y	Y
indexed direct	-	Y	Y	Y
READ FILE PARAMETERS	Y	Y	Y	Y
READ STATUS	Y	Y	Y	Y

Chapter 5

FILE PARAMETERS

5.1 INTRODUCTION

The characteristics of a file are defined by a number of parameters such as file name, record length, blocking factor. These parameters are stored in the VTOC record of the first file extent. For an E file, there is an index descriptor stored in the first block of the I file.

The file parameters are transferred from the application to data management and vice versa. File parameters are contained in the File Parameter Block.

When a new file is created by data management, the application must provide most of the file parameters and the complete index description.

When opening an existing file the application must provide those file descriptor items that are subject to change, such as Protection, Growth Factor, the I/O Option, and the index descriptors of the indexes to be opened.

The fields that are not relevant are "reserved" and must contain zeroes.

After opening a file, the file parameters can be obtained by by the application with the DSC instruction to read the File Parameters (code X'19'). The file parameters can be printed offline with the TOSS utilities Print VTOC (PVC) and Print Descriptor Block (PDB), which are described in the TOSS Utilities Reference Manual, module M8A.

The file parameters and index descriptors are discussed in the following sections. All the parameters used in ADM, EDM and SDM are listed here. Which parameters are required for each instruction is explained in Chapter 6 of this manual and also found in the corresponding instruction references in the CREDIT Reference Manual, module M4A.

5.2 DATA FILE PARAMETERS

The following parameters apply for all E, S, L and X files.

Record length

The length of the data records in bytes. The status byte is not included in the record length. The Record length may have any value from 1 to 2047 for standard files or from 4 to 2047 for E files.
The maximum record length for a file created and handled by TOSS utilities is 2047 bytes.

Blocking factor

The number of data records per block in the data file. The blocking factor may have any value from 1 to 255. The maximum block length in SDM is 2047 bytes.

File organization

This field indicates the file type.
0 = Standard file or data file of S-type opened without indexes.
1 = E file or indexed file of S-type
For ADM it can also have the values:
2 = L file (load file)
3 = X file (undefined file)

Device type

This field must contain the value 1 to indicate disk.

I/O option

This field indicates the output mode selected for the file when it was opened by the program:
0 = Physical read/write with read after write check.
1 = Basic read/write, no check.
For SDM:
2 = Physical read/write and delay: I/O is performed to an internal block buffer and not immediately to disk. (See also 8.7.2).
3 = Delay and basic read/write.

Reserved

A field where File Management stores the internal identification of the data file and index file. This field is only used by some of the TOSS utilities.

File name

Data file name, consisting of one alphabetic character followed by 0 to 7 alphanumeric characters. System reserved file names are found in section 3.6.

Logging type (only EDM)

0 = no logging done for this file.
1 = transaction logging.
2 = function logging.
3 = transaction logging and function logging.
For SDM and ADM this field must contain zero.

Growth factor

For E- and S- files, this field contains the percentage of the initial file size by which the file must be extended when during a Write instruction the end of the file is reached.

Data volume name 1

The name of the first volume where the data file resides. Volume name consists of up to 6 alphanumeric characters.

File section size 1

Number of data records residing on the first volume.

Data volume name 2

The name of the second volume where the data file resides. Volume name consists of up to 6 alphanumeric characters.

File section size 2

Number of data records residing on the second volume.

Data volume name 3

The name of the third volume where the data file resides. Volume name consists of up to 6 alphanumeric characters.

File section size 3

Number of data records residing on the third volume.

Data volume name 4

The name of the fourth volume where the data file resides. Volume name consists of up to 6 alphanumeric characters.

File section size 4

Number of data records residing on the fourth volume.

5.3 L AND X FILE PARAMETERS (only ADM)

For L and X files, the values of some file parameters are fixed:

Record length — Must be 256 for L files or a multiple of 256 for X files.

Blocking factor — must be 1.

File organization — 2 = L file
3 = X file

The following items are added to the File Parameter block:

File record number — The relative key of the last record of the file written by Sequential Write instructions.

Number of users — The number of successful Open instructions that have been performed for this file on this file code. At a Close instruction, this number is decreased by one.

Protection — 0 = file is opened for shared access.
1 = the file is opened for exclusive access

Creation date — A string of 6 characters representing YYMMDD or YYDDD left adjusted. The format will not be checked by the system.

NOTE:
File Record Number, Number of Users and Protection are used by the system and must not be set or updated by the application!

Retention period — A string of 3 characters representing DDD. The format will not be checked by the system.

Reserved — This field must contain binary zeroes.

5.4 INDEX FILE PARAMETERS

The following parameters define the indexes for E files and indexed S file structures. Fields that are not relevant must be set to zero, for example all the fields defining a conditional index must be set to zero for indexed files of S-type.

Index volume name

Name of volume where the index files and master index files reside for an indexed file structure for the SDM package, or the I file for the EDM package. Volume name consists of up to 6 alphanumeric characters. The first character must be alphabetic.

Index size

Size of the I file in number of index blocks (only significant for EDM).

Number of indexes

The number of index descriptors that follow for this data file. This is the number of indexes specified for the file when it is opened by the application. Value from 1 to 4 for SDM, from 1 to 10 for EDM.
This number may be less than the number of indexes that actually exist for the data file, if not all of these indexes are relevant for the task that has opened the file.

NOTE: In that case, if the records are updated and any keys are changed of the indexes that are not opened, file consistency is lost.

The following fields must be repeated for every index of the file structure. The index specified first is the primary index.

Internal index identification

A value generated by EDM to identify the part of the I file containing the index records for the key defined by the key description that follows. This parameter is used by the Read and the Posit instruction.

Index type

0 = no duplicate keys allowed
1 = duplicate keys allowed
For the prime key this field must be zero.

This parameter is only significant for EDM. In SDM, duplicate keys are always allowed for alternate keys.

Conditional index

0 = no conditional index
1 = conditional index

For SDM this parameter must be = 0

Conditional Index Descriptor (only EDM)

If conditional index is specified, the following parameters are significant. If not, these fields must contain zeroes.

Condition
0 = Equal, the key is included in the index file if the value of the conditional item in the data record equals the Conditional Item Value specified.
1 = Unequal, the key is included in the index file if the value of the conditional item on the data record is not the same as the Conditional Item Value specified.

Conditional item displacement
The character position of the conditional item within the data record. The first character position of the data record is counted as zero.

Conditional item value
Value with which the conditional item on the data record must be compared.

Symbolic Key Descriptor

Number of key-items
Number of items that make up the key. Value may be from 1 to 16.

There is one key-item descriptor for every key item. Note that for an indexed files of S-type the symbolic key can not consist of more than one key item.

Key-item displacement
Position of the key item concerned within the data record, expressed as character position. The first position is counted as zero.

Key-item length
The length in bytes of the key item concerned. The sum of the lengths of the key items must not exceed 64 characters.

5.5 Layout of the File Parameter Block

	Byte	
	01	reserved
	09	record length
	11	blocking factor
	12	file organization
	13	reserved, must contain 1
	14	I/O option
	15	reserved
	17	file name
	25	logging (only EDM)
	26	growth factor
	27	data volume name 1
	33	file size 1
	37	data volume name 2
	43	file size 2
	47	data volume name 3
	53	file size 3
	57	data volume name 4
	63	file size 4
From here, only for indexed files	67	index volume name
	73	index file size, for EDM
	77	number of indexes
	78	reserved

Index descriptor 1	79	reserved
	80	index type
	81	conditional index (only EDM)
	82	expression
	83	conditional item displacement
	85	conditional item value
	86	number of key items
Key descriptor	87	key displacement
	89	key length
	90	reserved

For L- and X-files, bytes 67 - 88 contain:

67	file record number (LRN)
71	number of users
72	reserved
73	sharability
74	reserved
75	reserved
76	reserved
77	reserved
79	creation date
85	retention period
88	reserved

Chapter 6

DISK FILE HANDLING INSTRUCTIONS

6.1 INTRODUCTION

The CREDIT instruction set for disk file handling is described in this chapter. The same instructions are used for all packages. The description is based on the instruction set supported by SDM; instructions, options and features not supported by ADM are marked with "not in ADM" and instructions, options and features only supported by EDM are marked with "only for EDM".

Each section starts with a functional description of the instruction, then the parameters are discussed and one or more examples are given at the end.

Statements listed at the foot of the pages should be used for reference to the CREDIT Reference Manual (module M4A), where syntax and further information on the statements can be found. The TOSS release 12.0 version of the CREDIT Reference Manual should be used.

6.1.1 Survey of Disk File Handling Instructions

File handling instructions:

OPEN .DOUT	Create a new file and open for direct output
OPEN .EXT	Open and Extend an existing standard file for sequential output
OPEN .IN	Open an existing file for input only
OPEN .INOUT	Open an existing file for input and output
OPEN .SOUT	Create a new file and open for sequential output
CLOSE	Close file
CLOSE .DROP	Close and delete file
POSIT .DIR	Set Current Record Number on specified record (not ADM)
POSIT .IXDIR	Set Current Record Number on record with specified key (not ADM)
DSC X'19'	Read File Parameters

Record handling instructions

READ .DIR	Read record with specified relative key
READ .SEQ	Read next record using the relative key (not ADM)
READ .IXDIR	Read record with specified symbolic key (not ADM)
READ .IXSEQ	Read record with next symbolic key (not ADM)
WRITE .DIR	Write record with specified relative key
WRITE .SEQ	Write next record using the relative key
WRITE .IXDIR	Write record with specified symbolic key (not ADM)
WRITE .IXSEQ	Write record with next symbolic key (only EDM)
REWRITE .CUR	Rewrite current record (not ADM)
REWRITE .DIR	Rewrite record with specified relative key
REWRITE .IXDIR	Rewrite record with specified symbolic key (not ADM)
DISCARD .CUR	Delete current record (not ADM)
DISCARD .DIR	Delete record with specified relative key
DISCARD .IXDIR	Delete record with specified symbolic key (not ADM)

Transaction Control Instructions (not ADM)

COMMIT	Release the records accessed during the current transaction (dummy instruction for ADM)
COMMIT .PROT	Release the records accessed during the current transaction except those on the specified files (only EDM)
COMMIT .REL	Release the records accessed during the current transaction, on the specified files only
ROLLBCK	Rollback the current transaction (only EDM)

6.2 OPEN FILE

The Open instruction links a data file and any associated indexes to a data set identifier. Open file can be used to open an existing file or to create new files, both indexed (not for ADM) and non-indexed. An existing file or file structure can be opened for input only (read only) or for input and output.

In EDM, an implicit Commit is executed after a successful Open.

Operands to the instruction define:

- the File Parameter block
- the Open mode
- the Sharability

The No Wait option is not allowed for the Open instruction.

File Parameter Block

The application must supply information about the file and any associated index files to be opened, in the File Parameter block.

Chapter 5 contains the layout of a File Parameter block required for opening files. Bytes 1 to 66 are required for all files. Bytes 67 to 90 are required for all indexed files. Bytes 87 to 90 must be repeated for each index item of a concatenated key (only for EDM), and bytes 79 to 90 must be repeated for every index defined for the file.

The parameters required for each Open mode are found in the diagram below. The items that need not be supplied must be filled with binary zeroes, and data management will set them to the proper value if they are relevant for the file, after opening the file.

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

File Parameters required when opening a file.

Parameter	Open Mode				
	.IN	.SOUT	.DOUT	.EXT	.INOUT
Record Length		x	x		
Blocking Factor		x	x		
File Organization	x	x	x	x	x
Device Type		x	x		
I/O Option	x	x	x	x	x
File Name	x	x	x	x	x
Logging Type	x	x	x	x	x
Growth Factor		x	x	x	x
Data Volume Name(s)	x	x	x	x	x
File Section Size(s)		x	x		

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

Additional parameters for indexed files

Parameter	Open Mode				
	.IN	.SOUT	.DOUT	.INOUT	
Index Volume Name	x	x	x	x	
Index Size		x	x		only EDM
Number of Indexes	x	x	x	x	
Internal Index Id					
Index Type	x	x	x	x	
Conditional Index	x	x	x	x	
Expression Value	x	x	x	x	
Conditional Item Displacement	x	x	x	x	only EDM
Conditional Item Value	x	x	x	x	
Number of Key Items	x	x	x	x	
Key Item Displacement	x	x	x	x	
Key Item Length	x	x	x	x	

The numeric items in the File Parameter block are binary values, the alphanumeric fields such as file name, must contain ISO-7 characters.

Since the File Parameter block must be on a word boundary, it is advisable to place it before any BCD or STRG declarations in the workblock.

Opening an existing indexed file under SDM

When an indexed file structure is opened under SDM, only the indexes specified in the File Parameter block are opened. For each index that is opened, a currency buffer is reserved and the master index is read into memory.

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

When an indexed file is opened with Open mode Input under SDM, only the indexes that are needed for record access need to be specified. It is also possible to set the Number of Indexes and all other index parameters except Index Volume Name, to zero. In that case all the indexes of the file will be opened.

When an indexed file structure is opened for input/output under SDM, all the indexes of the file should be opened. If not, the indexes that have not been opened will not be updated when the data file is updated and then the files will be inconsistent.

Opening an existing indexed file under EDM

When an indexed file structure is opened under EDM, the I file is opened and all indexes are updated when the data file is updated. Only the indexes needed for record access (either for input only or input/output) need to be specified. The indexes may be specified in any order.

When opening and indexed file under EDM it is also possible to set the number of indexes and all the index parameters except for the Index Volume Name to zero in the File Parameter block. The index descriptor block from the I file is then read into the File Parameter block. The File Parameter block length specified in the instruction determines the number of indexes that are opened. The indexes are opened in the order in which they are defined in the index descriptor block.

Open Mode

The Open mode specifies the type of access for which the file is opened. The instructions Close File and Read File Parameters (DSC) are allowed for every Open mode.

There are five Open modes:

.IN Input. Records can be read from the file but not written to it. The only record handling instructions allowed are all types of READ and POSIT.

.INOUT Input/Output. Records can be read from and written to the file. All record handling instructions are allowed.

.SOUT Sequential Output. A new standard file is created and records can be written to the file sequentially. The only record handling instruction allowed is WRITE .SEQ.

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

The new file is not formatted at the Open instruction. When the file is closed, the part after the LRN will be formatted automatically.

In EDM only, Open .SOUT can also be used for the creation of a new indexed file. The instruction WRITE .IXSEQ is then allowed. An E file created with Open .SOUT will be formatted by the Open instruction.

.DOUT Direct Output. A new file is created and formatted, and records can be written to the file directly, via the relative key, and for indexed files, via the prime key. The only record handling instructions allowed are WRITE .DIR and WRITE .IXDIR. For Standard files, WRITE .SEQ is also allowed.

.EXT Extend. Records can be added sequentially to an existing standard file. The only record handling instruction allowed is WRITE .SEQ. The added file extent is not formatted. When the file is closed, the part after the LRN will be formatted automatically.

Open modes .IN, .INOUT and .EXT are only allowed for existing files. Open modes .SOUT and .DOUT can only be used for creation of new files.

In SDM and ADM, a second task opening the same file must specify the same Open mode as the first task that opened the file. In EDM the Open mode need not be the same.

Sharability

The Sharability specifies the protection required for the opened file. There are three types of Sharability:

.NPROT Unprotected. The file can be opened by all tasks, and no records are protected. This is only allowed with Open mode .IN.

.PROT Protected. The file can be opened by all tasks, but any accessed records are held under exclusive access for the requesting task until a Commit, Rollback (EDM only) or Close instruction is executed. This is only allowed with OPEN mode .IN or .INOUT. Under EDM, when a file is opened with Sharability PROT, instructions with the No Wait option (see M21A) are not allowed for the file.

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

.EXCL Exclusive. The complete file is held under exclusive access for the task that issues the Open instruction, and no other task can open it. This Sharability is allowed for all Open modes and it must be used with Open mode .SOUT, .DOUT or .EXT, or if the file is to be deleted with a Close .DROP instruction.

In SDM and ADM, a second task opening the same file must specify the same Sharability as the first task that opened the file. If the Sharability is .EXCL, a second task can not open the same file.

In EDM, when a file is opened protected (.PROT), and transaction logging is not done, file consistency may be lost in the case of an automatic rollback.

In EDM, the Sharabilities need not be the same. If the Sharabilities specified are not in conflict the file will be opened and also be available to the second task. The combinations of Sharabilities allowed in EDM are shown in the diagram below.

task 1 task 2	.NPROT	.PROT	.EXCL
.NPROT	yes	yes	no
.PROT	yes	yes	no
.EXCL	no	no	no

Return Information

Condition Register

After the execution of the Open instruction, the Condition Register will be set to one of the following values:

CR Value	Meaning
0	File open successful
2	Error

OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT

More information may be found in the Status Word and the Return Status. The Status Word is obtained with the XSTAT instruction and the Return Status with the RSTAT instruction. All possible values are found in Chapter 10.

Currency

After a successful Open instruction the CRN and the currency for the indexes will be set to zero so that the first data record is accessed by the first Read Sequential instruction and the record associated with the first index entry by a Read Indexed Sequential instruction.

Corrupt EDM File

In EDM the first byte of the I file is a status byte, indicating if the files are in a consistent state (status byte =0) or corrupt (status byte =1). If at an Open it appears that the files are corrupt, the Open is unsuccessful. In that case the value of the Condition Register will be zero, the Status Word has bit 8 set, the value of the Return Status = 2 (I/O error) and the value of the Supplementary Return Status = 254 (File Corrupt). If this occurs it is best to run the recovery and restart the system.

Example of the OPEN Instruction

```
OPEN      DSDK1,.INOUT,.PROT,PBLOK,LEN
```

DSDK1 This is the data set identifier for the disk file.

