PART 7 TDFM

FUNCTIONAL CHARACTERISTICS

The Transaction-oriented Disc File Manager (TDFM) possesses the following characteristics:

- Simultaneous Access: several user programs may independently access the same record of the same file, without needing to know what other user programs are doing.
- Transactional Access.
- File Structure Independent of the Program: a user program does not need to know such things as the number and locations of keys in a record, or where a record is physically stored, since all this information is stored in a part of the file itself. Also, if the file structure is modified, by adding a new key, for example, none of the existing user programs need to be rewritten to take account of the change.
- Integrity and Uniqueness of Information: since there is only one file, in which each record is unique, the information in the file is always the latest available and is the same for all user programs accessing the file. Furthermore, all control information exists in only one place; the system updates this automatically for each request, and the user need not concern himself with it.
- Distributed Storage: different parts of the file may be held in the first userid's of different DAD's and on different physical disc volumes. Very large files may thus be created.
- Security Features: security copies of a whole TDFM file, or of selected portions, may be made. Changes to records in the file may be recorded in a back-up file and used with the security copies to recreate the file if necessary.

FILE STRUCTURE

A TDFM file is structured as a number of sub-files, as follows:

1	Descriptor sub-file;
n	Index sub-files, one for each key;
m	Data sub-files, each containing part of the data.

The descriptor subfile name may be treated as the name of the TDFM file as a whole, since the descriptor file is the first part of the TDFM file to be accessed. It contains information on all the sub-files in the file, and also some logging information.

Each index sub-file has a tree structure; level zero is the top level, level _n_ (n > 0) records point directly at the data records. The key values in each file are stored in ascending order.

Each data sub-file contains a number of data records. A record may be of any length between 1 and 4095 characters; however, a data record must be long enough to contain all the keys defined for this file. Apart from these restrictions, it is not necessary that all records within a data file should be of the same length.

All the sub-files comprising a TDFM file are consecutive permanent files of type UF.

KEYS

A key is a character string within each record, starting at a defined position and with a defined length. The first key defined for a file is the primary key; all other keys defined for the file are secondary keys. A key may be up to 20 characters long.

If the value of a key must be unique within the TDFM file (if it exists at all), it is called bijective. If several records with the same value for a key may exist within the file, the key is called homonymous. The primary key must always be defined as bijective; the secondary keys may be either bijective or homonymous.

One key - not necessarily the primary key - may be defined as a criterion key. In this case, each data subfile has a value for this key defined for it, in ascending order of the key values and data files; a record with a value for the criterion key will be placed into the first data file whose criterion value is not less than the key value in the record. If no criterion key is defined, the user may specify into which subfile a record is placed; otherwise the first data subfile will be filled, then the second, and so on.

ACCESS METHODS

A TDFM file may be opened by the user in one of three modes:

- Read-only: no updating is allowed;
- Update: updating is allowed;
- Exclusive Access: no other user may access this file until this user releases it. This access mode will only be used occasionally. Updating is allowed.

Once the file is opened, the user may access records in one of four ways:

- Direct, by key name and value;
- Sequential, read next (ascending order of key values);
- Sequential, read previous (descending order of key values);
- Direct, by physical location.

If the key specified with direct access (by key) is homonymous, the system returns the number of records with the same value for the key, as well as the first record with this value. The remaining records may then be accessed in sequence (read next).

If a record is to be updated, it must be attached; this prevents other users from accessing it until the record is detached after updating.

RECOVERY

In order to recover a TDFM file after a system failure, two optional recovery methods may be used, back-up and back-out. There are three levels of security:

- no protection;
- back-up protection;
- both back-up and back-out protection.

Back-up

As each update request is processed, information is written to a back-up file. If the system crashes, the back-up file may be used to update a security copy of the file with all updates made since the copy was taken, up to the time the system failed.

Back-out

A transaction is a set of requests to access a TDFM file. If the system fails in the middle of a transaction, some records are updated and some are not. In order to be able to restore the file to the status before the transaction was started, a back-out record for each update is written to the back-out file; this record allows the system to undo all the updates performed by the incomplete transaction.

If the transaction is successfully completed, all the back-out information relating to this transaction is erased.

SYSTEM STRUCTURE

The Transaction-Oriented Disc File Management system comprises:

- the TDFM file-handling routines, linked into the Monitor at SYSGEN time and called by LKM and the MAS file assignment routines;
- the EDF standard processor, which is activated in the background machine by a standard BCP call. It allocates disc space for the TDFM files, initialises the system data on these files, and provides facilities for copying, restoring, deleting, re-organising and for specifying back-up and back-out files.

FILE STRUCTURE

General

A TDFM file consists of a number of subfiles, which may be held in the first user areas (userid's) on different DAD's and possibly on different physical disc volumes. Each subfile is a permanent consecutive file of type 'UF'. The TDFM file is structured as follows:

1	Descriptor subfile;
n	Index subfiles, one for each key ($1 \leq n \leq 255$);
m	Data subfiles, each holding part of the data ($1 \leq m \leq 32$).

Descriptor Subfile

The descriptor subfile contains two types of information, permanent and logging. The permanent information consists of:
- The date the file was created or last restored;
- the level of protection for this file;
- for each index subfile: the file name, userid and DADname, keyname, key position and length, whether the key is bijective or homonymous and whether it is the criterion key;
- for each data subfile: the file name, userid and DADname and the criterion key value, if any.

The logging information consists of:
- for each subfile: status, i.e. whether overflow has occurred;
- for each data subfile: numbers of records in the file and deleted from it;
- a pointer to the record which caused overflow of an index file (if this has occurred);
- a scratch area, containing "before" images of logging information before update.

The logging information is read into memory when the TDFM file is opened, updated in memory and written back to disc when the file is closed.

Index Subfile

Each key may have a maximum of 32768 different values. All keys within one index file have the same length, which may be between 1 and 20 characters.

Each index subfile contains all the values of a given key, sorted into ascending order and stored in a tree structure. The root of the tree (level 0) consists of only one physical sector on the disc; the leaves of the tree at level $\underline{n}$ ($\underline{n} > 0$) point directly at the data records. All the sectors at each level are chained forward, to speed up sequential access; at the lowest level the sectors are also chained backward.

To compute the size of an index file in sectors, let:

K	=	number of entries per sector;
w	=	sector size (words);
k	=	key-length (words, rounded up if necessary);
n	=	level number (0 to N);
S(n)	=	number of sectors at level $\underline{n}$;
R	=	expected maximum number of records in the file;
T	=	total number of sectors in the file.

Then:

$K = w - 8 / (k + 6)$

$S(N) = R / K$ (rounded up to an integer)

$S(N-1) = S(N) / K$ etc.

$S(0) = 1$ (by definition)

So $T = (1 + S(1) + S(2) + \ldots + S(N)) * 115 / 100 + 2N + 9$

Except at the lowest level, each record in a sector points to a sector one level down; the key value of the record is the highest key value found in the sector. The records within a sector are arranged in ascending order of key values. Thus, when searching for a record with a given key value, only one sector needs to be searched at each level. The system scans the records in a sector until one is found whose key value is not less than the given value.

