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

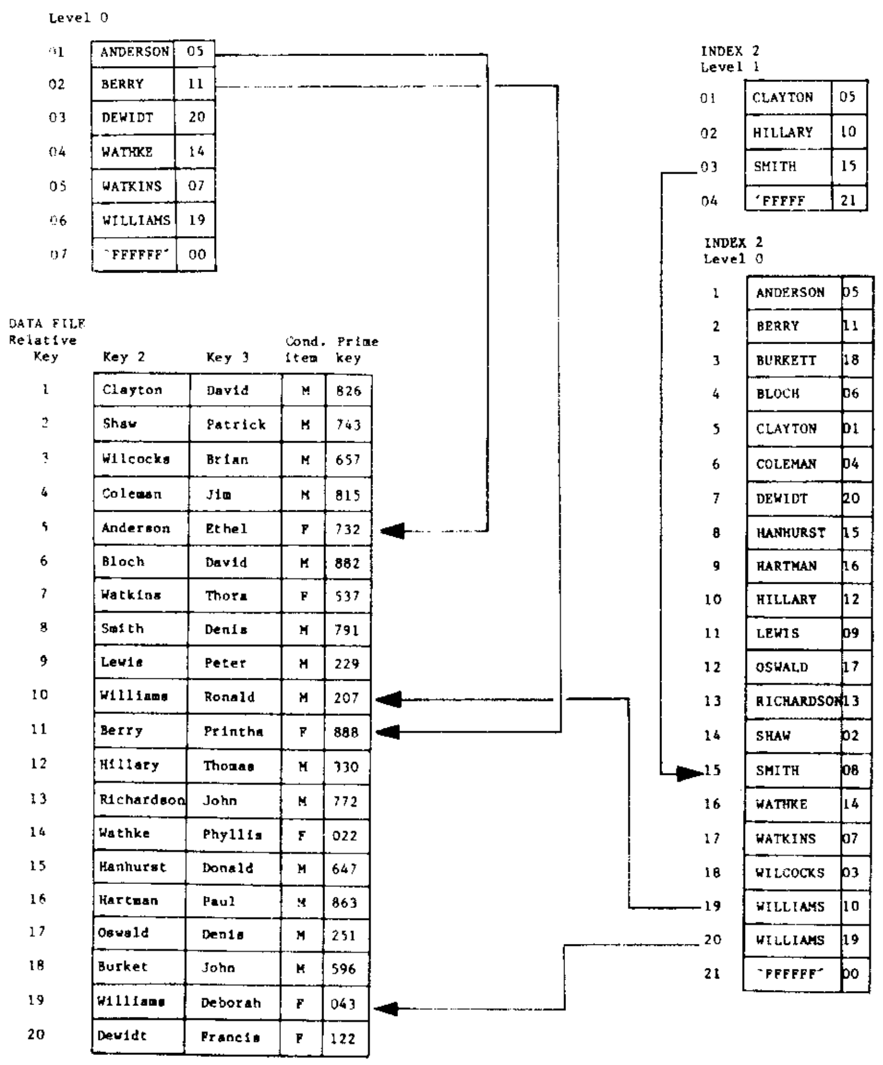
For the file in fig. 9-1, the item in position 22 in the data record is defined as a conditional item for index 4. The condition specified is `Equal', the index only contains entries for the data records for which the conditional item is equal to `F'.

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.