.INOUT Open mode. An Open mode of .INOUT permits execution of all data management instructions.

.PROT Sharability .PROT causes all records accessed to be held under exclusive access for the task, but allows other tasks to access other records in the file.

PBLOK This is the name of the string data item containing the File Parameter block.

LEN This is a binary data item containing the used length of the File Parameter block. In the above declaration this data item holds the value 66.

```
-------------------
|    OPEN .IN      |
|    OPEN .INOUT   |
|    OPEN .DOUT    |
|    OPEN .EXT     |
|    OPEN .SOUT    |
-------------------
```

Example of a File Parameter Block

The File Parameter block in this example has been defined as a dummy structure in the Data Division.

```
CB1       STRUC
DUMM      STRG      22X'0'
FILEORG   STRG      2X'0'
DUMO      STRG      2X'0'
RWOPTN    STRG      2X'01'
DUM1      STRG      4X'0'
FILENAM   STRG      8C'ACCDEV  '
LOGG      STRG      2X'03'
GRWTH     STRG      2X'10'
VOL1      STRG      6C'DEVMT1'
DUM3      STRG      8X'0'
VOL2      STRG      6C'  '
DUM4      STRG      8X'0'
VOL3      STRG      6C'  '
DUM5      STRG      8X'0'
VOL4      STRG      6C'  '
DUM6      STRG      8X'0'
          STRUCE    CB1
*
DB1       DSTRUC    CB1
PBLOK     STRG      66
          STRUCE    DB1
```

```
OPEN .IN
OPEN .INOUT
OPEN .DOUT
OPEN .EXT
OPEN .SOUT
```

6.3 CLOSE FILE

The Close file instruction is issued by a task to indicate that it no longer requires access to the file. It is also possible to delete a file with the CLOSE instruction.

The Close instruction causes the following events to occur:

- EDM and SDM execute an implicit Commit. Any records held under exclusive access when the instruction is executed are released.
- If the file has been opened exclusive, The exclusive access is now released and the file may be opened by other tasks.
- The block buffer is written to disk unless the same block is currently accessed by another task.
- If no other task currently has this file open, the LRN or, for EDM files, the start of the free record chain, is updated in the VTOC.
- If a standard file has been created or extended the part after the LRN will be formatted.

The No Wait option is not allowed for the Close instruction.

For a file opened with Delay option (see section 8.7.2), only after a successful Close instruction is it certain that all updates have been written to the files on disk.

Close and Delete File

An operand to the instruction may be used to specify that the file must be deleted after execution of the Close.

.DROP Delete file. The file will be deleted after the Close. The VTOC records of the data file and the index file if any, are deleted from the volumes where they reside. The files can no longer be accessed. Deletion is only allowed for files opened with Sharability .EXCL.

Return Information

The Condition Register will be set to one of the following values as a result of the execution of this instruction.

CR Value	Meaning
0	File successfully closed
2	Error

More information may be found in the Status Word and the Return Status. The Status Word is obtained with the XSTAT instruction and the Return Status with the RSTAT instruction. All possible values are found in Chapter 10.

```
CLOSE
CLOSE .DROP
```

If Close .DROP is issued for a file that is not under exclusive access, a normal Close is executed. The Condition Register will be zero and bit 8 is set in the Status Word. The Return Status value will be 6 "Illegal Close option".

When errors occur during execution of a Close, no recovery can be done. EDM will as much as possible leave files and Monitor table in a consistent state. In EDM, the status byte of an I file involved may be set to 1, "file corrupt".

Examples of the Close Instruction

Close File

```
         CLOSE    DSDK1
```

DSDK1 This is the data set identifier for a disk file. The file currently open on this data set identifier will be closed.

Close and Delete File

```
         CLOSE    .DROP,TEMPFL
```

.DROP This is the keyword indicating that the file must be deleted.

TEMPFL This is the data set identifier for a disk file. The file currently open on this data set identifier will be closed and deleted.

```
 ------------------
|   CLOSE          |
|   CLOSE .DROP    |
 ------------------
```

6.4 SET CURRENT RECORD NUMBER (not for ADM)

The POSIT instruction is used to set the currency for the data file or index file to such a value that a subsequent Read Sequential or Read Indexed Sequential instruction accesses the record specified.

The value of the CRN after a POSIT instruction will be one less than the relative key of the record specified, subject to the rules mentioned below, because the CRN is incremented prior to a sequential file access.

The POSIT instruction is allowed for files opened successfully for Input or Input-Output.

The record may be specified by the relative key (POSIT Direct) or by (part of) a symbolic key (POSIT Indexed).

In SDM, the CRN set by a POSIT instruction can only be used for Read (indexed) Sequential instructions. In EDM, the currency set by POSIT can also be used for Rewrite and Discard Current.

If the file was opened with Sharability .PROT, the record specified will be held under exclusive access for the task. If the record is already under protected access, the instruction is not successful.

Under EDM, if the file was opened with Sharability .PROT, the No Wait option is not allowed for this instruction.

In SDM and in EDM if no transaction logging is done, it is recommended to use the POSIT instruction to set the currencies for the files at the start of every transaction, after a Commit instruction. After a programmed (in EDM) or automatic Rollback during the transaction, the application may then go back to this point and restore the currencies of the files for reprocessing the transaction.

Operands to the instruction define:

- the access type
- the POSIT type
- the relative record key or the symbolic key

Access Type

There are two possible access types for the POSIT instruction:

.DIR Direct. The record is specified by the relative key. After the POSIT, the record required can be accessed with a READ .SEQ instruction.

.IXDIR Indexed Direct. The record is specified by a symbolic key or the part of a symbolic key. After the POSIT, the record required can be accessed with a READ .IXSEQ instruction.

```
-------------------
| POSIT .DIR      |
| POSIT .IXDIR    |
-------------------
```

The part of the key to be checked can be indicated by the application by specifying a keylength different from the actual keylength. The remainder of the key will not be checked and Positioning is on the first record with a key starting with the specified characters. This makes it possible to position at the start of a series of keys. In EDM, the keylength is the sum of the lengths of the key items.

POSIT Type

There are three types of positioning, each indicated by a value:

0 - Equal: The CRN is set to the record pointed to by the relative key specified. The record may be "used" or "free".

The index currency is set to the record identified by the specified symbolic key (or part of it) according to the index specified for the instruction. The record is always "used". If the record is not found, the error message "Key not found" is returned.

1 - Greater: The CRN is set to the next "used" record pointed to by the specified relative key.

The index currency is set to the record identified by the symbolic key (or part of it) next to the key specified, according to the index specified for the instruction.

2 - Not-less: The CRN is set to the record pointed to by the specified relative key if this record is "used", or to the next "used" record if this one is "free".

The index currency is set to the record identified by the symbolic key (or part of it) specified if it exists, or to the record identified by the next symbolic key according to the index specified for the instruction.

Examples of POSIT Types

Fig 6 - 1 is a stylised part of a file layout, with examples showing the effect of the POSIT type on the resulting value of the CRN for this file.

```
 -----------------
| POSIT .DIR      |
| POSIT .IXDIR    |
 -----------------
```

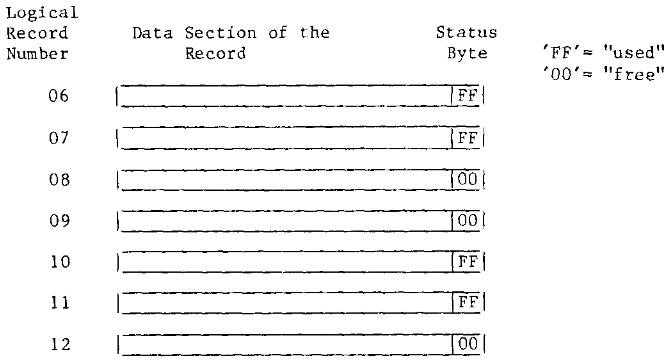


Fig 6-1 Example of POSIT Types

POSIT on Equal (type = 0)

```
TYPE      EQU       X'0'
          MOVE      RECNO,=X'A'
          POSIT     .DIR,DSDK1,RECNO,TYPE
```

The CRN will be set to 9 and the next record to be accessed will be record 10.

POSIT on Greater (type = 1)

```
TYPE      EQU       X'1'
          MOVE      RECNO,=X'7'
          POSIT     .DIR,DSDK1,RECNO,TYPE
```

The CRN will be set to 9 and the next record to be accessed will be record 10. For POSIT on Greater, the search for a used record starts at the record following that specified, so in this case the search for a used record will start at record 8.

POSIT on Not-less (type = 2)

```
TYPE      EQU       X'2'
          MOVE      RECNO,=X'A'
          POSIT     .DIR,DSDK1,RECNO,TYPE
```

The CRN is set to 9 and the next record to be accessed will be record 10. For POSIT on Not-less, the search for a used record starts at record 10.

POSIT .DIR
POSIT .IXDIR

Return Information

The Condition Register will be set to one of the following values as a result of the execution of this instruction.

CR Value	Meaning
0	Operation successful
1	End of File reached
2	Error
3	Attempt to use record number greater than number of records in file

The value of the currency (CRN) is not changed after an unsuccessful POSIT instruction. After an unsuccessful POSIT .DIR on a standard file, however, the Control Word will contain the relative key specified in the instruction.

After a POSIT .IXDIR with Posti type 0 (equal), the CR is set to 2 and bit 5 and 0 are set in the Status Word if the specified key is not found.

Examples of the POSIT instruction

POSIT .DIR

```
TYPE    EQU     X'1'
        POSIT   .DIR,DSDK1,RECNO,TYPE
```

DSDK1 This is the data set identifier for a disk file.

RECNO This is a decimal data item containing the relative key which will be used to set the CRN pointer.

TYPE This is a value expression indicating the POSIT type required: 0 = equal, 1 = greater, 2 = not-less.

POSIT .IXDIR

```
TYPE    EQU     X'2'
        MOVE    STRG,=C'                    BAAAAAAA'
        MOVE    LEN,='8'
        MOVE    IXID,='3'
        POSIT   .IXDIR,DSDK1,STRG,LEN,TYPE,IXID
```

DSDK1 This is the data set identifier for a disk file.

STRG This is a string data item containing the key to be used to position the currency of the index file, at the displacement defined for the key in the File Parameter block.

LEN This is a binary data item defining the length of the key which is to be searched.

TYPE This is a value expression indicating the POSIT type required: 2 = not-less.

IXID This is a binary data item or a literal constant specifying the index to be used.

The currency for the index file I3 will be set to such a value that the record with the key "BAAAAAAA" if it exists, else the record with the next higher key value, will be accessed with a READ .IXSEQ instruction.

6.5 READ FILE PARAMETERS

The File Parameters and the Current Record Number (CRN) are obtained by the application with a Data Set Control (DSC) instruction. The File Parameter block is the block specified in the OPEN instruction. Some of the reserved fields will have had values inserted into them by the data management.

The layout of the File Parameter block is found in Chapter 5.

The complete index descriptor is read from disk so that the index descriptors returned are those specified when the file was created, and in that order, independent of the indexes specified when the file was opened.

The instruction is only accepted after a successful Open of the file concerned. The instruction is allowed for each Open mode and Sharability. The No Wait option is not allowed.

Operands to the instruction define:

- The DSC control code. For Read File Parameters, this must be X'19'.
- The data item (BCD) where the CRN must be returned.
- The buffer where the File Parameters must be returned.
- The requested length. It is not necessary to read the complete File Parameter block. The file parameters will be copied into the buffer up to the specified length.

Return Information

The Condition Register will be set to one of the following values as a result of the execution of this instruction.

CR Value	Meaning
0	I/O successful
2	Error

The CRN is not affected.

Example of the DSC X'19' Instruction

An example of the DSC instruction used to obtain the File Parameters is shown below:

```
FILEP   EQU     X'19'
        DSC     DSDK1,FILEP,COUNT,STRG,LEN
```

DSDK1 This is the data set identifier for a disk file.

FILEP This is an equate identifier containing the control code. X'19' is the control code for reading the file parameters.

COUNT This is a data item (BCD), which after execution of the instruction, will contain the Current Record Number (CRN).

STRG This is a string data item into which the file parameters will be copied.

LEN This is a binary data item, defining the length of the buffer. After execution of the DSC instruction, LEN will contain the number of bytes actually transferred. The number of bytes transferred never exceeds the specified buffer length.

For example, if the File Parameter of a sequential file without indexes is read and LEN is set to 100, LEN will have the value 66 after the instruction. On the other hand, if LEN is set to 36, only the first 36 bytes of the File Parameter block will be transferred to the buffer.

6.6 READ RECORD

The READ instruction is used to read records into the application buffer. Records can be read from a file opened with Open mode .IN or .INOUT. The record read will become the current record for the requesting task.

If the file was opened with Sharability .PROT, the record read will be held under exclusive access for the requesting task until the task releases it with a COMMIT, ROLLBCK (only in EDM) or CLOSE.

Characters are read from the record into the string buffer specified in the instruction until the number of characters given in the length operand have been transferred or the end of the record is reached. If the number of characters specified in the length operand is less than the record length, the Condition Register will be set to 2 and bit 12 (Illegal length) is set in the Status Word.

Execution of the task issuing the READ will be suspended until the instruction is completed, unless the No Wait option is specified in the instruction. Under EDM, No Wait is not allowed for files opened with Sharability .PROT.

Operands to the instruction define:

- Access type
- Application buffer
- Requested Length

Access Type

The access type may be

.SEQ Sequential. This access type is allowed for non-indexed and indexed files opened with Open mode .IN or .INOUT. The "used" record following the current record according to the relative key will be read. Records with status "free" are ignored. The CRN is incremented before the record is accessed. When a file is opened, the CRN is preset to zero so the first READ .SEQ instruction will read the first record in the file.

End of File is returned when the last record of the file written by Write Sequential instructions, has been read.

.DIR Direct. This access type is allowed for standard and indexed files opened with Open mode .IN or .INOUT. The record to be read is identified by the relative key, specified by the application.

```
 -----------------
|  READ .SEQ      |
|  READ .DIR      |
|  READ .IXSEQ    |
|  READ .IXDIR    |
 -----------------
```

If the relative key points to a record with status "free", the Condition Register will be 2 and bit 0 and 4, "No Data", is set in the Status word. If the relative key points to a record outside the physical data file, the Condition Register is set to 3 and bit 2, "End of Medium", is set in the Status Word.

.IXSEQ Indexed Sequential. This access type is only allowed for indexed files opened with Open mode .IN or .INOUT. Records are read in the sequence in which the symbolic keys occur in the index specified. If the end of the index has been reached, the condition Register is set to 1 and bit 3, "End of File", is set in the Status Word.

If the current record has the same key as the next record, the Condition Register will be zero and bit 6, "Duplicate Key", is set in the Status Word. This is not an error message but an indication for the application.

When an indexed data file is opened, the currency for all indexes is set to zero so that the first READ .IXSEQ instruction will access the record associated with the first index entry in the index specified.

SDM only maintains one index currency at the time. This is the most recently used index. The index currency of the other indexes is set to zero.

EDM maintains the index currency for all indexes opened for the file.

.IXDIR Indexed Direct. This access type is only allowed for indexed files opened with Open mode .IN or .INOUT. The record to be read is identified by a symbolic key. The key must be supplied by the application, in the application record buffer at the displacement defined in the index descriptor for the index used.

If the key specified is a key for which duplicates exist in the file, the record associated with the first index entry with the same key, is read. The other records with that key can then be accessed by READ .IXSEQ instructions. ("Duplicate Key" is not returned after READ .IXDIR.)

If the requested symbolic key is not present in the specified index, the Condition Register is set to 2 and bit 5 "Key not Found" is set in the Status Word.

READ .SEQ
READ .DIR
READ .IXSEQ
READ .IXDIR

Return Information

After completion of the READ instruction, the following information is returned:

Condition Register

The Condition Register will be set to one of the following values as a result of the execution of a READ instruction:

CR Value	Meaning
0	Read successful
1	End of file reached
2	Error
3	End of device reached

More information can be obtained from the Status Word and the Return Status. All possible values are found in Chapter 10.

Effective Length

The actual number of characters read is returned in the length operand.

The CRN

After successful completion of the READ instruction, the CRN points to the record read, independent of the access type. After Read Indexed Sequential and Read Indexed Direct instructions, the currency of the index file is set to such a value that the record associated with the next index entry is accessed with the next READ .IXSEQ instruction.

If the READ is not successful, the value of the CRN is the same as before execution of the instruction, with the following exception:

After I/O errors during a READ .SEQ on a standard file, the CRN will point to the record that would have been read if the instruction had been successful. The next READ .SEQ will than access the next record. In this way it is possible to pass badspots in a standard file.

The Control Word

After successful completion of the READ instruction, the relative key of the current record is returned in the Control Word. The Control Word can be obtained by the application with a GETCW instruction.

If the READ is not successful, the value of the Control Word is the same as before execution of the instruction, with the following exception:

```
READ .SEQ
READ .DIR
READ .IXSEQ
READ .IXDIR
```

After I/O errors during a READ .SEQ or a READ .DIR on a standard file, the Control Word will contain the relative key of the record that would have been read if the instruction had been successful.

Examples of the READ Instruction

Read Sequential (READ .SEQ)

READ .SEQ,DSDK1,STRG,LEN

.SEQ Sequential. The CRN will be incremented and the first "used" record following the current record for data set DSDK1 will be read.

DSDK1 This is the data set identifier of the disk file.

STRG This is a string data item into which characters will be read. It should be large enough to hold one record.

LEN This is a binary data item containing the number of characters to be read (the record length). After execution it will contain the actual number of characters read.

Read Direct (READ .DIR)

READ .DIR,DSDK1,STRG,LEN,RECNO

.DIR Direct. The record with the relative key value specified in the operand RECNO will be read and the CRN pointer will be updated.