When a new key record is inserted into an index file, all the subsequent key records are shifted towards the end of the sector and the new key record is inserted in the space thus left. If there is no more room in the sector, a new sector is claimed, the key records are distributed between the two sectors and a new record, pointing to the new sector, is inserted in the sector one level up. This may in turn cause splitting, which can be propagated right up to level 0; if it reaches this level, the index file is said to be in overflow and must be re-organised before further records can be added. Overflow also occurs when no free sector can be claimed.

When a key record is deleted from an index subfile (because the corresponding data record has also been deleted), the space occupied by it is not re-used until the file is re-organised; the record is merely flagged as deleted. This means that a file with frequent updating will, after a short time, contain numerous "dead" entries.

Data Subfile

Each data subfile contains a number of records. Each record may be of any length up to 4095 characters, but it must be long enough to hold all the keys defined for this file. Every record is preceded by a control word; a record may span several sectors on the disc.

A new data record is written into a subfile chosen according to the following rules:

1) if a criterion key has been defined for the file, the new value is compared with the criterion value for each subfile in turn, and the record is inserted in the first subfile found whose criterion value exceeds the new value;
2) if no criterion key has been defined, the record is inserted into the subfile specified by the user in his write command. If he does not specify a subfile, the record is inserted in the first subfile which has room for it.

After modification, a record may be written back so as to re-occupy its original place if:

1) the length has not been changed, and
2) no key field has been altered.

If neither of these conditions is fulfilled the record must be deleted and re-inserted.

When a record is deleted the space it occupied is flagged, but is not re-used until the file is re-organised. This means that a file with frequent updating will, after a short time, contain numerous "dead" entries.

KEYS

A data record consists of a number of _fields_; each field contains some item of data. In a TDFM file, one or more fields may be defined as _key fields_, that is, a record may be accessed by specifying the required key name and value. A key field is of fixed length, between 1 and 20 characters, and starts at a fixed location within the record, specified as an offset from the first character (if the key begins in the first character, the offset is zero).

A key field may be defined as _bijective_, that is, at most one record with a given value for the key may exist in the file. If several records may exist with the same value for the key, the key is said to be _homonymous_ or _multiform_.

The first key field specified when a TDFM file is created is the _primary key_; all other keys are _secondary keys_. The primary key must be bijective; any secondary key may be either bijective or multiform, at the user's option.

For each index subfile a _padding character_ is defined; it may be used to pad key values to the defined length, if necessary. A key consisting entirely of padding characters is the _padding key_; it will always be the lowest key value in the file, and so the padding character must not be higher in the ASCII collating sequence than the lowest character which will be encountered in the key.

When there are several data subfiles in the TDFM file, one key may be defined as a _criterion key_. As each data subfile is defined, a _criterion value_ is supplied; these values must be in ascending order in the data subfile definitions. When a new record is to be inserted, its value of the criterion key is compared with the criterion value of each subfile in turn until a value is found which is not greater than the value in the record; the record is inserted in the corresponding data file. A criterion key need not be bijective.

If a record is added whose value of the primary key is the same as that of an existing record, an error status is returned and the new record is rejected. If the same thing happens with a bijective secondary key, a warning status is returned but the record is inserted into the file.

ACCESS METHODS

A TDFM file may be opened (by a 'Transaction Ready' call - see Chapter 4 of this Part) in one of three modes:
- Read Only: no updating allowed;
- Update: updating is allowed;
- Exclusive Access: updating is allowed. No other user may access the file until this user releases it.

In order to modify a record (add, delete or alter it) the user must first _attach_ it; this prevents any other user from accessing the record until it has been _detached_. The user must attach and detach a record even though he has opened the file for exclusive access. If the file is open in read-only mode, attaching a record is erroneous. Attaching and detaching is done through Monitor calls.

Once the file is opened, the user may access records in one of the following ways:
- Direct, by keyname and value;
- Sequential (either the next or the previous key);
- Direct, by physical location.

Direct Access on Key

The user specifies in his 'Read on Key' call the name and the desired value of the key he wishes to use for access. The record with that value for the key is returned to the user. If the key is homonymous, the first record is returned, together with a count of the remaining number of records with that value. These records may be read by using sequential access; the count of remaining homonymous records is decremented by one for each record read, until it reaches zero.

Sequential Access

The system maintains a pointer into the file; this pointer is initialized by a 'Position' or 'Read on Key' call, and is then maintained through a series of 'Read Next' or 'Read Previous' calls. 'Read Next' returns the record with the next higher key value; 'Read Previous' returns the record with the next lower key value. Within a homonymous chain of records, the next (or previous) record is returned along with the count of records remaining in the chain.

With the 'Position' call, no error is returned if the record with the specified key value does not exist; with the 'Read on Key' call, a warning status is returned under these conditions, but the pointer is correctly set up and may be used by 'Read Next' or 'Read Previous' calls.

In order to set the pointer to the start of the file, so that a sequence of 'Read Next' calls will read every record, the 'Position' call should be used with the padding key value. A 'Read Previous' call will cause 'Beginning of File' status to be returned.

In order to set the pointer to the end of the file, so that a sequence of 'Read Previous' calls will read every record, the 'Position' call should be used with the highest possible key value. A 'Read Next' call will cause 'End of File' status to be returned.

Simultaneous modification of the file by other users is taken into account, e.g. if another user creates a record immediately after the one just obtained by 'Read Next', then the new record will be the record obtained by the next 'Read Next' call.

Direct Physical Access

The user may access a record directly within the data file by giving the following information:
- data subfile number;
- sector number within subfile;
- displacement of the start of the record within the sector.

The user may compute these coordinates himself, or obtain them from the information returned after a previous 'Read' call.

The data subfile number identifies the subfiles in the order they were defined, starting from zero.

GENERAL

The EDF Processor may be loaded and started as a background (low priority) program if required, and is controlled by a set of commands submitted by the user. It may be terminated either normally or abnormally (when something has gone wrong).

COMMAND SYNTAX

A command consists of a keyword, three or four characters long, optionally followed by a space and one or more parameters. The keyword begins in the first character position of the input line. The second and subsequent parameters are each separated from the one preceding by one comma; if the command extends over two or more lines, the last parameter on a line is followed by a semi-colon and the next parameter begins in the first character position of the next line of input.

All commands input through the operator's typewriter are terminated by (CR)(LF).

PARAMETER SYNTAX

All parameters are keyword parameters, and may be entered in any desired order. In the vast majority of cases, the keyword is followed by an equals sign and a value.

The following conventions are used in syntax descriptions:

- Curly brackets {thus} enclose a list of syntactic items separated by vertical bars (|) meaning "or"; one of the syntactic items must be chosen.