DSDK1 This is the data set identifier for a disk file.

STRG This is a string data item into which the characters will be read. It should be large enough to hold one record.

LEN This is a binary data item containing the number of characters to be read (the record length). After execution this will contain the actual number of characters read.

RECNO This is a decimal data item containing the relative key value of the record to be read.

READ .SEQ
READ .DIR
READ .IXSEQ
READ .IXDIR

Read Indexed Direct (READ .IXDIR)

```
READ      .IXDIR,DSDK1,STRG,LEN,IXID
```

.IXDIR Indexed direct access. The record to be read is specified by a symbolic key.

DSDK1 This is the data set identifier for a disk file.

STRG This is a string data item into which the record will be read. Before the instruction is executed, it must contain the symbolic key of the record to be accessed at the displacement defined for the key in this index.

LEN This is a binary data item containing the number of characters to be read (the record length). After execution it will contain the actual number of characters read.

IXID This is a binary data item or literal constant specifying the internal index identifier (index number) of the index to be used.

Read Indexed Sequential (READ .IXSEQ)

```
READ      .IXSEQ,DSDK1,STRG,LEN,IXID
```

.IXSEQ Indexed Sequential. The record to be read is the next in sequence in the specified index.

DSDK1 This is the data set identifier for a disk file.

STRG This is a string data item into which characters will be read. It should be large enough to hold one record.

LEN This is a binary data item containing the number of characters to be read (the record length). After execution this will contain the actual number of characters read.

.IXID This is a binary data item or literal constant specifying the index to be used.

READ .SEQ
READ .DIR
READ .IXSEQ
READ .IXDIR

6.7 WRITE RECORD

The WRITE instruction is used to write new records from the application buffer to the file. Records can be written to files opened with Open mode .DOUT, .SOUT, .EXT or .INOUT. The record written will become the current record for the requesting task after Write Sequential instructions on files opened with Open mode .SOUT or .EXT.

New records can only be written to records with the status "free". If the WRITE is successful, the record status will be set to "used".

The new record is written from the application record buffer. The buffer should be long enough to hold one record. The characters are transferred sequentially until the end of the buffer is reached. If the buffer is shorter than the number of characters defined in the record length, the Condition Register is set to 2 and bit 12, "Incorrect Length", is set in the Status Word.

If the file is indexed, the new record is identified by the prime key. The new data record is written to the first free position after the LRN (for SDM) or to the first free record in the file (for EDM) and the index entries are inserted in the index files in the correct places. Under SDM, all index files should have been opened. Master indexes are not updated. Under EDM, all indexes are updated.

If the file was opened with Sharability .PROT, the record written will be held under exclusive access for the requesting task until the task releases it with a COMMIT, ROLLBCK (only in EDM) or CLOSE.

Execution of the task issuing the WRITE will be suspended until the instruction is completed, unless the No Wait option is specified in the instruction. Under EDM, No Wait is not allowed for files opened with Sharability .PROT.

Operands to the instruction define:

- Access type
- Application buffer

Access Type

The access type may be

.SEQ Sequential. WRITE SEQ is allowed for standard files opened with Open mode .SOUT, .DOUT, .EXT or .INOUT.

```
WRITE .SEQ
WRITE .DIR
WRITE .IXSEQ
WRITE .IXDIR
```

The record will be written to the first "free" record following the record pointed to by the Last Record Number pointer (LRN). After a successful WRITE .SEQ the LRN is incremented. The CRN is only incremented if the Open mode was .SOUT or .EXT. When a new file is created, the LRN is set to zero so the first WRITE .SEQ instruction will write the first record in the file.

If the end of the physical file space has been reached and the file is not automatically enlarged, the Condition Register will be set to 1 and bit 3, "End of File", is set in the Status Word.

.DIR Direct. WRITE .DIR is allowed for standard files opened with Open mode .DOUT or .INOUT. The record to be written is identified by the relative key, specified by the application. The currency is not affected by Write Direct instructions.

If the relative key points to a record with status "used", the Condition Register is set to 2 and bit 9, "Duplicate Error", is returned in the Status Word.

If the relative key points to a record outside the physical data file, the Condition Register is set to 3 and bit 2, "End of Medium" is set in the Status Word together with bit 0.

.IXDIR Indexed Direct. WRITE .IXDIR is allowed for indexed files opened with Open mode .DOUT or .INOUT. The prime key must be supplied by the application, in the application record buffer at the displacement defined in the index descriptor. The currency is not affected by Write Indexed Direct instructions.

If the prime key exists already in the primary index file, the Write is not executed. The Condition Register is set to 2 and bit 9, "Duplicate Key", is set in the Status Word.

.IXSEQ Indexed Sequential (only EDM). WRITE .IXSEQ is only allowed for indexed files opened with Open mode .SOUT. Records must be supplied by the application in sequence of the prime key. All indexes defined for the file are updated. The currency is not affected by Write Indexed Direct instructions.

If the prime key of the new record is not greater than the prime key of the last record written, the Write is not executed. The Condition Register is set to 2 and bit 5 "Invalid Key", is set in the Status Word.

```
 -------------------
|   WRITE .SEQ      |
|   WRITE .DIR      |
|   WRITE .IXSEQ    |
|   WRITE .IXDIR    |
 -------------------
```

Return Information

After completion of the WRITE instruction, the following information is returned:

Condition Register

The Condition Register will be set to one of the following values as a result of the execution of a WRITE instruction:

CR Value	Meaning
0	Read successful
1	End of file reached
2	Error
3	End of device reached

More information can be obtained from the Status Word and the Return Status. All possible values are found in Chapter 10.

The CRN

The CRN will be set to the record just written for a standard file opened for Output Sequential or Extend. For other file types and files opened in other Open modes, the CRN and the index currency are not affected.

If the WRITE is not successful, the value of the CRN is the same as before execution of the instruction.

The Control Word

After successful completion of the WRITE instruction, the relative key of the record written is returned in the Control Word. The control Word can be obtained by the application with a GETCW instruction.

If the WRITE is not successful, the value of the Control Word is the same as before execution of the instruction.

Examples of the WRITE Instruction

Write Sequential (WRITE .SEQ)

```
WRITE    .SEQ,DSDK1,STRG
```

.SEQ — Sequential. The record is written to the position after that indicated by the current contents of the LRN. The LRN pointer held in memory will be incremented.

DSDK1 — This is the data set identifier of the disk file.

STRG — This is a string data item from which characters are written. It should be large enough to hold one record.

Write Direct (WRITE .DIR)

```
WRITE    .DIR,DSDK1,STRG,RECNO
```

.DIR — Direct. The record is written to the position indicated by the relative key supplied by the application.

DSDK1 — This is the data set identifier of the disk file.

STRG — This is a string data item from which characters are written. It should be large enough to hold one record.

RECNO — This is a decimal data item (BCD) containing the relative key of the record to be written.

Write Indexed Direct (WRITE .IXDIR)

```
WRITE    .IXDIR,DSDK1,STRG
```

.IXDIR — Indexed Direct. The contents of STRG is written to the file opened on data set identifier DSDK1, and the corresponding index entries are inserted in the associated index files. In SDM, the record is written to the first free record after the LRN. In EDM, the record is written to the first free record in the free record chain of the data file.

DSDK1 — This is the data set identifier for a disk file.

STRG — This is a string data item from which characters will be written. It should be large enough to hold one record. The string data item must contain the prime record key at the displacement defined for the key of the primary index.

```
WRITE .SEQ
WRITE .DIR
WRITE .IXSEQ
WRITE .IXDIR
```

Write Indexed Sequential (WRITE .IXSEQ)

```
WRITE      .IXSEQ,DSDK1,STRG
```

.IXSEQ Indexed Sequential. The records must be supplied in sequence of the prime key.

DSDK1 This is the data set identifier for a disk file.

STRG This is a string data item from which characters will be written. It should be large enough to hold one record. The string data item must contain the prime record key at the displacement defined for the key of the primary index.

WRITE .SEQ
WRITE .DIR
WRITE .IXSEQ
WRITE .IXDIR

6.8 REWRITE RECORD

The REWRITE instruction is used to write an updated record from the application buffer to the file. Records can only be rewritten to files opened with Open mode .INOUT. The Rewrite instruction does not affect the currency.

Records can only be rewritten to records with the status "used". Before a record is rewritten to the file it should have been read, but this is not checked by data management.

The record is written from the application record buffer. The buffer should be long enough to hold one record. The characters are transferred sequentially until the end of the buffer is reached. If the buffer is shorter than the number of characters defined in the record length, the Condition Register is set to 2 and bit 12, "Incorrect Length", is set in the Status Word.

Under SDM, if the file is indexed, all record keys must be unchanged.

Under EDM, if the file is indexed, the value of the prime key must not be changed. If alternate keys and conditional keys have been updated, all indexes are updated. New index entries are generated and the old ones removed from the index file.

If the file was opened with Sharability .PROT, the record rewritten will be held under exclusive access for the requesting task until the task releases it with a COMMIT, ROLLBCK (only in EDM) or CLOSE.

Execution of the task issuing the REWRITE will be suspended until the instruction is completed, unless the No Wait option is specified in the instruction. In EDM, No Wait is not allowed for files opened with Sharability .PROT.

Operands to the instruction define:

- Access type
- Application buffer

Access Type

.CUR Current. Rewrite Current is allowed for standard and indexed files opened with Open mode .INOUT. The current record is rewritten.

SDM: After a POSIT instruction, Rewrite Current will rewrite the record last accessed by a Read instruction.

EDM: After a POSIT instruction, Rewrite Current will rewrite the record selected by the POSIT instruction.

```
REWRITE .CUR
REWRITE .DIR
REWRITE .IXDIR
```

.DIR Direct. Rewrite Direct is allowed for standard files opened with Open mode .INOUT. The record to be rewritten is identified by the relative key, specified by the application.

If the relative key points to a record with status "free", the Condition Register is set ot 2 and bit 4, "No Data", is set in the Status Word together with bit 0.

If the relative key points to a record outside the physical data file, the Condition Register is set to 3 and bit 2, "End of Medium" is set in the Status Word together with bit 0.

.IXDIR Indexed Direct. Rewrite Indexed Direct is allowed for indexed files opened with Open mode .INOUT. The prime key of the record must be unchanged.

In SDM, all keys must be unchanged.

In EDM, if alternate keys have been changed, the associated indexes will be updated.

Under EDM, if one of the alternate keys for which duplicates are allowed exist already, the Condition Register will be zero and bit 6 "Duplicate Key" is set in the Status Word.

If one of the alternate keys for which duplicates are not allowed exist already, the condition Register is set to 2 and bit 9 "Duplicate Key not allowed" is set in the Status Word. The Rewrite is not executed.

If the prime key has been changed, the Rewrite is not executed. The Condition Register will be zero and bit 8, "DM rule violated", is set in the Status Word. The Return Status is set to 2 and for EDM the Supplementary Return Status has the value 214.

REWRITE .CUR
REWRITE .DIR
REWRITE .IXDIR

Return Information

After completion of the REWRITE instruction, the following information is returned:

Condition Register

The Condition Register will be set to one of the following values as a result of the execution of a REWRITE instruction:

CR Value	Meaning
0	Read successful
2	Error
3	End of device reached

More information can be obtained from the Status Word and the Return Status. All possible values are found in Chapter 10.

The CRN

The currency is not affected by Rewrite instructions. In EDM however, if the currency had been set by a POSIT instruction, the currency will be set to the record rewritten with a Rewrite Current so that a subsequent Read Sequential will access the record following the record rewritten.

The Control Word

After successful completion of the REWRITE instruction, the relative key of the record rewritten is returned in the Control Word. The Control Word can be obtained by the application with a GETCW instruction.

If the REWRITE is not successful, the value of the Control Word is the same as before execution of the instruction, with the following exception:

After I/O errors during a REWRITE .CUR or a REWRITE .DIR on a standard file, the Control Word will contain the relative key of the record that would have been rewritten if the I/O had been successful

Examples of the REWRITE Instruction

Rewrite Current (REWRITE .CUR)

```
REWRITE    .CUR,DSDK1,STRG
```

.CUR — Current. The contents of the string data item is written back to the record indicated by the current record number. The contents of the CRN are not affected by execution of this instruction.

DSDK1 — This is the data set identifier of the disk file.

STRG — This is a string data item. It contains the characters to be written and must be large enough to hold one record.

Rewrite Direct (REWRITE .DIR)

```
REWRITE  .DIR,DSDK1,STRG,RECNO
```

.DIR — Direct. The contents of the string data item is written back to the record with the specified relative key.

DSDK1 — This is the data set identifier for a disk file.

STRG — This is a string data item from which characters are written. It should be large enough to hold one record.

RECNO — This is a decimal data item containing the relative key of the record to be rewritten.

Rewrite Indexed Direct (REWRITE .IXDIR)

```
REWRITE  .IXDIR,DSDK1,STRG
```

.IXDIR — Indexed Direct. The contents of the string data item is written back to a record in the data file. The record overwritten as a result of this instruction will be the record whose prime key matches the prime key of the record held in the string.

DSDK1 — This is the data set identifier for the disk file.

STRG — This is a string data item from which characters will be written. It should be large enough to hold one record. The string data item must contain the prime record key at the displacement defined for the key of the primary index.

```
 ------------------
| REWRITE .CUR     |
| REWRITE .DIR     |
| REWRITE .IXDIR   |
 ------------------
```

6.9 DISCARD RECORD

The Discard instruction is used to delete a record from the file. Records can only be deleted from files opened with Open mode .INOUT. Only records with status "used" can be deleted. The status will be set to "free".

If the file is indexed, the index files opened for the file are updated under SDM. Master indexes are not updated. All indexes should have been opened. Under EDM, all indexes are updated automatically.

If the file was opened with Sharability .PROT, the record deleted will be held under exclusive access for the requesting task until the task releases it with a COMMIT, ROLLBCK (only in EDM) or CLOSE.

Execution of the task issuing the DISCARD will be suspended until the instruction is completed, unless the No Wait option is specified in the instruction. Under EDM, No Wait is not allowed for files opened with Sharability .PROT.

Operands to the instruction define:

- Access type
- Application buffer

Access Type

The access type may be

.CUR Current. Discard Current is allowed for standard and indexed files opened with Open mode .INOUT. The current record is deleted.

SDM: After a POSIT instruction, Discard Current will delete the record last accessed by a Read instruction.

EDM: After a POSIT instruction, Discard Current will delete the record selected by the POSIT instruction.

.DIR Direct. Discard Direct is allowed for standard files opened with Open mode .INOUT. The record to be rewritten is identified by the relative key, specified by the application.

If the relative key points to a record with status "free", the Condition Register is set to 2 and bit 4, "No Data", is set in the Status Word together with bit 0.

```
DISCARD .CUR
DISCARD .DIR
DISCARD .IXDIR
```

If the relative key points to a record outside the physical data file, the Condition Register is set to 3 and bit 2, "End of Medium" is set in the Status Word together with bit 0.

.IXDIR Indexed Direct. Discard Indexed Direct is allowed for indexed files opened with Open mode .INOUT. The record is identified by the prime key.

Return Information

After completion of the DISCARD instruction, the following information is returned:

Condition Register

The Condition Register will be set to one of the following values as a result of the execution of a DISCARD instruction:

CR Value	Meaning
0	Discard successful
2	Error
3	End of Device

More information can be obtained from the Status Word and the Return Status. All possible values are found in Chapter 10.

The CRN

The CRN is not affected by Discard Direct or Indexed Direct instructions. After a Discard Current, the currency will be such that a subsequent Read Sequential or Read Indexed Sequential instruction will access the next record or the record associated with the next index entry according to the last used index.

The Control Word

After successful completion of the DISCARD instruction, the relative key of the record deleted is returned in the Control Word. The Control Word can be obtained by the application with a GETCW instruction.

If the DISCARD is not successful, the value of the Control Word is the same as before execution of the instruction, with the following exception:

```
DISCARD .CUR
DISCARD .DIR
DISCARD .IXDIR
```

After I/O errors during a DISCARD .CUR or a DISCARD .DIR on a standard file, the Control Word will contain the relative key of the record that would have been rewritten if the I/O had been successful.

Examples of the DISCARD Instruction

Discard Current (DISCARD .CUR)

```
DISCARD    .CUR,DSDK1
```

.CUR — Current. The current record is deleted from the file. If the file is indexed, the associated index entries are also deleted from the index files. After this instruction the currency is such that the next "used" record according to the last used access path is read with a Read (Indexed) Sequential instruction.

DSDK1 — This is the data set identifier of the disk file.

Discard Direct (DISCARD .DIR)

```
DISCARD  .DIR,DSDK1,RECNO
```

.DIR — Direct. The record with the specified relative key is deleted.

DSDK1 — This is the data set identifier for a disk file.

RECNO — This is a decimal data item containing the relative key of the record to be deleted.

Discard Indexed Direct (DISCARD .IXDIR)

```
DISCARD  .IXDIR,DSDK1,STRG
```

.IXDIR — Indexed Direct. The record with a prime key with the same value as the prime key of the record in application buffer is deleted from the file. If the file is indexed, the associated index entries are also deleted from the index files.

DSDK1 — This is the data set identifier for the disk file.

STRG — This is a string data item from which characters will be written. It should be large enough to hold one record. The string data item must contain the prime record key at the displacement defined for the key of the primary index.

```
DISCARD .CUR
DISCARD .DIR
DISCARD .IXDIR
```

6.10 COMMIT

The COMMIT instruction is used to release records held under exclusive access on a file opened with Open mode .PROT, and to initiate the next transaction if transaction and/or function logging is done (only in EDM). Only a certain number of records may be held under exclusive access at any one time. This number is specified during Monitor generation.

ADM does not support the Commit instruction. Commit instructions may be used for compatibility with SDM or EDM and will be treated as dummy instructions.

In SDM and EDM, a COMMIT is executed automatically when a CLOSE instruction is executed.

In EDM only, a COMMIT is executed automatically after a successful OPEN instruction.

If transaction logging is done for the files, the transaction log information of the current transaction is deleted. If function logging is done, the function log buffers are written to the function log file on disk or tape.

NOTE

Commit does not result in any I/O operations to the files on disk. At what time the contents of the internal block buffer is written to the disk is determined by data management, independent of the transaction control functions.

The No Wait option is not supported for the COMMIT instruction.

Operands to the instruction define:

- Commit type
- The data sets to which the Commit type applies
- The data item where to store the return information

Commit Type

As an option, one of the following types may be specified. If no type is specified, all records under exclusive access for the task are released.

.REL Release. Commit with Release is used to release the records held under exclusive access for a task on the specified files only. Protected records from other files remain protected.

```
COMMIT
COMMIT .PROT
COMMIT .REL
```

Commit with Release defines a "sub-transaction" in the application. The transaction log information is deleted, if transaction logging is being done for the files. This means that the application is responsible for file consistency at the point where a COMMIT .REL is issued.

At a subsequent automatic or programmed (only in EDM) Rollback, the records of the files that were not released with the COMMIT .REL instruction will also be released.

.PROT Protect. (only EDM). Commit with protection is used to release the records held under exclusive access for a task except those on the specified files. This defines a "subtransaction" in the application. The transaction log information is deleted, if transaction logging is being done for the files. This means that the application is responsible for file consistency at the point where a COMMIT .PROT is issued.

At a subsequent automatic or programmed (only in EDM) Rollback, the records of the files specified in the COMMIT .PROT instruction will also be released.

Return Information

The Condition Register and the currency are not affected and the Control Word is not used by the COMMIT instruction.

The return information of the instruction is returned in the binary data item supplied in by the application. This may be set to one of the following values:

Value	Meaning
0	Successful completion.
-1	I/O error. Additional information may be found in the Status word. All possible values are listed in Chapter 10.
-2	Data Management rule violated. Additional information may be found in the Return Status and Supplementary Return Status. All possible values are listed in Chapter 10.

The values -1 and -2 can only be obtained when EDM is used and transaction or function logging is done.

```
COMMIT
COMMIT .PROT
COMMIT .REL
```

Examples of the COMMIT Instruction

Commit

```
COMMIT   PARM
```

All records held under exclusive access for the task are released.

PARM This is a binary data item. After execution of this instruction it contains the return information.

Commit with Release (COMMIT .REL)

```
COMMIT.REL,PARM,DSDK1,DSDK3
```

.REL Commit with Release. Exclusive access is released only for records from the files opened on data set identifiers DSDK1 and DSDK3.

PARM This is a binary data item. After execution of this instruction it contains the return information.

Commit with Protection (COMMIT .PROT) (only for EDM)

```
COMMIT.PROT,PARM,DSDK1,DSDK3
```

.PROT Commit with Protection. Exclusive access is released for all files except from those opened on data set identifiers DSDK1 and DSDK3.

PARM This is a binary data item. After execution of this instruction it contains the return information.

COMMIT
COMMIT .PROT
COMMIT .REL

6.11 ROLLBACK (only for EDM)

The ROLLBCK instruction is used to abort the current transaction and to release records held under exclusive access on a file opened with Open mode .PROT.

If transaction logging is done, the before images of the records logged during this transaction will be re-applied to the disk files concerned. The files are then in the consistent state they were in at the previous Commit or Open, i.e. at the beginning of the transaction. If transaction logging is done, the currency for data and index files will also be reset to the value at the previous Commit. After that the transaction log information of the current transaction is deleted.

If function logging is done, the function log buffers are written to the function log file on disk or tape.

NOTE

If the previous Commit was a Commit with Protection or Commit with Release, Rollback will also release the records for the files that were not released. Consistency of these files is the responsibility of the program.

Rollback does not necessarily result in the before images of records being written back to the disk. They are restored to the internal block buffers in EDM.

The No Wait option is not supported for the ROLLBCK instruction.

The operand to the instruction defines:

- The data item where to store the return information

Return Information

After completion of the ROLLBCK instruction, the following information is returned:

Condition Register

The Condition Register is not affected and the Control word is not used by the ROLLBCK instruction.

The return information of the instruction is returned in the binary data item supplied in by the application. This may be set to one of the following values:

Value	Meaning
0	Successful completion.
-1	I/O error. Additional information may be found in the Status word. All possible values are listed in Chapter 10.
-2	Data Management rule violated. Additional information may be found in the Return Status and Supplementary Return Status. All possible values are listed in Chapter 10.

The values -1 and -2 can only be obtained when transaction or function logging is done.

The Currency

Only if transaction logging is done, the CRN and the currencies of the index files are reset to he values they had at the previous COMMIT or OPEN instruction.

If no transaction logging is done, the currencies are not affected.

Examples of the ROLLBCK Instruction

Rollback

```
ROLLBCK   CODE
```

CODE After execution of the Rollback, the return information will be stored here.

6.11.1 Automatic Rollback

If EDM detects a deadlock situation, Rollback is performed automatically for the task whose request caused the deadlock situation. The records held under protected access by this task are released.

NOTE:

If the previous Commit was a Commit with Protection or Commit with Release, Rollback will also release the records for the files that were not released. Consistency of these files is the responsibility of the program.

If transaction logging is done for the files concerned, the before images of the records are written back to the file and the currency is reset to the value it had at the previous Commit.

When this happens, the Condition Register is set to 2 and bit 11, "Sequence Error / Rollback Performed" is set in the Status Word.

Deadlock

A deadlock situation arises if tasks require protected access to the same records and start waiting for each other to release them.

Before a task goes into a wait state to wait for a record, the protected record administration is checked by EDM. If this task then is already holding a record which the other task is waiting for, this is deadlock and the transaction for this task is rolled back. Deadlock situations with more than two tasks involved are also detected.

The situation of two tasks waiting for each other's records can be partially avoided if different tasks all follow the same sequence of accessing records.

In SDM, an automatic Rollback occurs if access is requested to a record held protected by another task. There is no check if this is a real deadlock situation. A COMMIT, releasing the record protection for all records, is then executed for the second task, with the risk of file inconsistency.

For SDM and for EDM if transaction logging is not used, the following application design is recommended to avoid inconsistent files.

All records needed for a transaction are read by the task, before any record is updated. In this way the task has all records involved under exclusive access for the duration of the transaction. If the transaction is rolled back as a result of a Read for a record which is not available, no information on the files will have been updated yet.

Chapter 7

ABRIDGED DATA MANAGEMENT

7.1 INTRODUCTION

This chapter discusses the file types handled by ADM. A list of the file handling instructions supported by ADM and a survey of the Status Word and Return Status values that may be returned by ADM are also included.

7.2 ADM INSTRUCTION SET

The following instructions are supported by ADM:

File Handling Instructions:

OPEN .DOUT	Create a new file and open for direct output
OPEN .EXT	Open and Extend an existing standard file for sequential output
OPEN .IN	Open an existing file for input only
OPEN .INOUT	Open an existing file for input and output
OPEN .SOUT	Create a new standard file and open for sequential output
CLOSE	Close file
CLOSE .DROP	Close and delete file
DSC X'19'	Read File Parameters

Record Handling Instructions

READ .DIR	Read record with specified relative key
WRITE .DIR	Write record with specified relative key
WRITE .SEQ	Write next record to a standard file, using the relative key
REWRITE .DIR	Rewrite record with specified relative key
DISCARD .DIR	Delete record with specified relative key

The status of the records is not checked. The records addressed by Rewrite and Discard instructions are assumed to be "used" and the records addressed by Write instructions are assumed to be "free". A Read Direct instruction can be used to check if the status of a record is "used".

Sharability "Protected" is not supported. Record protection is the responsibility of the application. Sharability "Protected" may be specified for compatibility if required, it will then be treated as a dummy option.

Sharability "Exclusive" is supported and will result in the task's exclusive access to the file.

Transaction control instructions are not supported. Commit instructions may be included in the application for compatibility if required. These will then be treated as dummy instructions.

7.3 FILE TYPES

ADM handles the following file types:

- Standard file
  A Standard file is a file where the records are identified by the relative key. Record length +1 must be a multiple of 256 bytes. The blocing factor must be 1.

- Load file (L file)
  An L file is a file containing a Monitor or application load module. The records are identified by the relative key. The records of an L file have no status byte. The record length is 256 bytes, the blocking factor is 1. L files can not be split into a number of extents.

- Undefined file (X file)
  An undefined file is a file of which the internal structure is not checked by data management. The records are identified by the relative key. The records of an X file have no status byte. The record lenght must be a multiple of 256 bytes, the blocing factor must be 1. X files may consist of a number of file extents and file sections.

ADM may be included in the Monitor on its own or together with SDM or EDM. In that case, S and E files will automatically be handled by SDM or EDM, while the L and X files are handled by ADM.

7.4 FILE CREATION

S, L and X files to be handled by ADM can be created by ADM or by the TOSS utilities.

7.4.1 File Creation under ADM

To create a file under ADM, the file must be opened for Output Direct. For standard files, Open mode Output Sequential is also allowed. The records are written to the file by Write Sequential or Write Direct instructions. When the file is closed the LRN will be written to the VTOC.

The remaining part of the file is not formatted with empty records when the file is closed. The contents of records after the LRN is undefined. For a standard file, the status byte of the records will be set to "free".

7.4.2 File Creation by TOSS Utilities

Creation of a file by the TOSS utility CRF is described in the TOSS Utility Reference Manual module M8A.

The restrictions for record length, blocking factor, number of extents and sections for each file type are found in section 7.3.

7.4.3 Enlarging Files

Standard files are automatically enlarged by ADM if during Sequential Write operations the end of the file is reached and the Growth Factor is not zero. The file is enlarged by adding another extent, the size of which is indicated by the Growth Factor in the File Parameter block. ADM does not allow the user to add new file sections, on another volume, to the files.
File enlargement is further discussed in chapter 2 section 2.6.

7.4.4 Buffer Management

ADM does not contain internal block buffers. All I/O is performed directly to and from the application buffer.

7.5 L AND X FILE HANDLING

L and X files can only be handled by a CREDIT application if Abridged Data Management (ADM) is included in the Monitor. The file handling instructions allowed are:

OPEN .IN	Open the file for read only.
OPEN .DOUT	Create a new file and open for direct output.
OPEN .INOUT	Open the file for input and output.
CLOSE	Close file.
CLOSE .DROP	Close file and delete it from the disk.
DSC	Read file parameters.

The record handling instructions available for L and X files are:

READ .DIR	Read direct
WRITE .DIR	Write direct
REWRITE .DIR	Rewrite direct
DISCARD .DIR	Delete record

To open an L or X file, the File Parameter block must be set up in the same way as for data files. Numeric fields contain a binary value and alphanumeric fields contain ISO-7 characters. The first 66 bytes of the File Parameter block is the same as for data files, with the following fixed data:

- Record length must be 256 for L files or a multiple of 256 for X files.
- Blocking factor must be 1
- File organisation must be 2 for L files or 3 for X files.

The File Parameter block must be extended with a further 22 bytes, filled with binary zeroes, where ADM will store additional information as described in Chapter 5, section 5.3.

Note also, that the binary data item containing the File Parameter block length must contain a value of 88 (X`0058').

7.6 RETURN INFORMATION

This section lists briefly the error messages that can be returned by ADM. A detailed description of the errors indicated by the status word and the return status, and possible remedies, is supplied in Chapter 10, Return Information.

7.6.1 Status Word

In the Status Word, the following bits may be set by ADM:

Bit 0	Request error
2	Boundary violation
3	End of File
7	Retries performed
8	Data Management rule violated. This bit can only be set after an 'Open File' instruction. When it is set, the Return Status may also be obtained.
9	File opened exclusive for other task
10	New volume loaded
12	Incorrect length
13	Data error
14	Throughput error
15	Disk not operable.

7.6.2 Return Status

The Return Status will only have a significant value after the Open File instruction. It may be set to one of the following values:

3	Overflow
4	Illegal file parameter
6	Illegal function option
7	File code already used (illegal file code)
9	File name or volume name unknown.

Chapter 8

STANDARD DATA MANAGEMENT

8.1 INTRODUCTION

This chapter discusses the file types handled by SDM, and how to create them. The indexing mechanism for an indexed file structure for SDM is explained. In addition, details of file enlargement under SDM and notes on file maintenance are given. A list of the file handling instructions supported by SDM and a survey of the Status Word and Return Status values that may be returned by SDM are also included.

8.2 SDM INSTRUCTION SET

The following instructions are supported by SDM:

File handling instructions:

OPEN .DOUT	Create a new standard or indexed file of S-type and open for direct output
OPEN .EXT	Open and Extend an existing standard file for sequential output
OPEN .IN	Open an existing file for input only
OPEN .INOUT	Open an existing file for input and output
OPEN .SOUT	Create a new standard file and open for sequential output
CLOSE	Close file
CLOSE .DROP	Close and delete file
POSIT .DIR	Set Current Record Number on specified record
POSIT .IXDIR	Set Current Record Number on record with specified key
DSC X'19'	Read File Parameters

Record handling instructions

READ .DIR	Read record with specified relative key
READ .SEQ	Read next record using the relative key
READ .IXDIR	Read record with specified symbolic key
READ .IXSEQ	Read record with next symbolic key
WRITE .DIR	Write record with specified relative key
WRITE .SEQ	Write next record using the relative key
WRITE .IXDIR	Write record with specified symbolic key
REWRITE .CUR	Rewrite current record
REWRITE .DIR	Rewrite record with specified relative key
REWRITE .IXDIR	Rewrite record with specified symbolic key
DISCARD .CUR	Delete current record
DISCARD .DIR	Delete record with specified relative key
DISCARD .IXDIR	Delete record with specified symbolic key

Transaction Control Instructions

COMMIT	Release the records accessed during the current transaction
COMMIT .REL	Release the records accessed during the current transaction, on the specified files only

Record Protection

When a file is opened with Sharability "Protected", any records accessed will be under exclusive access for the task. A second task requiring access to a record which is already under exclusive access will have the message "record protected" (bit 11) in the Status Word, and the other records held protected for this task will be released.

This is called automatic rollback, because the task should then go back to the previous COMMIT.

As a consequence, an application design is recommended where all records used during one transaction are read before any record is updated.

8.3 FILE TYPES

SDM handles standard files and indexed files of S-type.

It is possible to handle common files (with a common file code) in SDM but then the Assembler interface must be used for the Open, Close and I/O functions. Common files can not be handled by the CREDIT interface.

8.3.1 Standard Files

A standard file is a file where the records are identified by the relative key. The records have a fixed length and are grouped into blocks that always start on a logical sector boundary. Each record has a status byte indicating if the record is "used" or "free". There is no free record chain. The Last Record Number (LRN) defines the logical end of the file (see section 8.4.3). Up to 64 file extents are allowed per volume, and up to 4 file sections. The file name consists of one alphabetic ISO-7 character followed by up to 7 alphanumeric characters.

8.3.2 Indexed File of S-type

An indexed file of S-type is a data file with the same characteristics as a standard file. In addition to the relative key, the records are identified by up to four symbolic keys. Each key must be contained in a separate index. For each symbolic key there is an Index file and Master Index file of S-type.

8.3.3 Index File of S-type

An Index file of S-type has the same characteristics as a standard file. It contains one index of an indexed data file of S-type. Each record contains the symbolic key of a data record together with the relative key of that record in the data file. The index records are is ascending order of the binary value of the symbolic keys. An Index file of S-type can not have more than one file section.

Index files are sequentially searched during an indexed file access. The index file name must consist of the first 6 characters of the name of the data file to which it belongs, with the prefix "In" where n is the sequence number of the index. The prefix "I1" must always denote the primary index.

Index Record

Each index record has the following layout:

KEY	RESERVED	DUPLICATE KEY	RECORD NUMBER	STATUS BYTE

Where:

Key : The symbolic record key.
Reserved : 2 bytes containing binary zeroes.
Duplicate key : 1 byte binary value representing the minimum number of leading characters of the symbolic key which is identical with the symbolic key in the next index record.
Record number : 3 bytes containing the relative key of the data record identified by this symbolic key.
Status byte : One byte indicating if this index record is "used" (status X'FF') or "free" (status X'00').

8.3.4 Master Index File

A Master Index file of S-type has the same characteristics as a standard file. It contains the level 1 index or "master index" to one Index file of S-type. A Master Index file of S-type can not have more than one file section.

Each master index record corresponds with one partition in the Index file and contains the highest symbolic key and the relative record number of the first index record in that partition. The last record of the master index file contains the value X`FF' in every character position of the symbolic key field.

With the Master index file it can be determined which partition of the index file must be searched for the specified key. SDM performs a fast binary searchof the Master index.

The master index file name must consist of the first 6 characters of the data file name to which it belongs, with the prefix Mn where n is the sequence number of the corresponding index.

When an index file is opened, the corresponding master index file is read into memory. When the file is closed, the master index area in memory is released.

After the index and master index files have been built or reorganised by utility RIX, the number of index records per index partition is equal to:
The number of used index records divided by the total length of the master index expressed in number of records.

During dynamic use of the file structure, when records are added or deleted, the index file is updated but the master index is not and after some time the number of used index records in each partition will vary. Running RIX again with the same size of the master index, will result in a smaller or larger number of index records per partition.

It is also possible to have an empty master index, e.g. when there is no memory space available for it. After running RIX the master index file must then be deleted, and a new file of S-type with the same master index name must be created but left empty (LRN = 0). When the index file is opened this empty master index is read into memory, occupying only a few bytes. As a result, the index file will always be searched from the beginning (sequential search).