- Square brackets [thus] enclose an optional syntactic item. If it is followed by three full stops (...) possible repetition is indicated.

- Angle brackets <thus> enclose a syntactic item whose nature is indicated by the word(s) enclosed.

The following syntax items are used throughout the descriptions:

```
<dadfc>    ::=  <filecode>
<filecode>::=  /<hexadigit>[<hexadigit>]
<userid>   ::=  <ASCII char>[<ASCII char>]... (max 8 chars)
<filename>::=  <ASCII char>[<ASCII char>]... (max 6 chars)
<keyname>  ::=  <filename>
<integer>  ::=  <digit>[<digit>]...
```

Where two or more syntax items of the same nature appear in a description, the appearances are distinguished by a decimal digit at the end of each item; for example:

```
IDAD=<dadfc1>,ODAD=<dadfc2>
```

The following syntax items appear only in the descriptions of error messages:

```
    <dadname> ::=  <filename>
    <ss>      ::=  <hexadigit><hexadigit>
    <ssss>    ::=  <ss><ss>
```

CONCURRENT ACCESS

The execution of many of these commands makes physical alterations within a TDFM file; in these cases the user should ensure that there is no other activity within the relevant file due to other users, and he should delete all the filecodes in all machines which refer to this TDFM file. This ensures that the first access after the EDF session creates a new file table in memory.

If the user attempts to modify a file while other activity is still in progress, the following error messages are output and the command is aborted:

```
THE FILE IS ALREADY ASSIGNED IN THE SYSTEM
USE PFC COMMAND (FCL) TO KNOW THE F. CODES ASSIGNED TO IT
```

The commands for which this is necessary are:

FILE	KEY	DATA	LOAD	IDRG	MTDK	NKEY	DLKE
REST	SAVE	INSE	RBUP	SPRO	DKMT	NDAT	

ERROR HANDLING

If an error occurs while executing an EDF command in batch mode, the command is aborted and the EDF processor exits. The user must then correct the input and restart the EDF Processor. If an error occurs in interactive mode, the user has the choice of either retrying the command or aborting the EDF Processor.

DISC SPACE ALLOCATION

The EDF Processor has three commands to allocate space on disc for a TDFM file, and to initialise the system information; the FILE, KEY and DATA commands. The numbers of KEY and DATA commands are specified in the FILE command; there must be at least one of each. Up to 255 KEY commands and 32 DATA commands may be entered for each FILE command. All the KEY commands precede all the DATA commands.

If the name of a subfile created by one of these commands is the same as that of a file already existing under the same userid and on the same DAD, the previous file is destroyed before the new file is created.

If the user wishes to add a new key or data file to an already existing TDFM file, he may use the NKEY or NDAT commands, respectively. The DLKE command exists to delete a key file from a TDFM file.

FILE COMMAND

```
Syntax:  FILE FNAM=<filename>,USID=<userid>,DAD=<dadfc>;
         NKEY=<integer1>,NDAT=<integer2>,MREC=<integer3>[,SECU={BU | FULL}]
```

This command allocates space on the DAD defined by <dadfc> for the Descriptor Subfile <filename> within user area <userid>. The number of index files is given by <integer1>, the number of data files by <integer2> and the expected maximum number of records by <integer3>.

If the parameter SECU=BU is present, the file is protected by back-up; if the parameter SECU=FULL is present, the file is protected by both back-up and back-out. If the SECU parameter is omitted, no protection is applied to the file. The security level applies to all subfiles in the TDFM file.

Errors

```
<keyword> PARAM MUST BE CHARACTER STRING
<keyword> PARAM MUST BE NUMERIC
-   where <keyword> identifies the wrong parameter.
<value> PARAM FOR SECU NOT ACCEPTABLE
-   <value> is neither BU nor FULL
FILENAME TOO LONG. TRUNCATED
<ss> FC NOT FOR DAD
-   <ss> is the status from a 'get device description' request
<ss> ASSIGN NOT DONE
<ssss> I/O ERROR DETECTED
PROCESSING OF COMMAND ABORTED
```

KEY COMMAND

```
Syntax:  KEY DAD=<dadfc>,USID=<userid>,KNAM=<keyname>,KPOS=<integer1>;
         KLGT=<integer2>,KPAD=<padchar>[,FNAM=<filename>][,BIJ][,DISP]

         <padchar> ::=  /<hexadigit><hexadigit>
```

This command allocates space on the DAD defined by <dadfc>, within user area <userid>, for an Index Subfile with name specified by <filename>. If the FNAM parameter is omitted, the file name is assumed to be the same as <keyname>. The value of <integer1> gives the start of the keyfield, as a displacement from the start of the record; zero means that the keyfield begins at the first character of the record. The value of <integer2> gives the length of the keyfield, from 1 to 20 characters.

The <padchar> has two uses: it may be used to pad key values, supplied in order to access the file, to the defined key length, if necessary; and a key consisting entirely of <padchar>'s will have a value less than that of any other value in the file. Thus the ASCII character defined as a <padchar> should not be higher than the lowest actual character expected to be used in the keyfield.

The parameter BIJ, if present, specifies that this key is bijective. The first key specified is the primary key, which must always be bijective; any of the remaining (secondary) keys may be specified as bijective, if desired.

The parameter DISP, if present, specifies that this is the criterion key for this file. At most one KEY command may have this parameter specified. (The criterion key is also known as the dispatching key.)

Errors

```
FIRST KEY MUST BE BIJECTIVE
MAX. KEY LENGTH IS 20
<keyword> PARAM MUST BE CHAR. STRING
<keyword> PARAM MUST BE NUMERIC
<filename> FILENAME TOO LONG. TRUNCATED
<keyname> KEYNAME TOO LONG. TRUNCATED
DISPATCHING KEY PREVIOUSLY DEFINED
<ss> <userid> <filename> KEEP FILE FAILED
<ss> ASSIGN NOT DONE
<ss> FC NOT FOR DAD
<ssss> I/O ERROR DETECTED
PROCESSING OF COMMAND ABORTED
```

DATA COMMAND

```
Syntax:  DATA FNAM=<filename>,DAD=<dadfc>,USID=<userid>;
         NBGR=<integer1>[,CRIK=<keyval>]

         <keyval>  ::=  <integer2>{A | D | H}<value>$$[<keyval>]
```

This command allocates space on the DAD defined by <dadfc>, within user area <userid>, for the Data Subfile named <filename>. It also inserts information into the Descriptor Subfile. <integer1> specifies the number of granules to be allocated to the file.

If the DISP parameter was specified for one KEY command in this sequence, the CRIK parameter must be specified for every DATA command in the sequence. If the DISP parameter was not specified, neither should the CRIK parameter be specified for any DATA command.

The key value defined in the CRIK parameter is built up from one or more subfields, each with length defined by <integer2>. The value is as follows:

A	<integer2> ASCII characters
D	a decimal number, left-padded with zeroes if necessary to a length of <integer2>
H	a hexadecimal number (two hexa digits per byte) left-padded with zeroes if necessary.