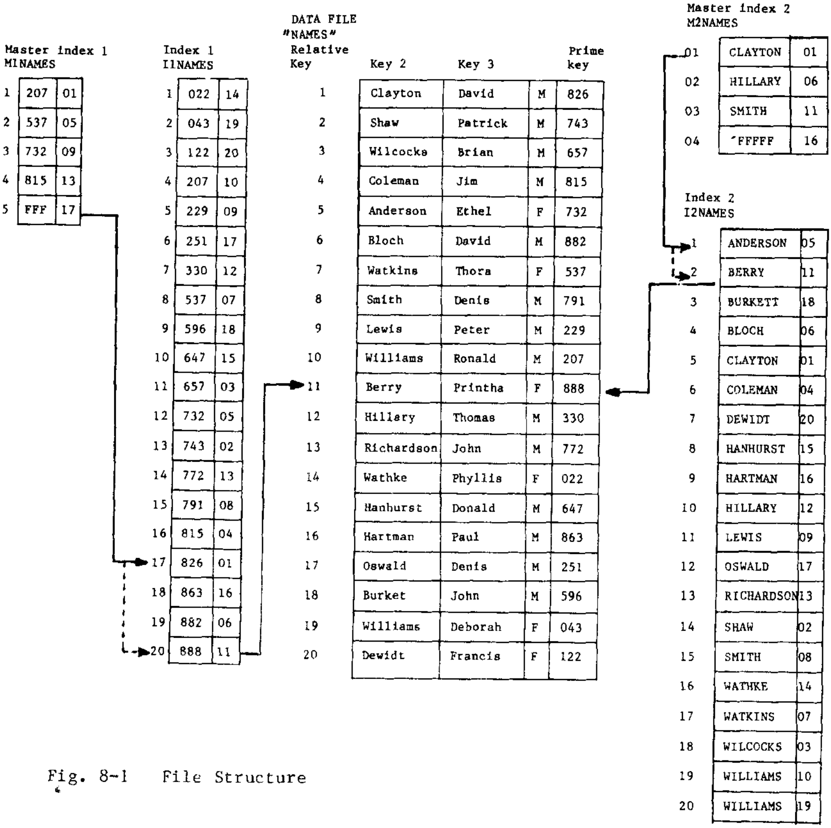
Both index files and master index files may be opened as standard files and the records read, written and updated if required. Consistency of the files is the user's responsibility.

For index files and master index files several file extents are allowed but all index and master index files belonging to one data file must reside on one volume.

8.3.5 File Structure

Data file, index files and master index files together constitute a file structure. A data file with the name NAMES may have the index files I1NAMES, I2NAMES, with their master indexes M1NAMES and M2NAMES.

Fig 8-1 is an example of such a file structure. Data record 11 is found via prime key "888" and via second key "Berry".



Master index 1
M1NAMES

1	207	01
2	537	05
3	732	09
4	815	13
5	FFF	17

Index 1
I1NAMES

1	022	14
2	043	19
3	122	20
4	207	10
5	229	09
6	251	17
7	330	12
8	537	07
9	596	18
10	647	15
11	657	03
12	732	05
13	743	02
14	772	13
15	791	08
16	815	04
17	826	01
18	863	16
19	882	06
20	888	11

DATA FILE
"NAMES"

Relative Key	Key 2	Key 3		Prime key
1	Clayton	David	M	826
2	Shaw	Patrick	M	743
3	Wilcocks	Brian	M	657
4	Coleman	Jim	M	815
5	Anderson	Ethel	F	732
6	Bloch	David	M	882
7	Watkins	Thora	F	537
8	Smith	Denis	M	791
9	Lewis	Peter	M	229
10	Williams	Ronald	M	207
11	Berry	Printha	F	888
12	Hillary	Thomas	M	330
13	Richardson	John	M	772
14	Wathke	Phyllis	F	022
15	Hanhurst	Donald	M	647
16	Hartman	Paul	M	863
17	Oswald	Denis	M	251
18	Burket	John	M	596
19	Williams	Deborah	F	043
20	Dewidt	Francis	F	122

Master index 2
M2NAMES

01	CLAYTON	01
02	HILLARY	06
03	SMITH	11
04	´FFFFF	16

Index 2
I2NAMES

1	ANDERSON	05
2	BERRY	11
3	BURKETT	18
4	BLOCH	06
5	CLAYTON	01
6	COLEMAN	04
7	DEWIDT	20
8	HANHURST	15
9	HARTMAN	16
10	HILLARY	12
11	LEWIS	09
12	OSWALD	17
13	RICHARDSON	13
14	SHAW	02
15	SMITH	08
16	WATHKE	14
17	WATKINS	07
18	WILCOCKS	03
19	WILLIAMS	10
20	WILLIAMS	19

Fig. 8-1 File Structure

8.4 RECORD IDENTIFICATION

8.4.1 Record Keys

Symbolic record keys must consist of one key item or character string, and the key length must not exceed 64 bytes.

Prime Key

At least one of the keys must be unique for each data record. This is the prime key. The index containing the prime key must be defined as the first index when the file is created or opened.

The prime index must not contain duplicate keys. The records are identified by the prime key for the following instructions:

Write Indexed Sequential
Write Indexed Direct
Rewrite Indexed Direct
Delete Indexed Direct

The other keys are called alternate keys, and for those duplicates may exist. SDM does not check to see if there are duplicates. Note that the parameter "Index Type", indicating if duplicate keys are allowded for an index, is not significant for SDM but only for EDM.

Duplicate Keys

Duplicate keys are keys that have the same value in a number of data records. For indexed accesses on these records, the first one is found by an indexed direct access and the others are then accessed by indexed sequential instructions.

8.4.2 Currency

SDM holds a Current Record Number (the CRN) per data file for each task. This is the relative key of the current record for the task. The CRN is set by Read and Posit instructions. The CRN is used for record identification by Read Sequential, Rewrite Current and Discard Current instructions.

In SDM, the Posit instruction can not be used to set the CRN for Rewrite and Discard instructions. Rewrite Current and Discard Current after a Posit instruction will access the record last read.

Per task, SDM maintains the currency for one index at the time, that is the index specified for the last indexed instruction. The currencies for the other indexes associated with the data file will be zero, so that the data record associated with the first entry in such an index is accessed when a different index is used.

The Index currency is used for record identification by Read Indexed Sequential instructions.

8.4.3 Last Record Number (LRN)

Per S file (not per task), SDM holds a Last Record Number (the LRN). This denotes the logical end of the file, and the start point for subsequent Write instructions except Write Direct on a standard file.

For a Standard file, the LRN points to the last record written by Write Sequential instructions.

For an Indexed file of S-type, the data records are written to the data file sequentially with Write Indexed Direct instructions. The LRN points to the last record written by Write Indexed Direct instructions.

When the LRN is reached by Read Sequential instructions (CRN and LRN have the same value), the message "End of file" is returned.

When a new file is created, the LRN is preset to zero.

With Read Direct and Write Direct instructions (non-indexed!) it is possible to write and read records after the LRN. The LRN is not updated by direct access instructions. However, only "used" records can be read and rewritten, and only "free" records can be written.

The LRN is stored in the VTOC record for the first file extent. When a file is opened the LRN is read into memory and updated when new records are written to the file, except by non-indexed Write Direct instructions. The LRN is written back to the VTOC record when the file is closed.

For an Index file of S-type, the LRN is the relative key of the last index record in the last used partition of the file. This may be a "free" record when index entries have been deleted afterwards, because the LRN is not updated when records are deleted.

8.5 FILE CREATION

Indexed files of S-type are indexed random files, where the data records need not be in sequence of any of the keys. The records are located internally by the relative key.

When an indexed file structure is created, the file size specified for the data file and the index files must allow for updates on the file because SDM does not allow re-use of deleted records, and indexed files can not be extended.

Standard files and indexed files of S-type may be created by SDM or by the TOSS utilities Create File (CRF), Build Index File (BIX) and Reorganise Index File (RIX). These are described in the TOSS Utility Reference Mnaual, module M8A.

8.5.1 Creating a File by SDM

New standard and indexed files can be created during runtime by SDM. To create a new file the file must be opened with open mode Output Sequential or Output Direct.

The number of index files and master index files specified in parameter "Number of Indexes" is created on the volume specified for "Index Volume Name". The index files will have the size needed to contain the number of index records equal to the number of records specified for the data file. The master index files will have the size needed to contain the number of master index records equal to the number of index records in the index file, divided by 8.

When an indexed file structure is created by SDM, the master index file will be created but it is not filled. After writing the data records to the file, the master index must be built by running the utility RIX as described in phase 7 in the next section.

When opened for Output Direct, the file will be formatted at the Open instruction (the records are filled with spaces and the status is set to "free"). When opened for Output Sequential, the part of the file after the LRN will be formatted when the file is closed.

8.5.2 Creating Files by the TOSS Utilities

Creation of a file structure is performed in the following phases:

1 The data file must be created by the TOSS utility Create File (CRF).
2 The index file and master index file must be created with utility CRF.
3 The data records are written to the file.
4 One or two intermediate files must be created as workfiles to sort the data records and build the index file.
One work file is needed if the records in the file are in sequence of the key for which the index file is built. If this is not so, one extra workfile is needed for the Sort utility.
5 The intermediate index file must be built on the first workfile, by the TOSS utility Build Index File (BIX).

6 The index records in the workfile must be sorted on key value, by the TOSS utility Sort (SRT). Output of SRT is the second workfile.

7 The index file is built from the sorted workfile, by the TOSS utility Reorganise Index File (RIX).
Free records are distributed over the sectors on the index file according to the load factor specified, and the master index is created.

8 Data file, index file and master index file are now available for use by the application. The workfiles can be deleted by the TOSS utility Delete File (DLF).

If the data records contain more than one symbolic key, steps 4 through 8 must be repeated for every index file of the file structure.
All index files and master index files belonging to the same data file must reside on one volume.

Detailed descriptions of the utilities CRF, BIX, SRT and RIX are found in the TOSS Utilities Reference Manual M8A.
Most of the parameters for the utilities are self-explanatory.
However, some that may need more explanation are discussed here:

Phase 1 - Create the data file with CRF.

- File organisation: "S" must be stated for all files.
- Number of records:
  File size should allow for extension and updates of the file.
- Number of index files:
  Up to 4 index files may be specified. If there are no indexes, answer zero.
- Key address in data record:
  This question is only relevant when creating an index file. When creating the data file answer zero.

CRF now searches the volume(s) for free extents large enough to hold the stated file size. The file is created with the required number of records, all containing space characters and all with a status byte indicating "free" (X'00'). The LRN is set to zero in the VTOC record of the first file extent.

Phase 2 - Create the index file and master index file with utility CRF.

- File name:
  For the index file, the file name is the first six characters of the data file name, with the prefix In where n is the sequence number of the index file. The prefix "I1" denotes the primary index.
  For the master index file, the file name is the first six characters of data file name with the prefix Mn where n is the sequence number of the associated index file.
- File organization :
  "S" must be specified for index and master index files.
- Volume name:
  For index files and master index files the same volume name must be specified.
- Index volume name:
  Not relevant when creating index and master index files.

- Blocking factor:
  Number of records per block. This is not the number of index records per partition. The blocking factor is determined in the same way as for the data file.
- Record length:
  For both files, the record length depends on the length of the symbolic key of the data record.
  For the index file the record length must be keylength +6, and for the master index file the record length must be keylength +3, specified in bytes.
- Number of indexes:
  Not relevant, answer zero.
- Number of records:
  For the index file, a larger number of records must be specified than the number of records for which the data file was created, to allow for insertion of index records into the last sector of the index file.
  For the master index file, the most efficient file size must be chosen for the memory space available and the access times required. See also section 8.5.3, Master Index File Size.

Phase 3 - Write data records to the file.

In most cases the data records will be written to the file by the application. This may be performed by Sequential or Direct Write instructions.

The LRN is not updated by Direct Write instructions. However, TOSS utility BIX does not check the LRN but reads the entire data file until End of Medium, ignoring records with status "free".

Phase 4 - Create the intermediate files with CRF

CRF is run to create the work files. File names and volume names may be chosen as convenient.

- Record length:
  For both files record length must be the same as for the index file, which is keylength +6.
- Number of records:
  Number of records must be at least the (estimated) number of used data records now in the data file. This number is equal to the number indicated by the LRN of the data file if the records have been written with Sequential Write instructions.
  No free space is needed in the intermediate files.
- Key address in data record:
  Answer zero when creating the intermediate files.

Phase 5 - Build intermediate index file with BIX

- Address of key in record:
  Specify the position of the first character of the symbolic key in the data record. The first character position in the data record is counted as zero.
- Key length:
  Specify the key length in characters (max 64).

BIX then scans the data file and copies the specified fields to the workfile, together with the relative keys of the data records. The index records thus built are written to the workfile sequentially, irrespective of the value of the key.

Phase 6

Sort the index records, if necessary. The TOSS utility Sort File (SRT) is run to sort the index records. Sort is not necessary if the data records have been written to the file in key sequence.Sorting is done on the binary value of the symbolic keys. The input file for Sort is the intermediate index file as built by BIX, and the output file is the second workfile.
Parameters for Sort routine:

- Sub-key address in record:
  Answer zero. The intermediate index records start with the symbolic keys.
- Sub-key length:
  Specify the length of the symbolic key, in number of characters.
- Max number of records:
  This must be equal to the number of records on the data file.
- Effective record length:
  Answer zero to sort the complete index records. For duplicate keys the index records will be in sequence of the relative keys of the data records.
- Ascending order:
  Answer yes.

Phase 7 - Build the index file and master index file with RIX

This phase is also needed if a new indexed file structure has been created under SDM.
Utility Reorganize Index file (RIX) is run to build the index file and master index file. Input for RIX is the sorted intermediate file, either created by Sort or by BIX if the data records were already in key-sequence on the data file.

The size of the partitions of the index file is determined by the size of the master index file. Each master index record corresponds to one index partition. The number of records per index partition is equal to:
the number of used index records divided by the total number of master index records (the length of the master index file expressed as number of records).

The following questions will be output by RIX:

- Maximum number of records on the output index file:
  An estimation of the highest record number (the LRN) on the new index file. This can be derived from the LRN of the data file, and the load factor. The space in the new index file after the record indicated by the LRN is reserved for future extensions.
- Load factor:
  A decimal value indicating which percentage of each block in the index file must be used. See the discussion of the load factor in section 8.5.4.

The utility RIX returns the following information to the operator:

- Number of index records per partition: nnnn.

The next question: "OK?" can be answered with Yes if this number corresponds with what was estimated and with the requirements of the application (mainly regarding access times). If this is not so, answer `No'. The utility will be aborted and a master index file with a different size can be created before running RIX again.

Index records are read from the sorted work file and written to the index file in the required format. Free records are added at the end of each sector according to the load factor.
Records are written to the master index file sequentially. RIX performs a check on the record sequence. If a key sequence error is detected, the utility is aborted and an error message is output on the operator's console.

8.5.3 Master Index File Size

The master index file size influences the performance.

If the master index is very large this may cause memory problems. The search time is not much influenced by the master index size because SDM does a binary search of the master index.

If the master index is too small, there is a large number of index records per partition. This means many disk accesses and sequential search of the index blocks, which will reduce the performance.

A master index file created during runtime will have the size needed to contain the number of master index records equal to the number of records specified for the data file, divided by 8.

8.5.4 Load Factor

SDM does not allow the index files to be extended. When the index file is created, two characteristics specified by the user must allow for future extension of the data file:

- Index file size
  The number of records in the index file must be larger than the number of records in the data file, to allow for index entries with a high key value to be added in the last partition of the index file.

- Load factor
  When new records are written to the data file by the application, the corresponding index records must be inserted in the index file in the correct position. To make this possible free space is reserved in every block. The Load Factor determines which percentage of each block of the index file will be filled with index records when the index file is built.

Example 1

7 Data records have been written to the data file, and utility RIX is run with a load factor of 30 specified. The blocking factor of the index file is 10. RIX will build an index file where every block is filled for 30%.

DATA FILE
"NAMES"

Relative Key	Key 2	Key 3		Prime key
1	Clayton	David	M	826
2	Shaw	Patrick	M	743
3	Wilcocks	Brian	M	657
4	Coleman	Jim	M	815
5	Anderson	Ethel	F	732
6	Bloch	David	M	882
7	Watkins	Thora	F	537
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

1	537	07	Block 1
2	657	03	
3	732	05	
	FREE		
	SPACE		
	-		
	-		
	-		
	-		
10			
11	743	02	Block 2
12	815	04	
13	826	01	
14	FREE		
	SPACE		
	-		
	-		
	-		
	-		
	-		
21	882	06	Block 3
22	FREE		
23	SPACE		

The next data record written to the file has prime key 791. The data record gets the relative key 8. The new index entry is inserted in the index file at position 12 and the entries 12 and 13 are shifted to the positions 13 and 14 to make space.

After some time the data file has been filled to 90%. Index entries are inserted in the proper positions. The first block has no free space left, the second block has 40% and the third block has been needed for the highest keys.