The total length of the value so built must equal that of the key defined as the criterion key. Successive DATA commands must have increasing key values; the key value for the last DATA command should be at least as great as the highest key value expected for the file.

Errors

```
<keyword> MUST BE CHAR. STRING
<keyword> MUST BE NUMERIC
<filename> FILENAME TOO LONG. TRUNCATED
CRITERION KEY NOT EXPECTED
LENGTH OF CRITERION KEY IS WRONG. SEE KEY COMMAND
KEY VALUE < PREVIOUS OR PADDING KEY VALUE
KEY VALUE = PREVIOUS OR PADDING KEY VALUE
CRITERION KEY MISSING
<ss> <userid> <filename> KEEP FILE FAILED
<ss> ASSIGN NOT DONE
<ss> FC NOT FOR DAD
<ssss> I/O ERROR DETECTED
PROCESSING OF COMMAND ABORTED
```

NKEY COMMAND

```
Syntax:  NKEY FNAM=<filename>,USID=<userid>,DAD=<dadfc>[,XNAM=<new name>]
              ,XUSI=<userid2>,XDAD=<dadfc2>,KNAM=<keyname>,KPOS=<int>
              ,KLGT=<int2>,KPAD=<padchar>[,FREE=<int3>][,BIJ]
```

This command defines a new access key for an existing TDFM file, and generates the corresponding index file. The new key cannot be either the primary key or a dispatching (criterion) key.

For a discussion of the parameters see the KEY command. The default for the XNAM parameter is the KNAM value.

It is strongly recommended to make a copy of the file before issuong this command, in case of disc errors during execution. If the file is protected, the copies must be updated after creating the new key.

The filecodes used during execution are:

```
    /20, /21, ... (/20 + <number of data subfiles> - 1)
```

Errors

```
UNKNOWN DATA OR INDEX FILE DAD NAME: .....
BAD ASSIGN DATA FILE STATUS = <ssss>
2 KEYS EQU IN BIJ. IND. .....
INDEX OVERFLOW LEVEL 0
BAD ASSIGN INDEX FILE STATUS = <ssss>
INDEX FILE OVERFLOW: <ssss>
EMPTY DATA SUBFILES: ......
WORK FILECODES /D6 OR /D7 NOT ASSIGNED
INVALID PARAMETER = ......
BAD ASSIGN DESCRIPTOR FILE, STATUS = <ssss>
ERROR GET INFO ON DAD, STATUS = <ssss>
DAD NOT ASSIGNED TO A DAD
TOO MANY RECORDS DECLARED: ....
ONLY ONE INDEX LEVEL; PARAM MREC TOO SMALL
I/O ERROR WHEN DELETING VERSIONS OF INDEX FILE
BAD ASSIGN NEW DESCRIPTOR, STATUS = <ssss>
KEEP FILE <name> ERROR STATUS = <ssss>
KNAM ALREADY EXISTING
THE FILE IS ALREADY ASSIGNED IN THE SYSTEM
 USE PFC COMMAND (FCL) TO KNOW THE F. CODES ASSIGNED TO IT
```

Warning and Informative Messages

```
WARNING: OLD VERSION OF DESCRIPTOR NOT DELETED
<nn>% FREE SPACE IN INDEX FILE
```

NDAT COMMAND

```
Syntax:  NDAT FNAM=<name>,USID=<userid1>,DAD=<dadfc1>,DNAM=<name2>
              ,DDAD=<dadfc2>,DUSI=<userid2>,NBGR=<int>[,CRIK=<keyval>]
```

For the syntax of the CRIK parameter see the DATA command.

This command defines a new data subfile for an existing TDFM file.

If the existing TDFM file has a criterion key defined for it, the CRIK parameter is mandatory in this command; the criterion key value supplied must be greater than that of any existing data subfile. If no criterion key is defined for the TDFM file, the CRIK parameter must not be supplied.

The TDFM file date is updated.

Errors

```
BAD ASSIGN DESCRIPTOR FILE, STATUS = <ssss>
ERROR GET INFO ON DAD STATUS = <ssss>
CRITERION KEY NOT DEFINED
WRONG CRITERION KEY LENGTH
USE A HIGHER CRIK VALUE
CRITERION KEY EXPECTED
DDAD NOT ASSIGNED TO A DAD
DATA FILE NAME ALREADY EXISTS
I/O ERROR WHEN DELETING VERSIONS OF DATA FILES
BAD ASSIGN NEW DESCRIPTOR, STATUS = <ssss>
KEEP FILE <name> ERROR, STATUS = <ssss>
INVALID PARAM = ....
BAD ASSIGN DATA FILE, STATUS = <ssss>
```

Warning Message

```
WARNING: OLD VERSION OF DESCRIPTOR NOT DELETED
```

DLKE COMMAND

Syntax: DLKE FNAM=<name>,USID=<userid>,DAD=<dadfc>,KNAM=<keyname>

This command is used to remove an access key and its corresponding index sub-file from a TDFM file. Neither the primary key nor the criterion (dispatching) key may be deleted in this way.

It is strongly recommended to make a copy of the file before issuing this command. If the file is protected, existing copies must be updated after deletion of the key.

The TDFM file date is updated.

Errors

```
INVALID PARAMETER = ....
BAD ASSIGN DESCRIPTOR FILE, STATUS = <ssss>
ERROR GET INFO ON DAD STATUS = <ssss>
UNKNOWN KEY NAME
BAD ASSIGN NEW DESCRIPTOR, STATUS = <ssss>
KEEP FILE <name> ERROR, STATUS = <ssss>
THE FILE IS ALREADY ASSIGNED IN THE SYSTEM
 USE PFC COMMAND (FCL) TO KNOW THE F. CODES ASSIGNED TO IT
```

Warning Message

```
WARNING: OLD VERSION OF DESCRIPTOR NOT DELETED
```

LOADING and UNLOADING

The EDF Processor has two commands for loading a TDFM file from, or unloading it to, a sequential file; these are the LOAD and UNLD commands, respectively. Together, these commands are useful for reorganising a file, or providing a tool for recovery; the LOAD command may also be used for initial loading of a file, for which it is faster, and gives a cleaner index structure, than a loop of Monitor calls to write new records.

The sequential file may be:

a) a DFM disc file, held on one volume;
b) held on magnetic tape or cassette. Both multi-file volumes and multi-volume files are handled; end-of-volume within a file is signalled by the sequence :EOV (TM) (TM).
c) (for input only) punched cards;
d) (for input only) paper tape, terminated by :EOF.

The sequential file records are handled on a multiblock basis, i.e. one record on the TDFM file may be split into, or built up from, several records on the sequential file. For an output sequential file the blocking factor is always fixed. For an input sequential file the blocking factor may be fixed or variable; in the latter case, the last input record used to build a TDFM record is terminated by two ASCII characters not otherwise used together within a record.

LOAD COMMAND

```
Syntax:  LOAD ONAM=<filename>,ODAD=<dadfc>[,OUSI=<userid>],ICOD=<filecode>;
         TYPE={CONT | CRIK | <integer1> | SAME}
         [,{N=<integer2> | SEP=<separator>}]
         [,IGEN={YES | NO}][,FREE=<integer3>]

         <separator>     ::=  '<ASCII char><ASCII char>'
```

This command uses the records in the sequential file specified by <filecode> to build data records in the TDFM file whose Descriptor file is located on DAD <dadfc>, within the user area <userid> and with name <filename>. If the userid is omitted, the :JOB USID is assumed.

The TYPE parameter specifies where TDFM records are to be stored:

CONT	specifies that the first data subfile is to be filled, then the second, and so on;
CRIK	specifies that the criterion key is to be used to decide into which data subfile this record is to be inserted. It may only be used if a criterion key has been established for this TDFM file.
<integer1>	specifies that all records are to be inserted into the <integer1>th data subfile.
SAME	specifies that the records are to be partitioned between the data subfiles in the same way as they were stored on a TDFM file, of which this input file is a copy produced by UNLD.

The N and SEP parameters are mutually exclusive. The N parameter specifies a fixed blocking factor of <integer2> input records to build one TDFM record; the SEP parameter specifies the two ASCII characters used to terminate the last input record of a set used to build one TDFM record. If neither parameter is present, N=1 is assumed.

The IGEN parameter specifies whether the index files are to be updated with the keys of the loaded records; if omitted, NO is assumed. Not updating the index files allows the user to load a number of input files, letting the indexes be updated once when all data has been input, and thus saving time (index generation is a very slow process). However, the index files will not be updated until IGEN=YES is entered, and consequently they will not be consistent with the data. For this reason, and because there is no error recovery, it is advisable to make a copy of the TDFM file before starting to load new records, if the file already contains some data.

The FREE parameter is only significant if IGEN=YES; <integer3> specifies the number of free entries to be kept in each sector of an index, as a percentage of the total number of entries per sector. This may be done in order to avoid early overflow of the index, or splitting of index blocks, if many records are to be added later by write calls. If the parameter is absent, the default value is 0%.

Filecodes

The LOAD command uses filecodes /20 to /3F for its processing; these codes must therefore not be in use elsewhere while the LOAD command is still in operation, either as the value of the ICOD parameter or in any user program.

If IGEN=YES, two temporary work files with filecodes /D6 and /D7 must have been previously assigned. If these files are consecutive, their minimum size may be calculated as follows:

No. of sectors = R * (K + 3) / (S - 1) + 10%

where	R	is the total number of records in the TDFM file, after loading;
	K	is the keylength, in words, of the longest key;
	S	is the size of a sector, in words, in the work file.

Errors

```
2 CHAR FOR 'SEP' PARAMETER
N OR SEP IS SUPERFLUOUS
INVALID PARAM = <keyword>
WORK FILE /D6 OR /D7 NOT ASSIGNED
ERROR GET INF. ON DAD STATUS = <ssss>
DYNAMIC AREA OVERFLOW
ERR. DATA FILE NB. IN TYPE PARAM
TOO MANY SUBFILES (32 MAXIMUM)
DAD NAME: <dadname> UNKNOWN
BAD ASSIGN DATA FILE STATUS = <ssss>
ERROR ICOD, STATUS = <ssss>
BAD ASSIGN DESCRIPTOR FILE, STATUS = <ssss>
INPUT DEVICE: <dev> NOT COMPATIBLE WITH SEP. PARAM
INPUT DEVICE: <dev> REFUSED
OVERFLOW IN DATA FILE
INPUT RECORD TOO LONG
PAUSE ERROR
END OF TAPE (INPUT FILE)
BEGINNING OF TAPE
INPUT FILE = INCORRECT LENGTH
INPUT FILE = DATA FAULT
EOS INPUT FILE, NEED 1 MORE DATA FILE
DATA FILE <filename> OVERFLOW
INPUT FILE I/O ERROR, STATUS = <ssss>
PARAM VALUE OF N MUST BE SMALLER
-   a record has been found with length less than the value of N
UNKNOWN (DATA OR INDEX FILE) DAD NAME: <dadname>
2 KEYS EQUAL IN BIJ. IND. <filename>
INDEX OVERFLOW LEVEL 0
BAD ASSIGN INDEX FILE STATUS = <ssss>
INDEX FILE OVERFLOW: <filename>
```

Report Messages

```
END OF VOLUME, LOAD NEXT TAPE AND RESTART
-   end-of-volume has been recognized on the current input magnetic tape or
    cassette. The next volume in sequence should be loaded and the EDF
    Processor restarted.
**<integer>% OF FREE SPACE IN INDEX FILE <filename>
-   the named index file has been updated; it contains the stated percentage of
    free entries.
```

UNLD COMMAND

Syntax: UNLD FNAM=<filename>,DAD=<dadfc>,USID=<userid>,OCOD=<filecode>
[,DATA=<integer1>][,N=<integer2>]

This command outputs data records to a sequential file identified by <filecode> from the TDFM file whose Descriptor subfile named <filename> is held within the used area <userid> on DAD <dadfc>. If the DATA parameter is present, records are only output from the <integer1>th data subfile, otherwise from all data subfiles.

The N parameter, if present, gives the blocking factor, i.e. a single TDFM record will be split into <integer2> records on the sequential file. If the parameter is absent, N=1 is assumed. <integer2> must satisfy both the following conditions:

1) it must not be greater than the minimum logical record length, and
2) <integer2> times the output physical record length must not be less than the maximum logical record length.

The output sequential file may be used as input to the LOAD command, using any of its four loading types.

Errors

INVALID PARAM = <keyword>
BAD ASSIGN DESCRIPTOR FILE STATUS = <ssss>
ERROR GET INF. ON DAD STATUS = <ssss>
ERROR OCOD, GET INF. STATUS = <ssss>
OUTPUT DEVICE <dev> NOT ALLOWED
DADNAME: <dadname> OF DATA FILE: <filename> IS UNKNOWN
BAD ASSIGN DATA FILE STATUS = <ssss>
OUTPUT DFM FILE IS FULL
ERROR BACKWARD ON TAPE STATUS = <ssss>
WRITE ERROR STATUS = <ssss>
N PARAM MUST BE SMALLER
- <integer2> does not satisfy condition 1).
N PARAM VALUE NOT VALID
- <integer2> does not satisfy condition 2).

Report Messages

THE TAPE IS NEARLY FULL, TAKE CARE!
- this tape should not be used to output a new data file.
END OF VOLUME, LOAD NEXT TAPE AND RESTART
- end-of-volume has been recognized on the current output magnetic tape/cassette. The next volume in sequence should be loaded and the EDF Processor restarted.