DATA FILE "NAMES"

Relative Key	Key 2	Key 3		Prime key
1	Clayton	David	M	826
2	Shaw	Patrick	M	743
3	Wilcocks	Brian	M	657
4	Coleman	Jim	M	815
5	Anderson	Ethel	F	732
6	Bloch	David	M	882
7	Watkins	Thora	F	537
8	Smith	Denis	M	791
9	Lewis	Peter	M	229
10	Williams	Ronald	M	207
11	Berry	Printha	F	888
12	Hillary	Thomas	M	330
13	Richardson	John	M	772
14	Wathke	Phyllis	F	022
15	Hanhurst	Donald	M	647
16	Hartman	Paul	M	863
17	Oswald	Denis	M	251
18	Burket	John	M	596
19				
20				

1	022	14	Block 1
2	207	10	
3	229	09	
4	251	17	
5	330	12	
6	537	07	
7	596	18	
8	647	15	
9	657	03	
10	732	05	
11	743	02	Block 2
12	772	13	
13	791	08	
14	815	04	
15	826	01	
16	863	16	
17	FREE		
18	SPACE		
19			
20			
21	882	06	Block 3
22	888	11	
	FREE		
	SPACE		

1	022	14	Block 1
2	207	10	
3	229	09	
4	251	17	
5	330	12	
6	537	07	
7	596	18	
8	647	15	
9	657	03	
10	FREE		
11	732	05	Block 2
12	743	02	
13	772	13	
14	791	08	
15	815	04	
16	826	01	
17	863	16	
18	882	06	
19	888	11	
20	FREE		
21	FREE		Block 3
	SPACE		

To distribute the free space evenly over the blocks, utility RIX is run again. The data file is now filled to 90%, and for the index file a load factor of 90 is specified. Block 3 of the index file becomes free.

Fig 8-2 Load Factor

If the keys of the next data records that are added to the file have values lower than 888, the index entries are inserted in the free space in block 1 and block 2. But if they have high key values they must be inserted after the last entry and the index records will be shifted into block 3. For this reason, the index file must allow for more entries than would be needed for a 100% filled data file.

Example 2

A file structure is created and only few data records are as yet available. The load factor specified for the index file is 10%. This leaves 90% free space in every sector. After the application has written a number of records to the file and the corresponding index entries have been inserted in the index file, some sectors may still be almost empty and others may have little free space left.

To distribute the free space evenly over the sectors again, the TOSS utility Reorganise Index File must be run.

The load factor specified this time must reflect the new status of the data file. If the data file now has been filled for about three quarters, a new load factor of 70 or 75 may be specified. The index file will be rebuilt and every sector will contain 30% or 25% empty space.

8.6 ENLARGING FILES IN SDM

Only standard files can be enlarged by SDM. Files will be automatically extended by SDM when during Write Sequential instructions the end of the data file (indicated by the LRN) has been reached and the Growth Factor is not zero. Standard files are explicitly enlarged when opened with Open mode Extend.

A new extent is added to the file, with the size indicated by the Growth Factor in the File Parameter Block, rounded upwards to a file extent length which is multiple of three logical sectors and of the block length. SDM does not allow new file sections (on another volume) to be added to the file.

8.7 FILE MAINTENANCE

Discarded data records in an indexed file of S-type can not be reused. When such a file has been much updated, it is necessary to run the TOSS utility Reorganise Index file (RIX) again, to reallocate the free space available and to update the master index. When RIX is rerun after some time, a new load factor can be specified, representing the real or estimated percentage of used records in the data file.

The TOSS utility RIX must be run in the following situations:

- The message End of File has been returned after a Write Indexed Direct instruction, to indicate that index records in the last block have been written after LRN. When a number of free blocks are available at the end of an index file, the message End of File (Condition Register set to 1) is returned and bit 3 is set in the Status Word each time when a free block is used. This may occur several times before it is necessary to reorganize the files.

- When index records have been shifted into the next partition, the master index file no longer represents a good picture of the index file. As searching of the index file for the specified key is done sequentially, starting at the record pointed to by the master index, the correct record will still be found, but search time increases.

- If the last block of the index file is filled completely the message End of Medium is returned (condition register set to 3) and the Write instruction is not completed. The files must be closed, and BIX-SORT-RIX must be run immediately.

- After the data file has been much updated and consists of many file extents or contains many discarded records, it may be reorganized by copying it into a newly created data file of S-type. BIX, SORT, RIX must then be run to build the index file and master index file for each index.

8.8 FILE RECOVERY

The possibilities for file recovery in SDM are limited. The following points must be taken into account when designing recovery procedures:

For files opened without the Delay option (see section 8.7.2, Delay option), each Write instruction issued by the application results in a disk access and the records are written to the file immediately.

When a file is opened with the Delay option, the information is written to the block buffer, and will only be written to the disk when another block of this file must be accessed, or when the file is closed.

During Write Sequential and Write Indexed Direct instructions, the LRN of the data file is updated in memory and not written to the VTOC on disk until the file is closed.

If a system failure occurs during updating of the file the value of the LRN will be lost. When the file is opened again the old value of the LRN will be read from the VTOC. Used records after the old LRN can be recovered with the instruction sequence:

- Read Sequential up to End of File (the old LRN is reached)
- Read Direct, reading the used record after the old LRN
- Discard Indexed Direct, using the symbolic keys of the record just read. The index entries are deleted from the index files (all indexes must be opened!).
- Write Indexed Direct, using the keys of the record just read and discarded. The LRN will be updated, and new index entries will be inserted in the indexfiles.

8.9 BUFFER MANAGEMENT

8.9.1 Block Buffers

SDM contains internal block buffers, used for the data files and for the index blocks. The block buffers are physically fixed in memory and have a fixed length. This means that they must be long enough to contain the longest block that has to be accessed, and that it may be inefficient use of memory space to have files with very different block lengths.

The minimum number of block buffers is 2 per disk driver.
The maximum number of block buffers that SDM can handle is 16. Which buffer will be used for a request is determined by a Least Recently Used (LRU) algorithm. A block buffer remains attached to a file during an I/O instruction and is released when the I/O is completed, except when the Delay option is used.

When an I/O operation is started and there is no free block buffer, SDM will wait until a block buffer is released.

The number of block buffers and their length is specified during Monitor generation.

8.9.2 Delay Option

SDM supports the Delay option. When this option is specified, a block buffer is attached to the file when the file is opened, and released when the file is closed.

Updates to the file are not written to the disk immediately but to the block buffer. This buffer is written to the disk when another block must be accessed, when the file is closed or when there are no free block buffers available.

The Delay option improves performance when files with a blocking factor greater than 1 are processed sequentially.

One extra block buffer per file opened simultaneously for which the Delay option is required, must be reserved during Monitor generation.

8.10 RETURN INFORMATION

Under SDM, the following return information may be generated:

8.10.1 Status Word

The following bits can be set in the Status word. These error messages are further discussed in chapter 10, Return Information.

bit 0	Request Error
bit 2	End of Medium
bit 3	End of File
bit 4	No Data
bit 5	Key not Found
bit 6	Duplicate Key at Read Indexed Sequential Instruction
bit 7	Retries performed for the disk transfer
bit 8	Data Management rule violated, more information in Return Status
bit 9	Duplicate Key Error
bit 10	New Volume Loaded
bit 11	Protection Error, Rollback
bit 12	Incorrect Length
bit 13	Data Error
bit 14	Throughput Error
bit 15	Disk not Operable

8.10.2 Return Status

In SDM, the Return Status may be set to the following values:

1	Not enough memory
3	Overflow
4	Illegal File Parameter
7	Illegal file code
8	Illegal ECB parameter
9	File name unknown

Chapter 9

EXTENDED DATA MANAGEMENT

9.1 INTRODUCTION

This chapter discusses the file types handled by EDM and how to create them, the types of indexing available and the logging functions supported by EDM. In addition, details of file enlargement in EDM and notes on EDM file maintenance are given.

9.2 INSTRUCTION SET

File handling instructions:

OPEN .DOUT	Create a new file and open for direct output
OPEN .EXT	Open and Extend an existing standard file for sequential output
OPEN .IN	Open an existing file for input only
OPEN .INOUT	Open an existing file for input and output
OPEN .SOUT	Create a new file and open for sequential output
CLOSE	Close file
CLOSE .DROP	Close and delete file
POSIT .DIR	Set Current Record Number on specified record
POSIT .IXDIR	Set Current Record Number on record with specified key
DSC X'19'	Read File Parameters

Record handling instructions

READ .DIR	Read record with specified relative key
READ .SEQ	Read next record using the relative key
READ .IXDIR	Read record with specified symbolic key
READ .IXSEQ	Read record with next symbolic key
WRITE .DIR	Write record with specified relative key
WRITE .SEQ	Write next record using the relative key
WRITE .IXDIR	Write record with specified symbolic key
WRITE .IXSEQ	Write record with next symbolic key
REWRITE .CUR	Rewrite current record
REWRITE .DIR	Rewrite record with specified relative key
REWRITE .IXDIR	Rewrite record with specified symbolic key
DISCARD .CUR	Delete current record
DISCARD .DIR	Delete record with specified relative key
DISCARD .IXDIR	Delete record with specified symbolic key

Transaction Control Instructions

COMMIT	Release the records accessed during the current transaction
COMMIT .PROT	Release the records accessed during the current transaction except those on the specified files
COMMIT .REL	Release the records accessed during the current transaction, on the specified files only
ROLLBCK	Rollback the current transaction

9.3 FILE TYPES

EDM handles standard files and indexed files of E-type (EDM-files).

9.3.1 Standard Files

A standard file is a file where the records are identified by the relative key. The records have a fixed length and are grouped into blocks that always start on a logical sector boundary. Each record has a status byte indicating if the record is "used" or "free". There is no free record chain. The Last Record Number (LRN) defines the logical end of the file (see section 8.4.3). Up to 64 file extents are allowed per volume, and up to 4 file sections. The file name consists of one alphabetic ISO-7 character followed by up to 7 alphanumeric characters.

9.3.2 Indexed File of E-Type

Indexed files of E-type (EDM files) are indexed random files. The data records are identified by up to 10 symbolic keys. Symbolic keys may consist of up to 64 key-items. The file name consists of one alphabetic ISO-7 character followed by up to 7 alphanumeric characters.

Internally, each data record is located by EDM by its relative key.

The data file of an EDM file is the "D-file" and the index file is the "I-file".

D-file

A data file of E-type is a file which contains only data records. The records are identified by the application by symbolic keys. Internally, the records are identified by the relative key. The records have a fixed length and are grouped into blocks that always start on a logical sector boundary. Each record has a status byte indicating if the record is "used" or "free".

The D-file has a free record chain (see also section 9.3.3). Discarded records are added to the chain and thus may be re-used. The relative key of the first free record in the chain is stored in the VTOC, as described in chapter 3. Up to 64 file extents are allowed per volume, and up to 4 file sections. The file name consists of one alphabetic ISO-7 character followed by up to 7 alphanumeric characters.

I File

The I-file is a file which contains all the indexes defined for one indexed EDM file. The index entries are grouped into records. The record length of the I-file is always 256 bytes, with a blocking factor of 1.

The index file name must consist of "I$" followed by the first 6 characters of the corresponding data file name.

Each index entry contains the key with the highest value in one index block on the one lower level, plus the relative key of that block. The entries in each level are sorted on the value of the keys in ascending order. The keys are packed, that is, only the part that is different from the preceding key is stored in the index entry. Each key is compared with the preceding key from left to right, to find the first character which is different. The key is then stored, starting from this character.

The first index block contains the index descriptors, which are discussed in Chapter 5.

Each index block except the first one has the following layout:

LEVEL NUMBER	INDEX ENTRIES	FREE SPACE	STATUS BYTE

Level Number : One byte containing the sequence number of the index level of this index block.

Index Entry : Each index entry consists of the following items:

Key Length : One byte. The number of characters of the symbolic key stored in this entry.
Key : The part of the symbolic key which is different from the previous key.
Pointer : Four bytes, containing the relative key of the index block at the next lower level which has this key as the highest key value. For the entries at the lowest level, this is the relative key of the corresponding data record.

Free Space : Starting with a dummy index entry, with a key field filled with X'FF' and a pointer value of zero. This is to prevent all index levels from having to be updated if a key is inserted with a higher value than the highest value that occurs in the index.

Status Byte : Value X'00' if the record is free, and the value X'FF' if the record is used.

The size of an I-file, in sectors, is:

$$n + 1 + \sum_{1}^{n} \frac{(\text{keylength} + 5) * \text{number of records}}{\text{logical sector length} - 2} * 3$$

'n' is the number of indexes defined for the data file.
The size will be rounded upward to a multiple of three sectors.

Index Levels

Indexes in EDM may have up to 16 levels. The index with references to the data records has index level zero. Index levels are transparent to the application program.

For an indexed direct access, first the highest index level is searched for a symbolic key with the same or a higher value than the key specified. The index entry thus found, indicates from which point the next lower level must be searched. The next lower level is then searched for a key with the same or a higher value than the key specified, and this index entry again indicates from where to search the next lower level. This is repeated until, on index level zero, the reference to the required data record is found.

When an index block in any level becomes full, (no free space left) its contents are split over two blocks each filled about half, and on the next higher level one new index entry is created corresponding with this new block. If the block on the higher level becomes full, it is split in the same way and on the next higher level one new index entry is added.

When there are already 16 index levels and a block at the highest level has to be split, an automatic rollback is performed if transaction logging is required for the file. Without transaction logging, the I-file is left in an inconsistent state (File Corrupt). If function logging is required, the files can be recovered with the utility Recover EDM File (RCF). The application is informed in the Return Status (value 10).

9.3.3 Free Record Chain

Deleted data records are re-used by EDM. The "File Record Number" in the VTOC record for the file contains the relative key of the first free record. This first free record then contains the relative key of the next free record, and so on. When a data record is deleted, its relative key is added to this free record chain. When new records are written to the file, they are written into these free record positions and the chain is updated. When the file is closed, the updated File Record Number is written back to the VTOC on disk.

9.3.4 File Status

The first byte of the I-file indicates the status of the I-file (correct or corrupt). The value of this byte is set to 1, indicating corrupt, when the Monitor detects an I/O error during a block split in the I-file or EDM detects an inconsistency. When the file is accessed and found to be corrupt, the request is completed with bit 0 and 8 set in the Status Word and with Return Status indicating I/O error. In that case, the only order that will be accepted is Close File. The files may have to be recovered from the backups with the information from the function log file.

9.4 RECORD IDENTIFICATION

9.4.1 Prime Key

At least one of the symbolic keys must be unique for each data record. This is the prime key. The index containing the prime key must be defined as the first index when the file is created or opened. The other keys are called alternate keys, and for those the File Parameter "Index Type" may indicate that duplicates are allowed.

9.4.2 Concatenated Keys

Record keys may be concatenated keys, that is, they may consist of several data items (up to 16). The key descriptor must contain the number of items the key consists of and their position within the data record.
The keylength of these keys is the sum of the separate key items.

9.4.3 Duplicate Keys

If the File Parameter "Index Type" indicates that duplicates are allowed, the symbolic key may have the same value in several records. For indexed accesses on these records, the first one is found by an indexed direct access and the others may be accessed by indexed sequential operations. Figure 9-1 gives an example for the key "Williams", which occurs twice in the data file.

9.4.4 Conditional Indexing

If conditional indexing is required, reference to a data record is only included in the index where it should go according to the symbolic key under certain predefined conditions.

One character item of the data record is defined to be the Conditional Item. The value of this item is compared with the value of Conditional Item Value set in the conditional index description by the application when the file was opened. If the condition, which may be "equal" or "unequal", is satisfied, the entry is included in the index.

During updating of the record the value of the conditional item may be changed, e.g. an account holder's balance may change from negative to positive, or the amount on stock of a certain article may change from positive to zero or to less than a minimum value. When the record is rewritten to the file the index will be updated accordingly: the entry is deleted from the index for which the conditional item no longer has the required value, and/or included in the index for which the condition is now fulfilled.

Example

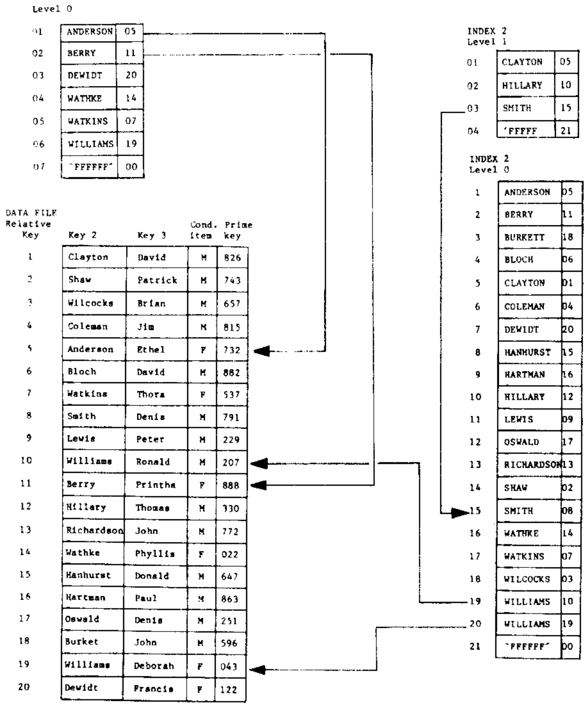
For the file in fig. 9-1, the item in position 22 in the data record is defined as a conditional item for index 4. The condition specified is `Equal', the index only contains entries for the data records for which the conditional item is equal to `F'.

The record for a female with the name of Williams is retrieved by an indexed access via index 4 with the key 'Williams'. This will only give record 19, and the application need not check the item indicating 'F' or 'M'.

To find the record for a man with the name of Williams, a fifth index can be specified with the same conditional item. The same conditional value can be stated and the condition Not Equal specified, but then also records with an erroneous contents (for example, 'G') for this item will be included. It is safer to specify conditional value 'M' and the condition Equal.