FILE REORGANISATION

When either an index entry or a data record is deleted and then re-entered, the space used by the deleted record is not automatically recovered for re-use. Thus, after many amendments have been made within a file, it will require reorganisation in order to recover the lost space.

There are two levels of reorganisation possible:
a) one or more index files need to be reorganised, and
b) part or all of the data needs to be reorganised.

Index Reorganisation

The IDRG command, described overleaf, is available for the reorganisation of one index file. It may be repeated for each index file needing to be reorganized.

Data Reorganisation

The following sequence of commands will reorganise not only the data but all the indexes as well:
1) unload the TDFM file to disc or magnetic tape;
2) delete the TDFM file;
3) recreate the TDFM file;
4) reload the TDFM file and regenerate the indexes.

IDRG COMMAND

Syntax: IDRG FNAM=<filename>,FDAD=<dadfc1>,USID=<userid>;
KNAM=<keyname>,KDAD=<dadfc2>[,FREE=<integer>]

This command reorganises the given index file, by re-using the space occupied by deleted entries, freeing sectors used for splitting and no longer required, and optionally leaving a percentage of each sector unused (to allow for further updating without early splitting or overflow).

The <filename>, <dadfc1> and <userid> are those of the Descriptor subfile of the TDFM file to which the index belongs; <dadfc2> specifies the DAD on which the Index subfile for key <keyname> is located.

The FREE parameter, if present, specifies the number of free entries to be kept in each sector, as a percentage of the total number of entries per sector. If the parameter is absent, the default value is 0%.

Since this command works directly on an index file, it is recommended to make a copy of the index at least - possibly the whole TDFM file - before using this command.

Errors

```
DEFINED KEY NOT FOUND
<keyword> PARAM MUST BE NUMERIC
<keyword> PARAM MUST BE CHAR. STRING
<filename> FILENAME TOO LONG. TRUNCATED
<keyname> KEYNAME TOO LONG. TRUNCATED
<ss> FC NOT FOR DAD
<ss> ASSIGN NOT DONE
<ssss> I/O ERROR DETECTED
PROCESSING OF COMMAND ABORTED
```

Report Messages

```
<integer>% OF FREE SPACE HAS BEEN GIVEN
```
- this message is issued when the IDRG command has completed its processing.

TDFM FILE HOUSEKEEPING

It is important to be able to make security copies of TDFM files, particularly when they are protected by the back-up facilities. The EDF Processor provides facilities to copy, restore and delete TDFM files, either at the subfile level or entire files. (Subfiles may be deleted individually only through the use of the Librarian.)

When copying, the date and time of the copy process are written into the copied file; the System Manager must therefore ensure that the date and time held inside the machine are correct, and that the Real Time Clock is running, before making a copy. When restoring, the date and time in the restored file are those held in the file from which the restore is being made. This is important, as it enables the back-up recovery process to update only those records which were altered since the copy was made.

The COPY and REPL commands, working on individual subfiles, operate even when the Descriptor subfile has been corrupted. The SAVE, REST and DEL commands, working on whole files, fail unless the Descriptor subfile contents are still valid.

COPY COMMAND
REPL COMMAND

```
Syntax:  {COPY | REPL} FNAM=<filename>,IDAD=<dadfc1>,IUSI=<userid1>;
                       ODAD=<dadfc2>,OUSI=<userid2>
```

The COPY and REPL commands copy one subfile, with name <filename>, of a TDFM file held in user area <userid1> on DAD <dadfc1> to user area <userid2> on DAD <dadfc2>. The COPY command updates the date and time, held in the Descriptor subfile of the copy, to those when the copy is made; the REPL command does not update the date and time. Otherwise the two commands are identical in their operation.

An entire TDFM file should be copied (or replaced) by a series of commands. Subfiles should be copied in the following order:
- the Descriptor subfile;
- all Index subfiles (in any order);
- all Data subfiles (in any order).

A COPY should only be made between two 'runs' using this file. See below under 'Recovery Procedures' for more details.

None of the subfiles to be created by either of these commands should already exist in the target DAD and user area. If necessary, the Librarian should be used to delete previous versions before the COPY or REPL command is executed.

The sector sizes of both DAD's must be equal.

Errors

```
OLD DAD AND USERID = NEW DAD AND USERID
<keyword> PARAM MUST BE NUMERIC
<keyword> PARAM MUST BE CHAR. STRING
<filename> FILENAME TOO LONG. TRUNCATED
<ss> FC NOT FOR DAD
<ss> ASSIGN NOT DONE
<ssss> I/O ERROR DETECTED
OLD AND NEW DAD MUST HAVE EQUAL SECTOR LENGTHS
FILE DESCRIPTOR REFERS TO <dadname> <userid> <filename>. ERROR
USERID NOT CATALOGUED ON ODAD
FILE ALREADY CATALOGUED ON ODAD
ERROR WHILE SEARCHING IN DIRECTORY
CONVERSION ERROR (DEBI)
<ss> <userid> <filename> KEEP FILE FAILED
<dadname> <userid> <filename> INDEX FILE NOT FOUND
<dadname> <userid> <filename> DATA FILE NOT FOUND
PROCESSING OF COMMAND ABORTED
```

SAVE COMMAND
REST COMMAND

```
Syntax:  {SAVE | REST} FNAM=<filename>,IDAD=<dadfc1>,IUSI=<userid1>;
                       ODAD=<dadfc2>,OUSI=<userid2>
```

The SAVE and REST commands copy an entire TDFM file, with name <filename>, held in user area <userid1> on DAD <dadfc1> into a previously existing TDFM file with the same filename held in user area <userid2> on DAD <dadfc2>. The SAVE command updates the date and time, held in the Descriptor subfile, to those when the copy is made; the REST command does not update the date and time. Otherwise there is no difference between the two commands.

These commands only work if the contents of the Descriptor subfiles of both the source and target files are still valid. If the target Descriptor subfile has been corrupted, the user has two possible courses of action:

1) use the Librarian to delete all subfiles of the target file, then use a sequence of COPY or REPL commands to copy each subfile in turn, or
2) recreate an empty target file using the FILE, KEY and DATA commands (which automatically delete the previous versions), then use the SAVE or REST command to copy all the subfiles.

In the rare case that a source Descriptor subfile is corrupted, the user should recreate an empty source file using the FILE, KEY and DATA commands and reload it from a sequential file produced by the UNLD command.

Restoring a TDFM file with back-up security should only be done before back-up recovery is initiated.