INDEX 4
Conditional Index
Cond. Item value "F"

Duplicate Keys
in index 2



Level 0

01	ANDERSON	05
02	BERRY	11
03	DEWIDT	20
04	WATHKE	14
05	WATKINS	07
06	WILLIAMS	19
07	'FFFFFF'	00

DATA FILE Relative Key	Key 2	Key 3	Cond. item	Prime key
1	Clayton	David	M	826
2	Shaw	Patrick	M	743
3	Wilcocks	Brian	M	657
4	Coleman	Jim	M	815
5	Anderson	Ethel	F	732
6	Bloch	David	M	882
7	Watkins	Thora	F	537
8	Smith	Denis	M	791
9	Lewis	Peter	M	229
10	Williams	Ronald	M	207
11	Berry	Printha	F	888
12	Hillary	Thomas	M	330
13	Richardson	John	M	772
14	Wathke	Phyllis	F	022
15	Hanhurst	Donald	M	647
16	Hartman	Paul	M	863
17	Oswald	Denis	M	251
18	Burket	John	M	596
19	Williams	Deborah	F	043
20	Dewidt	Francis	F	122

INDEX 2
Level 1

01	CLAYTON	05
02	HILLARY	10
03	SMITH	15
04	'FFFFF	21

INDEX 2
Level 0

1	ANDERSON	05
2	BERRY	11
3	BURKETT	18
4	BLOCH	06
5	CLAYTON	01
6	COLEMAN	04
7	DEWIDT	20
8	HANHURST	15
9	HARTMAN	16
10	HILLARY	12
11	LEWIS	09
12	OSWALD	17
13	RICHARDSON	13
14	SHAW	02
15	SMITH	08
16	WATHKE	14
17	WATKINS	07
18	WILCOCKS	03
19	WILLIAMS	10
20	WILLIAMS	19
21	'FFFFFF'	00

Fig. 9-1 Conditional Indexing and Duplicate Keys

9.4.5 Currency

EDM holds a current record number per data file for each task. This is the relative key of the current record for the task. The currency is set by Read and Posit instructions. The currency is used for record identification by Read Sequential, Rewrite Current and Discard Current instructions. Rewrite Current and Discard Current after a Posit instruction will access the record that has become the current record as a result of the Posit instruction. When a file is opened the currency is set to zero so that the record with relative key 1 is read with the first Read Sequential (non-indexed) instruction.

Per task, EDM holds the currency for each index opened for the file. When a file is opened all currencies will be zero, so that indexed sequential access starts with he record associated with the first entry for each index.

The index currency is used for record identification by Read Indexed Sequential instructions.

9.5 LOGGING

9.5.1 File Recovery

File recovery after error situations or system failure is possible when logging is used. Two types of logging are implemented in EDM:

- Transaction logging
- Function logging

Transaction logging is needed for file recovery when a single transaction can not be successfully completed. It is possible that while a transaction is being executed, it becomes necessary to cancel it. It may be that erroneous data have been keyed in by the operator, that the input data from the files are in conflict or that it is not possible to access all the records needed for the transaction. Transaction logging makes it possible to undo a transaction that has not yet been completed, and bring the files back in the consistent state they had at the start of the transaction.

Function logging is needed for file recovery after hardware failures. Disk failures or power failures may disturb the files that are on-line so that they can no longer be used. The transactions performed up to that moment are lost. With function logging, a log has been made of the functions performed and with this a back-up copy of the files can be recovered before the application continues.

9.5.2 Transaction Logging

Transaction logging means that for every record which is modified during a transaction, a "before image" is stored in the log file. On the execution of a Rollback (programmed or implicit by EDM) the before images are restored to the files, and the files are again in the consistent state they were in at the previous Commit. The currency for these files will also be reset to the value at the previous Commit. For this reason it may be useful to require transaction logging for a file opened for Input.

At the execution of a Commit, Rollback, Open or Close instruction, the information is deleted from the log file and logging of the next transaction is initiated.

Note that Rollback does not guarantee that the before images are physically written back to the disk. They are restored to the internal buffers of EDM and EDM determines when the I/O should take place.

If transaction logging is required for accesses on a file, it must be specified in the File Parameter block when the file is opened.

9.5.3 Transaction Log Information

The records of a transaction are stored in the transaction log file in blocks of 255 bytes. The first 22 bytes contain system information. The before-images are stored immediately after that. User records are stored without the status byte. For before-images longer than the remaining 232 bytes, more blocks will be used.

The 22 bytes system information consists of the following items:

- Address of the preceding log record
- Record type, indicating: user record, index record, deleted record, or empty record
- File identifier
- Record length
- Record address on disk
- File-type
- Number of blocks occupied by this record image
- Next free log record

The log records, which each occupy at least one block of 255 bytes, are grouped into segments in the transaction log file. One segment contains blocks of only one transaction.

When a transaction requires logging information to be written to the transaction log file, one segment is assigned to the transaction. The segment is released again at the execution of the next Commit or Rollback.

The segment size is specified during Monitor generation and should allow for logging of the largest transaction in the system. The size required is found by adding the lengths of all the records that are updated during one transaction. If during run-time a transaction can not be logged because the segment is full, the transaction is rolled back and the Return Status indicates "overflow" (3).

9.5.4 Transaction Log Buffers

For transaction logging, one buffer of 256 bytes is reserved in memory.

9.5.5 Transaction Log File

The transaction log file may be defined on disk, in CMOS memory or on the "simulated disk in primary memory". It is created by EDM on the volume specified during Monitor generation, when the application is started. The transaction log file has the following characteristics:

Volume name	:	Specified during Monitor generation (default SYSRES)
File name	:	TLOGFILE
Record size	:	255 bytes
Blocking factor	:	1
Segment size	:	Number of blocks per segment. The total file size is the segment size multiplied by the number of user tasks defined. Default segment size is 30 blocks of 255 bytes.

9.5.6 Initializing the Transaction Log File

The transaction log file is created by EDM during system initialization (IPL), according to the parameters specified during Monitor generation. If there is already a transaction log file on the volume specified, this is deleted and a new file is created.

9.5.7 Function Logging

Function logging means that all functions that result in a modification of the user files (write new records, rewrite updated records, delete records) are logged on a function log file. After a hardware failure, backup copies of the user files can be recovered with the utility Recover EDM File (RCF). This utility reprocesses all functions logged on the function log file up to the last logged Commit, Rollback or Close. The use of the utility is described in the TOSS Utilities Reference manual, module M8A.

The files can be recovered to the consistent state of the last Commit, Rollback or Close. Only the transactions that were being executed when the failure occurred, are lost. If function logging is required for accesses on a file, it must be specified when the file is opened.

9.5.8 Function Log Information

Function log information is stored in the function log file at the execution of the instructions Open, Close, Commit, Rollback, Write, Rewrite and Delete. For Open and Close, a Commit is also logged. When the last user of a file closes the file, an "End of User" log record of 4 bytes is written to the log file in addition to the log record of the Close itself.

The information logged for each function is shown in the diagram below, where:

x = included
– = not included
u = included but not relevant, contents undefined.

Function Log Information	field length (bytes)	Open	Close	Commit	Roll back	End of user
Function code	1	x	x	x	x	x
Function option	1	x	x	x	x	x
Transaction ident	2	x	x	x	x	x
File reference	1	x	x	–	–	–
Filler	1	u	u	–	–	–
File identifier	42	x	–	–	–	–
Relative rec no.	4	–	–	–	–	–
Record length	2	–	–	–	–	–
After image	rec ln	–	–	–	–	–
Before image	rec ln	–	–	–	–	–
Total length in function log file (bytes)		48	6	4	4	4

Function log Information	field length (bytes)	Standard files		
		Write	Rewrite	Delete
Function code	1	x	x	x
Function option	1	x	x	x
Transaction id.	2	x	x	x
File reference	1	x	x	x
Filler	1	u	u	u
File identifier	42	–	–	–
Relative rec. no.	4	x	x	x
Record length	2	x	x	u
After image	rec length	x	x	–
Before image	rec length	–	–	–
Total length in function log file (bytes)		12 + rec ln	12 + rec ln	12

Function log Information	field length (bytes)	Indexed files (EDM files)		
		Write	Rewrite	Delete
Function code	1	x	x	x
Function option	1	x	x	x
Transaction id.	2	x	x	x
File reference	1	x	x	x
Filler	1	u	u	u
File identifier	42	-	-	-
Relative rec. no.	4	u	u	u
Record length	2	x	x	x
After image	rec length	x	x	-
Before image	rec length	-	-	x
Total length in function log file (bytes)		12 + rec ln	12 + rec ln	12 + rec ln

The number of transactions that can be logged on a tape or on a disk file with the default size of 2048 records depends entirely on the record length and on the number and type of functions executed per transaction.

9.5.9 Function Log Buffers

For function logging, two alternating buffers of 256 bytes are reserved in memory.

The function log information is written to the function log file, packed into blocks of 255 bytes. The function log blocks are shared, which means that functions executed by different tasks may be logged in one block. The function information itself is always stored completely in one block, but after-images or before-images of records are split over more blocks if necessary.

A block is written to the function log file on disk or tape when the buffer in memory is full or when a Commit, Rollback or Close function has been logged.

"Overlapping transactions in function log file" may be specified during Monitor generation. This means that when the end of a transaction (Commit, Rollback or Close) has been logged, the remaining part of the block is used for the next log information. The block is written twice to the function log file: first after the Commit, Rollback or Close, and again when the buffer is full.In this way more efficient use is made of the space in the function log file, but the degree of security decreases.

9.5.10 Function Log File

The function log file may reside on tape or disk. This is specified during Monitor generation.

The characteristics of the file are:

Volume name :	specified during Monitor generation (default SYSRES)
File name :	FLOGFILE
Record size :	255 bytes
Blocking factor:	1
File size :	specified during Monitor generation, default 2048 records of 255 bytes.

The blocks are written to the file sequentially. After a system failure, the LRN of the log file is not up to date and can not be used to find the end of the log file. Therefore, each function log block is identified by a certification character in the last two bytes, by which the utility RCF recognizes the valid log information.

9.5.11 Initializing the Function Log File

The way in which the function log file on disk or tape is to be initiated at system start (IPL) is specified during Monitor generation. There are two possibilities:

- The existing function log file is deleted and a new file is created. The existing function log file is also deleted if it is not in a consistent state because of a system failure.

- Function logging continues on the existing function log file.

  There are two situations where this is not possible:

  - There is no existing function log file. A new file is created.

  - The existing function log file is not in a consistent state. This can only occur after I/O errors on the function log files or after a system failure. When this is detected during initialisation the system will halt and the SOP lamps indicate "Log file protected". In that case the recovery utility RCF must be run and then the function log file must be deleted.

9.5.12 Function Log File on Disk

If the function log file is on disk, the volume name must be specified during Monitor generation. The disk must be TOSS formatted.

The function log file is created during system initialization, or the existing file is extended during runtime. The new file or new file extents are formatted if this is specified during Monitor generation. Especially formatting of new file extents during runtime will slow down the system. For that reason, formatting can be excluded, but in that case full safety can not be guaranteed.

If formatting is excluded, more safety can be obtained by initializing the volume with zeroes offline.

At the end of a system session, all files must be closed by all the tasks that have opened them, to leave the user files and the function log file in a consistent state.

The log files should reside on a different volume from that which contains the user files. Otherwise, if a disk becomes unusable because of a hardware failure, the log files can also not be accessed and no recovery can be performed.

Logging will slow down the system because of the extra disk accesses needed. This is especially the case, if the transaction log file and the function log file are on the same volume, or on the same volumes as the user files, because then the read-write head will have to shift continually between the files.

9.5.13 Automatic Enlargement of Function Log File

During Monitor generation it can be specified that the function log file on disk must be automatically enlarged when it is full. If this option is included, the function log file is enlarged by EDM when necessary and the Supplementary Return Status (see chapter 10) indicates "function log file enlarged". The file is enlarged with 10 percent of its size at system start, rounded upward to a multiple of three logical sectors.

9.5.14 Function Log File Full (Disk)

When the function log file is almost full and it is not possible to enlarge the function log file, the message "function log file almost full" (191) is returned in the Supplementary Return Status. It is not possible to enlarge the function log file when the automatic enlarge option has not been included or when the disk volume is full.

This Supplementary Return Status is given when there is still space to close all the files. The files must be closed by all the tasks that have opened them, to leave user- and function log file in a consistent state. After that the application must ask the operator to load another volume (with the same volume name), and then halt. The operator can load the new volume and restart the system. A new function log file is created on the new volume and logging can continue.

If no action is taken on the message "function log file almost full", the function log file gets full. "Function log file full" (246) is returned in the Supplementary Return Status. The current transaction is rolled back if transaction logging is required.

When no transaction logging is provided and the function log file is full, the user files and the function log file are in an inconsistent state. Backup copies of the user files can be recovered to the point of the last Commit, Rollback or Close by running the recovery utility RCF.

9.5.15 Function Log File on Tape

If the function log file is on tape, the tape file code must be specified during Monitor generation.

The function log file on tape is started during system initialization, or logging is continued in the existing file.

At the end of a system session, all files must be closed by all the tasks that opened them, to leave the user files and the function log file in a consistent state, and a tape mark must be written.

9.5.16 Function Log File Full (Tape)

When the end-of-tape mark is read the message "begin/end of tape" is returned in the Supplementary Return Status (197).

The files must be closed by all the tasks that have opened them, to leave user- and function log file in a consistent state, and a tape mark must be written.

It is the user's responsibility to leave enough space after the end-of-tape mark to log the closing of the files. The space needed is 6 bytes for each Close instruction plus 4 extra bytes when the last user closes the file. This is then rounded upward to a multiple of 256. It is recommended to reserve space for one extra block, in case the end of tape mark was detected at the start of a block.

After the files have been closed the operator must load another tape. Function logging is continued on the new tape. It is not necessary to halt and restart the system.

If no action is taken on the message "End of tape" the function log file gets full. "I/O error on function log file"(247) is returned in the Supplementary Return Status. The current transaction is rolled back if transaction logging is provided.

When this happens the user files and the function log file are in an inconsistent state. Backup copies of the user files can be recovered with the function log file to the point of the last Commit, Rollback or Close by running the recovery utility RCF. The current transactions are lost.

9.6 FILE CREATION

E-files are created either by utility Create File (CRF) or by EDM. In both cases, the data file and index file are created at the same time, in one step.
Data records are written to the file after creation, and for every data record the index entry is written to the index file in the correct place.

Up to 10 indexes are allowed for an E-file, and for every index the key may consist of up to 16 key items. However, the total number of key items is limited because the key descriptors together must be stored in the first sector (256 bytes) of the I-file. One key descriptor occupies 8 + (number of items)*4 bytes. From this it follows that it is not possible to have, for example, 10 keys each consisting of more than 4 key items.

When a file is created, the file size is specified as a number of records. This number is rounded upward by EDM to a multiple of three logical sectors and of the block length. The actual size of the file created, in number of records, is returned in the File Parameter block.

E-files can be enlarged until the maximum number of file extents (64 per volume) and file sections (4) has been reached. However, it is not possible to have more file extents on a volume than the number of entries in the VTOC (see Chapter 3).

9.6.1 Creating an E-file with utility CRF

A detailed description of the TOSS utility Create File is found in the TOSS Utilities Reference manual, module M8A.
Note that the questions to describe an index, from "duplicate key" to "key item length", are repeated for every index, and the questions describing a key item are repeated for every key item within one index.

This means that the length of the index descriptor block of an E-file is not fixed but depends on the number of keys and key items.

9.6.2 Creating an E-File by EDM

An E-file is created by EDM by using the Open File instruction. The Open mode must be Output Sequential or Output Direct. The information defining the data file and the indexes must be provided on the File Parameter Block (see chapter 4, File Parameters).

Data records can be written to the file by the application, and the corresponding index entries are inserted in the index file in the correct postion by EDM. The index file will contain 50% free space or even more, at this stage.

9.7 ENLARGING FILES

Automatic enlargement of files by EDM is possible for indexed and standard files.

Files are automatically enlarged when during Write instructions the end of the data file has been reached. A new extent is added to the file, with a size as indicated by the Growth Factor on the File Parameter Block, rounded upward to a file extent length which is a multiple of three logical sectors and of the block size. The Write instructions are executed without the message "End of File" being returned. New file extents and new file sections may be added.

If an E-file is enlarged the new file extent is immediately formatted.

Standard files may be explicitly enlarged by using the Open mode "Extend". When an S-file is enlarged the new file extend is not preformatted. Formatting is done while new records are written to the file and when the file is closed the remaining part is formatted.

Automatic Enlargement of I-Files

When the end of the index file is reached before the current Write order has been completed, the I-file is automatically enlarged with the number of sectors needed to execute the current Write order completely. This is also done if a Growth Factor of zero is specified. The Write request will be completed with bit 8 set in the return code, and the Supplementary Return Status indicating "Index file enlarged" (189).

Automatic Enlargement of Function Log File

Automatic enlargement of the funtion log file will take place as described in section 9.5.13 if this is specified during Monitor generation.

9.8 FILE MAINTENANCE

For EDM files, maintenance in the form of reorganizing the index file is needed less than for indexed files of S-type (in SDM). However, when a data file has been updated by indexed direct instructions, or much extended, the index file will contain free space within the blocks, and consequently more index levels than is necessary. Index blocks may have been split and become empty again.

To avoid overflow of the I-file, or long search times caused by many empty blocks, it is recommended to reorganize the index file with the TOSS utility Reorganize EDM index File (REF). With this utility a load factor of 95 or 75 can be specified, by the indication "static" or "dynamic" use.

It is recommended to specify "static" use if the file will not be updated much, or if it is updated by Indexed Sequential operations. "Dynamic" use is preferred for files on which many new records will still be written by Indexed Direct instructions.

9.9 RETURN INFORMATION

The return information generated by EDM in the Status Word, the Return Status and the Supplementary Return Status is listed in chapter 10, Return Information. The Status Word is obtained in CREDIT by the XSTAT instruction, the Return Status and Supplementary Return Status by the RSTAT instruction.

Chapter 10

RETURN INFORMATION

10.1 INTRODUCTION

After every instruction, the result is reported by a value in the Condition Register, the Status Word and the Return Status. If EDM is used more detailed information is also returned in the Supplementary Return Status.