Errors

```
OLD DAD FC AND USERID = NEW DAD FC AND USERID
<keyword> PARAM MUST BE NUMERIC
<keyword> PARAM MUST BE CHAR. STRING
<filename> FILENAME TOO LONG. TRUNCATED
<ss> ASSIGN NOT DONE
<ss> FC NOT FOR DAD
<ssss> I/O ERROR DETECTED
OLD AND NEW DAD MUST HAVE EQUAL SECTOR LENGTHS
FILE DESCRIPTOR REFERS TO <dadname> <userid> <filename>. ERROR
USERID NOT CATALOGUED ON ODAD
FILE DESCRIPTOR NOT VALID. USE COPY COMMANDS
ERROR WHILE SEARCHING IN DIRECTORY
<ss> <userid> <filename> KEEP FILE FAILED
<ss> <dadname> <userid> <filename> INDEX FILE NOT DELETED
<dadname> <userid> <filename> INDEX FILE NOT FOUND
<ss> <dadname> <userid> <filename> DATA FILE NOT DELETED
<dadname> <userid> <filename> DATA FILE NOT FOUND
CONVERSION ERROR (DEBI)
<dadname> <userid> <filename> NO DAD FILE CODE FOR THIS INDEX FILE
<dadname> <userid> <filename> NO DAD FILE CODE FOR THIS DATA FILE
PROCESSING OF COMMAND ABORTED
```

DEL COMMAND

Syntax: DEL FNAM=<filename>,DAD=<dadfc>,USID=<userid>

The DEL command deletes all the subfiles of a TDFM file, whose Descriptor subfile is held on DAD <dadfc> in user area <userid> under the name <filename>. The command only works if the Descriptor subfile contents are still valid; if not, the Librarian must be used to delete all the subfiles, one by one.

Errors

```
DAD PARAM MUST BE NUMERIC
USID PARAM MUST BE CHAR. STRING
FNAM PARAM MUST BE CHAR. STRING
<filename> FILENAME TOO LONG. TRUNCATED
<ss> ASSIGN NOT DONE
<ss> FC NOT FOR DAD
<ssss>I/O ERROR DETECTED
FILE DESCRIPTOR REFERS TO <dadname> <userid> <filename>. ERROR
<ss> <dadname> <userid> <filename> INDEX FILE NOT DELETED
<ss> <dadname> <userid> <filename> DATA FILE NOT DELETED
<ss> <dadname> <userid> <filename> FILE DESCRIPTOR NOT DELETED
<dadname> <userid> <filename> NO DAD FILE CODE FOR THIS INDEX FILE
<dadname> <userid> <filename> NO DAD FILE CODE FOR THIS DATA FILE
PROCESSING OF COMMAND ABORTED
```

DKMT COMMAND

Syntax: DKMT FNAM=<filename>,IDAD=<dadfc>,IUSI=<userid>,OCOD=<fcod>

This command is used to copy a TDFM file with name <filename> onto magnetic tape, assigned to filecode <fcod>. The date of the copy is set to the current date.

If several TDFM files are to be saved, they are separated on the tape by one tapemark; the last TDFM file is followed by two tapemarks.

The user must position the tape correctly before issuing this command.

If a physical end-of-tape is encountered during the copy, the EDF processor halts after sending the following message:

```
    LOAD NEXT TAPE AND RESTART
```

The operator must mount a new tape, position it to the start of tape and restart the EDF Processor.

Notes:

1) Saving a protected file must only be done between two TDFM runs using this file.
2) a protected file must not be saved using the Librarian, because the internal date must be modified.

Errors

```
/XXXX: BACKSPACE ERROR ON MT
/XXXX: IO ERROR ON MT WHEN WRITING EOV
/XXXX: IO ERROR ON MT WHEN WRITING TM
/XXXX: WRITE IO ERROR ON MT
/XXXX: IO ERROR ON MT WHEN WRITING EOS
/XXXX: READ IO ERR. ON SUBFILE FNAM = <name> USER = <userid> dad = <dadfc>
         PACK = <pack>
/XXXX: BACKSPACE ERR. ON MT WHEN POSIT IT FOR NEXT DKMT
         FILE CORRECTLY OUTPUT
/XXXX: ASS. ERR. ON DESC. FILE
/XXXX: ASS. ERR. ON SUBFILE FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
INCORRECT DAD VALUE
INCORRECT FNAM PARAM
DAD FC NOT FOUND FOR SUBFILE FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
WRONG OCOD FC
OCOD IS NOT ASSIGNED TO A TAPE
```

MTDK COMMAND

Syntax: MTDK FNAM=<name>,ODAD=<dadfc>,OUSI=<userid>,ICOD=<fcod>

This command is used to restore a TDFM file from a magnetic tape copy; the date on the restored file will be that of the copy. The user must correctly position the tape before issuing this command.

If physical end-of-tape is encountered during execution of this command, the EDF Processor sends this message:

```
    LOAD NEXT TAPE AND RESTART
```

The operator must then mount the next tape, position it to start of tape and restart the EDF Processor.

Notes:

1) During a restore run, all necessary DADs must be on-line.
2) When restoring a TDFM file whose Descriptor subfile is corrupt, one of the following procedures must be used:
   a) delete all subfiles using the Librarian DEL command, then use a sequence of COPY or REPL commands to restore each subfile in turn, or
   b) re-create the same empty TDFM file using FILE, KEY and DATA commands (old subfiles are automatically deleted), then use SAVE or REST commands to copy all the subfiles.
3) Restoring a protected file must only be done before back-up recovery.

Errors

```
/XXXX: READ IO ERR. ON MT, DESC. FILE
/XXXX: READ IO ERR. ON MT, SUBFILE: FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
/XXXX: READ IO ERR. ON DISC, DESC. FILE
/XXXX: WRITE IO ERR. ON DISC, TEMPORARY FILE
/XXXX: ASS. ERR. ON DESC. FILE
/XXXX: ASS. ERR. ON SUBFILE: FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
/XXXX: DELETE ERR. ON DESC. FILE
/XXXX: ASSIGN ERR. ON TEMPORARY FILE
/XXXX: CATALOGUE ERR. ON NEW DESC. FILE
/XXXX: WRITE IO ERR. ON SUBFILE: FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
INCORRECT DAD VALUE
INCORRECT FNAM PARAM
DAD F.C. NOT FOUND FOR SUBFILE: FNAM = <name> USER = <userid> DAD = <dadfc>
         PACK = <pack>
WRONG ICOD FC.
ICOD NOT ASSIGNED TO A TAPE
DESC. ON DISC NOT = TODESC. ON TAPE, SUBFILE: FNAM = <name> USER = <userid>
         DAD = <dadfc> PACK = <pack>
THE FILE IS ALREADY ASSIGNED IN THE SYSTEM
 USE PFC COMMAND (FCL) TO KNOW THE F. CODES ASSIGNED TO IT
```

Warning Message

```
NO MORE TDFM FILES ON THIS TAPE
```

RECOVERY COMMANDS

The topic of recovery of TDFM files is covered more extensively in Chapter 5. Here it suffices to mention that the EDF Processor contains commands to assign back-up and back-out files, to initialise the recovery mechanisms, to perform back-up and back-out recovery and to change the protection level of a file. These commands are described on the following pages.

BOGN COMMAND

Syntax: `BOGN FNAM=<filename>,USID=<userid>,DAD=<dadfc>[,NBGR=<int>]`

This command creates a back-out file with name <filename> in the user area <userid> held on DAD <dadfc>. The file is a non-consecutive catalogued system file, type UF. The userid under which it is catalogued must be the first userid in the DAD. The file may provide back-out protection for a number of TDFM files.