In addition, the relative key of the record currently accessed is returned in the Control Word. This can be obtained by the application with the GETCW instruction.

The possible values of the condition Register, Status Word, Return Status and Supplementary Return Status are listed in the following sections.

10.2 CONDITION REGISTER

10.2.1 Condition Register

The Condition Register may have one of the following values:

0 = The instruction was successfully completed (but see NOTE).

1 = End of file.
The last used record of the file has been read (LRN reached) by Read Sequential instructions, or the end of the file has been reached by Write Sequential instructions and the Growth Factor is zero.

2 = Error. The instruction was not successfully completed because of sequence errors, illegal parameters or options, or permanent I/O errors.

3 = End of device, or End of medium. This indicates that the application tries to access disk space outside the physical area reserved for the file. This occurs, for example, when the relative record key supplied in an instruction is negative or higher than the total number of records in the file.

The Status Word and in some cases the Return Status and Supplementary Return Status may contain more information.

NOTE: Under EDM it is possible that a fatal error has occurred but the Condition Reigster is zero. Bit 8 is set in the Status Word, and the Return Status and Supplementary Return Status may be read to obtain more information.

10.2.2 Condition Register and Status Word

The relation between the values of the Condition Register and the bits set in the Status Word is as follows:

Condition Register Value	Status Word Bits
2	0 + any other bit, except bit 2 or 3
2	any of the bits 9 - 15
1	3 or 0 + 3
3	2 or 0 + 2
0	in all other cases

NOTE: If bit 8 is set in the Status Word to indicate that more information can be obtained from the Return Status, this is not necessarily indicated by the value of the condition register; this may still have the value zero.

10.3 STATUS WORD

The Status Word is a value set by the TOSS Monitor, to give more information about the result of an instruction. The Status Word is obtained by the Extended Status Transfer instruction (XSTAT).

If bit 8 is set in the Status Word, indicating that a data management rule has been violated, more information may be obtained by reading the Return Status.

Bits set in the Status Word indicate:

No bits set : Successful completion

Bit 0 Request Error
This bit may be set in combination with bits 2, 3, 4, 5 and any of the bits 8 - 15.
An error is detected in the order option, in the instruction or in the parameters in the File Parameter block, or the file access could not be completed because of hardware errors (indicated by bits 13 - 15).

Bit 1 Not used by data management.

Bit 2 End of Medium
The application tries to access space outside the physical file space. Bit 0 is also set.
When this error occurs for an indexed file, file consistency may be lost.

If during a Write instruction the end of the file is reached and a Growth Factor has been specified, but it is not possible to enlarge the file any more, this bit is set and the value 3, Overflow, is set in the Return Status.

It is not possible to enlarge the file any more, if there is no free VTOC entry or no space left on the volume, and no next volume is specified on the File Parameter block where to continue the file, or if the maximum number of file sections and file extents has been reached.
In SDM files can not be automatically enlarged into a new file section (on another volume).

Bit 3 End of File
During a Read Sequential instruction the record indicated by the LRN has been reached. This need not be the physical end of the file, there may be "free" records after it.

During a Write instruction for an indexed file the last free record in the file has been written, and the Growth Factor is zero or further enlargement is not possible.
Bit 0 is also set.

During a write instruction for a standard file the end of the file has been reached and the file is enlarged by the percentage defined in the Growth Factor. No other bits are set.

During Read Indexed Sequential instructions the data record associated with the last entry in the index has been read.

Bit 4 No Data

A Read, Rewrite or Discard request has been issued for a record with status "free".

Bit 5 Key not found

The requested record key is not present in the index file.

Invalid key

The prime key of a record to be written by a Write Indexed Sequential instruction has a value not higher than the prime key of the preceding record (only for EDM).

Bit 6 Duplicate Key

A duplicate key has been detected in an index where duplicate keys are allowed.
During Read Indexed Sequential, a record is read with the same symbolic key as the next record.

Under EDM only:
During Indexed Write or Rewrite functions, a record has been written and one of its keys, for which duplicates are allowed, exists already in the index.

Bit 7 Retries performed

When this bit is set the driver has performed up to the maximum number (disk driver dependent) of retries to perform the I/O. If the I/O was not successful, one of the bits 13, 14 or 15 will be set. If the I/O was successful, one of the other bits may still be set to indicate another error.

Bit 8 Data Management Rule violated

If this bit is set, more information can be found in the Return Status and the Supplementary Return Status.
Bit 8 may be set together with any other bit (except bit 1).

Bit 9 Duplicate Key Error

The relative key specified for a Write Direct, points to a record with status "used".

Under EDM only:
During an Indexed Write or Rewrite instruction a record must be written to the file, and a key for which duplicates are not allowed exists already in the index.

Bit 10 New Volume Loaded

This indicates that a new volume has been loaded after the file had been opened. Close File is the only order which will be accepted after this error message. All the files must be closed and opened again before they can be accessed. For more information on the New Volume Loaded situation, see the description of the disk drivers in the Device Drivers Reference manual, module M5A.

If not carefully handled by the application, the New Volume Loaded situation may result in corrupt files.

Bit 11 Automatic Rollback

In SDM, this indicates that the task has tried to access a record already held exclusive for another task. All records that the requesting task had under exclusive access, are released, to prevent a deadlock situation.

In EDM, a deadlock situation or another error has been detected and automatic Rollback is performed for the requesting task. This means that all records under protected access for the task are released. If transaction logging is done, the before images of the records involved are written back to the files and the currencies are reset to the values at the last Commit (only for EDM).

Bit 12 Incorrect length

The requested length set by the application was not correct. In most cases, this indicates that the requested length is shorter than the record length.

Bit 13 Data error

Transmission unsuccessful because of parity check errors.

Bit 14 Throughput error

Transmission unsuccessful because the system is overloaded or a seek error has occurred on the disk.

Bit 15 Not operable

Bit 15 is set in the case of a disk failure or when an EDM segment has not been loaded.

If one of the status bits 2, 10, 13, 14 or 15 are set after accessing an indexed file, file consistency may be lost (file corrupt), because then it is possible that the data file has been updated but not the index file(s).

10.4 RETURN STATUS

The Return Status is a value set by Data Management to give more information on the result of an instruction.

If bit 8 is set in the Status Word, the Read Status instruction (RSTAT) may be used to obtain the Return Status value in a binary data item.

The errors indicated by the Return Status will normally only occur when an application is being tested. Most of them are not recoverable by the application. If transaction logging is done (only for EDM) the transaction will be rolled back. In all other cases these errors result in corrupt files which have to be recovered from the back-up copies.

If any type of overflow is indicated by the Return Status, this must be solved by generating a new Monitor and/or a reorganisation of the files or of the disk volumes.

The Return Status may have binary values from 0 up to 10, each indicating a number of possible error situations:

1 Memory Overflow

There is not enough work space to open the file.

- In EDM and SDM, this may mean that there is no free File Work Table (FWT) available.
- In SDM it may also mean that there is not enough space to read the master index into memory.

The system must be reorganised and a new Monitor generated. An estimate of the work space needed by EDM is found in Appendix A

2 Input-output error

When a new file is created this may indicate:

- Index file descriptor incorrect
- Error during formatting

When a file is opened, it may indicate:

- One or more file descriptors are not supplied. The file can not be opened.
- File or index file corrupt.
  The file status indicates "corrupt" and the file can not be opened.
  The data file and the index file do not match. This may occur when the files reside on different volumes and the volumes that are on line do not match.

- Index block corrupt
  The index file is corrupt. The files must be recovered from back-up copies.

- Prime key d sturbed
  A Rewrite is issued and the value of the prime key has been changed.

- Transaction log file disturbed.
  The transaction log file can no longer be accessed.

- Function log file full
  Perform system close down, make backup of files if necessary, and restart system.

- Write error on function log file.

- File disturbed during ROLLBACK
  The file is now disturbed. Perform System close down and recover the files from back-up copies.

- File Management detected error

3 Overflow when a file is opened

- File Control Area Table overflow
- File identification table overflow
- Too many files for logging
- Protected record administration table full
- No free Currency buffer (only for SDM)
- Disk overflow
- Transaction log file is full.

- Free space exhausted on the I-file:
  When this message is returned, the I-file is corrupt and can no longer be accessed. A new I-file must be created by the TOSS utility Reorganise EDM file, by copying the D-file into a new E-file by the application. The utility Maintain EDM Indexed File (MEF) may also be used to rebuild the indexes to a data file.

4 Wrong File Parameter

One or more file parameters specified when the file is opened or created, are not correct.

- Wrong file organization
- Wrong record length
- Wrong blocking factor
- Illegal number of key items
- Index descriptor too large
- Illegal device type
- The file organization has been specified as "Indexed", but the index specification is not present in the File Parameter block

- Illegal number of indexes
- Invalid key definition
- Conditional index specified for prime key
- Too many extents

5 Illegal instruction

The instruction issued by the application is not allowed for the file or in this sequence, for example an indexed access is requested for a non-indexed file, or a Write request for a file opened for input only.

- Transaction logging not allowed
  Transaction logging is not allowed when version 3 of EDM is used.

- Illegal statement sequence

6 Illegal function option

- Illegal Open mode
- Illegal Sharability
- Illegal Close option (CLOSE .DROP for a file not opened Exclusive)
- File is opened exclusive by another task
- Illegal type of logging specified

7 Illegal file code

- Illegal index identification
- Illegal file identification
- Illegal file number
- Illegal file code

8 Illegal ECB parameter

- Incorrect File Parameter Block length (too small)

- Incorrect key length specified
  Only for the Posit instruction is it allowed to specify a keylength shorter than the actual key length. Incorrect key length is also returned if the key length specified is greater than the actual keylength.

- Illegal record address
  The record buffer address specified by the application is not valid.

- Incorrect key value
  Illegal characters in symbolic key.

9 Name not found

- File unknown
  The file with the specified name is not found on the volume on-line.

- Volume unknown
  The volume with the specified name is not on-line.

10 EDM error (only for EDM)

- Too many index levels
  The I-file has reached too many levels. If transaction logging is done, the current transaction is rolled back. If not, the E-file is now corrupt and can not be accessed any more. The files must be recovered from back-up copies, and the I-file must be reorganised by the TOSS utility Reorganise EDM File (REF).

- Internal EDM error.
  These will only occur during testing of EDM itself, for example when a special version has been generated.

10.5 SUPPLEMENTARY RETURN STATUS (EDM only)

The Supplementary Return Status is a binary value giving more detailed information about a situation already indicated by the value of the Return Status. The value of the Supplementary Return Status is only significant if EDM is used and the value of the Return Status is not zero.

The Supplementary Return Status is returned together with the Return Status after a Read Status (RSTAT) instruction.

Binary Code	Meaning
000	No Supplementary Status information available
185	Incorrect EDM version. This message is returned when, for example, logging is requested while EDM version 3 is used, or a incorrect non-standard version of EDM has been generated.
186	File identifier table overflow The MAX NUMBER OF OPEN STANDARD FILES and/or the MAX NUMBER OF OPEN INDEXED FILES specified during Monitor generation was too small.
187	Memory overflow. Generate a new Monitor, reserving no more space for block buffers, currency buffers and protection table than strictly necessary.
188	No free buffer available The maximum number of user tasks or the number of index buffers specified during Monitor generation was too small.
189	The index part of an E-file has been enlarged
190	Function log file on disk has been enlarged.
191	Function log file almost full; all opened files must be closed.
192	System Operator's Panel error
193	Tape mark detected
194	Other error on function log tape Run the recovery (RCF), mount a new tape and restart the system.
195	Function log tape not operable
196	Function log tape write protected

197 Begin or end of tape detected on function log tape.
Close all the files, ask the operator to change the tape and continue.

198 Not used

199 No-Wait option not allowed

200 Error detected by File Management
For example, no free FWT available, or too many file extents or file sections. Close all the files, run the recovery (RCF) and restart the system.

201 Not used

202 Illegal close option

203 Conditional primary index not allowed

204 File corrupted during Roll-Back
It is recommended to close all the files, run the recovery (RCF), and restart the system.

205 Log file corrupt
It is recommended to close all the files, if possible, run the recovery (RCF), rename the function log file and restart the system.

206 Duplicates not allowed for prime key

207 Duplicate file descriptor

208 File descriptor not present

209 Invalid key definition

210 Sequential write not allowed

211 Direct write not allowed

212 Incorrect key value
A Write Indexed Sequential request has been issued and the prime key of the record to be written has a lower value than that of the preceding record.

213 Not used

214 Prime key disturbed

215 Not used

216 Not used

217 Incorect key length

218 Record not free

219 Illegal internal index identifier
For example, a file has 2 indexes but internal index identifier 3 has been specified for the instruction.

220 Illegal internal file identifier or file number
This is an internal EDM error. It is recommended to close all the files, run the recovery and restart the system. Describe the situation, take a memory dump and dump the function log file and transaction log file if possible, and send a problem report.

221 Illegal internal index identifier.
The internal index identifier has been set to a higher value than the number of indexes that have been opened for the file, or an internal index identifier has been set for access on a standard file.

222 Illegal function option

223 Posit not allowed

224 Delete not allowed

225 Rewrite not allowed

226 Write not allowed

227 Read not allowed

228 Logging not allowed

229 Illegal number of indexes

230 Index block corrupt
It is recommended to close all files and run the recovery (RCF).

231 File Descriptor area too small
The File Descriptor block length specified for an Open instruction is too short.

232 Specified index not found in file descriptor

233 Incorrect file descriptor parameter

234 Illegal function code.
The function specified is not allowed, or the function code is non-existent.

235 Protection error
Protection Table overflow. Generate a new Monitor with a larger MAXIMUM NUMBER OF USERS, and /or use COMMIT instructions to reduce the number of records held protected per transaction.

236 Exclusive access error

237 Illegal logging parameter

238 Not used

239 Incorrect device type parameter

240 Illegal file organisation parameter

241 Error during move
This is an internal EDM error. It is recommended to run the recovery (RCF) and restart the system. Describe the situation, take a dump and send a problem report.

242 Incorrect File Descriptor Block length

243 Illegal open mode

244 Illegal protection parameter

245 Illegal open mode parameter

246 Function log file full
It is recommended to run the recovery (RCF) and restart the system. This error can be avoided by defining a larger function log file or specify automatic enlargement of the function log file, during Monitor generation.

247 I/O error on function log
It is recommended to run the recovery, use another volume for the function log file and restart the sytem.

248 Index descriptor too large

249 Illegal number of key items

250 Illegal blocking factor

251 Incorrect record length

252 Incorrect file organisation. The file organisation specified is illegal or incorrect.

253 File Control Area Table overflow
The maximum number of open standard or indexed files specified during Monitor generation was too small.

254 File corrupt
The file can not be opened.
It is recommended to run the recovery (RCF).

255 Core space exhausted
EDM has not enough workspace to open the file. The maximum number of open standard or indexed files specified during Monitor generation was too small. Generate a new Monitor, specifying a larger number for one or more of the following SYSGEN parameters: MAX NUMBER OF USER TASKS, NUMBER OF EXTRA INDEX BLOCK BUFFERS, SIZE OF RECORD BUFFER AREA and SIZE OF BLOCK BUFFER AREA. The minimum size of the record and block buffer area required is obtained by taking the sum of the record lengths and the sum of the block lengths of all files open simultaneously.

Appendix A

SPACE REQUIREMENTS

A.1 INTRODUCTION

The approximate amounts of memory space required for the different data management packages and versions are listed in this appendix.

All values are given in bytes, or in K bytes if stated.

By "users" of a file is meant the number of tasks that access the file simultaneously.

To the space required for each package must be added the space required by File Management (see A.4), which is the interface between the data management package and the disk driver. This is also configuration dependent.

A.2 EXTENDED DATA MANAGEMENT

A.2.1 Size of Object Code

- Version 1, Complete EDM package, memory resident	45 K bytes
- Version 2: Complete EDM package, disk resident	
Memory resident part	5 K
Additional segment frame per EDM task	1 K
- EDM subversion 3	39 K

A.2.2 Size of Data Areas

Per task	80+4x no of file code (bytes)
Per opened S-file	160
+ one record buffer	record length
+ one block buffer	block length
+ per user	20
Per opened E-file	570
+ record buffer	record length
+ block buffer	block length
+ per user	90
Index buffer pool	520 x max number of users
Protection table	(17 + max no of users x 6) x 18

An average of 6 protected records per user and an overflow area for 17 records. Each protection entry consists of 18 bytes.

Work areas for transaction logging	700 bytes
Work areas for function logging	700
Fixed work fields	100
Additional if segmentation is used	550
Per EDM task	590

The number of EDM tasks is 2 x number of disk drivers in the system. Minimum 2 EDM tasks.

The total size in memory must not exceed 64 K bytes.

A.3 STANDARD DATA MANAGEMENT

A.3.1 Size of Object Code

Only non-indexed S-files	4 K bytes
Additional for indexed S-files	3 K
Additional when Create, Delete, Extend files option included	0.75 K bytes

A.3.2 Size of Data Areas

Per file code	4 bytes
Per file	100
Per protected record	6
Currency buffer per user	8
Additional per indexed file	4
Per block buffer	block size + 10
Per DM task table	100
Master index pool	user defined

A.4 ABRIDGED DATA MANAGEMENT

A.4.1 Size of Object Code

Object code	500 bytes
Additional when Create, Delete, Extend files option included	1200 bytes

A.4.2 Size of Data Area

Per file code	4 bytes
Per file	ca. 80 bytes
Per ADM task	ca. 100 bytes

A.5 FILE MANAGEMENT

A.5.1 Size of Object Code

I/O requests only	2 K bytes
Additional for Create, Delete, Extend	2 K

A.5.2 Size of Data Areas

Per disk file code	1 byte
Per file	50
Per additional file extent	12