The NBGR parameter specifies the number of granules to be allocated to the file; if omitted, the initial allocation is no granules, but granules are allocated to the file dynamically as they are needed. A granule must consist of at least three sectors.

The sector size, $\underline{s}$, within the back-out file limits the maximum number of simultaneous transactions, $\underline{t}$, as follows:

$$t = s - 3 / 2$$

Before issuing this command, the user must ensure that:
1) No back-out recovery is required for any of the TDFM files associated with this file, and
2) the old back-out file, if any has been deleted.

This file may be used until it is full; with dynamic allocation of granules, this means until DAD overflow occurs. Note that a granule is freed when a transaction is successfully completed.

Errors

```
TEMP. FC. ASS. ERR.
WRITE IO ERROR ON B-OUT FILE
B-OUT KEEP FILE ERR.
B-OUT DAD GET INFO. ERR.
ERRONEOUS B-OUT F. CODE
# SEC./GRAN. < 3
```

BUGN COMMAND

Syntax: BUGN FNAM=<filename>,USID=<userid>,DAD=<dadfc>,NBGR=<integer>

This command creates a back-up file with name <filename> in user area <userid> on DAD <dadfc>. The file is a consecutive catalogued system file, type 'UF'. The userid under which it is catalogued must be the first userid of the DAD. The file may provide back-up protection for a number of TDFM files.

The NBGR parameter specifies the number of granules to be allocated to this file. The file may be used until it is full, regardless of the incidence of IPL's or saving, restoring and recovery of associated TDFM files. The number of free sectors is returned:

1) by the INSE command, on the operator's console;
2) by the 'Transaction Ready' LKM call, in the user's ECB.

Before issuing this command, the user must ensure that:

1) no back-up recovery is required for any of the TDFM files associated with this file, and
2) the old back-up file, if any, has been deleted, and
3) security copies have been made or updated of all TDFM files associated with this file.

Errors

TEMP. FC. ASS. ERR.
B-UP SECTOR 0 WR. ERR.
B-UP KEEP FILE ERR.
B-UP DAD GET INFO. ERR.
ERRONEOUS B-UP F. CODE
B-UP PREMARK ERROR

INSE COMMAND

Syntax: INSE IDEN='<runident>',BUFC=<filecode1>[,BOFC=<filecode2>]

```
    <runident>      ::=   <ASCII char>[<ASCII char>]...  (max 10 chars)
```

This command initialises the recovery mechanisms at the beginning of a 'run'. The <runident> may be chosen by the user; it must uniquely identify each run. The BUFC parameter specifies the back-up filecode and the BOFC parameter, if present, specifies the back-out filecode; these filecodes must previously have been assigned in the System Machine - by SCL ASG commands, if necessary.

At the time this command is entered, the date and time in the system must be correct, and the Real Time Clock started.

The number of free sectors remaining in the back-up file is printed on the operator's console before the command exits.

Errors

STILL ACTIVITY ON PR. FILES
UNKN. B-UP F. CODE
WRONG B-UP F. CODE
UNKN. B-OUT F. CODE
WRONG B-OUT F. CODE
DYN. AREA OVF. IN SYSTEM MACHINE
PREVIOUS RUN INCOMPLETE - B-UP RECOVERY COMPULSORY
PREVIOUS RUN INCOMPLETE - RECOVERY COMPULSORY
B-UP FILE PARTIALLY OVERWRITTEN - B-UP RECOVERY COMPULSORY
B-UP READ IO ERR.
B-UP WRITE IO ERR. - RECOVERY NOT NEEDED - SAVE EXT. FILES & GENERATE NEW B-UP FILE
B-UP OVERFLOW
- the command is executed in this case to enable back-out recovery of this overflow
B-OUT READ IO. ERR.
B-OUT WRITE ER. - RECOVERY NOT NEEDED - GENERATE NEW B-OUT FILE

RBUP COMMAND

Syntax: RBUP BUFC=<filecode1>[,BOFC=<filecode2>][,NRUN=<integer>]

This command recovers a number of TDFM files from their back-up file, and their back-out file if present, after a system failure. All the TDFM files are assumed to have been restored from their latest copies; for each file, the date and time of the copy marks the point at which recovery begins. From this point on, all modifications made by complete transactions are re-performed.

For each run on the back-up file, the run identifier is output followed by one of these messages:

a) EMPTY RUN
b) COMPLETE RUN
c) NOTHING TO BE REDONE IN RUN
d) FILE COPIES ARE POSTERIOR TO RUN
e) NO COMPLETE TRANSACTION ON FILES ANTERIOR TO RUN
f) INCOMPLETE RUN, RECOVERED TRANSACTIONS:
   - followed by a list of the transaction numbers recovered.

When recovery is complete, the back-out file is empty; both files may continue to be used for subsequent runs.

If an unrecoverable I/O error occurs while reading the back-up file during a run, the recovery process should be restarted. When entering this command again, the NRUN parameter should be supplied, where <integer> gives the number of complete runs successfuly recovered. In this case the back-up file must not be used again, but deleted and a new file generated.

Errors

```
INCORRECT BU FILE
INCORRECT BO FILE
BO ACCESS ERROR - RECOVERY SUCCEEDED - GENERATE NEW EMPTY B-OUT FILE
ST = /<ssss>, GET BUF. ERR.
BO FC. NOT GIVEN
B-UP FILE EMPTY
DYN AREA OVF. IN SYST. MACH.
ST = /<ssss>, GET INFO. ERR.
UNKN. FILE
FC = <filecode>, ST = /<ssss>, IO ERR.
ST = /<ssss>, TR. READY ERR.
FC = /<filecode>, ST = /<ssss>, TDFM IO ERR
FC = <filecode>, ST = /<ssss>, ASS. ERR.
ST = /<ssss>, TR. FINISHED ERR.
FC = /<filecode>, ST = /<ssss>, DEL. FC. ERR.
B-UP WRITE ERR - RECOVERY SUCCEEDED - SAVE EXT. FILES & GENERATE NEW EMPTY B-UP
    FILE
FREE BUFF ERR
```

Warning Messages

If the NRUN parameter is present, the following messages are output when the command has finished processing:

```
END OF PARTIAL RECOVERY
CARE, BU FILE MUST NO LONGER BE USED FOR LOGGING!!!
```

SPRO COMMAND

Syntax: SPRO FNAM=<filename>,USID=<userid>,DAD=<dadfc>,SECU={NO | BU | FULL}

This command alters the protection level of an existing TDFM file with name <filename>, whose Descriptor subfile is held in user area <userid> on DAD <dadfc>. The new protection level is specified by the SECU parameter:

NO	no protection
BU	back-up protection only
FULL	back-up and back-out protection.

The command must not be issued for a file which needs to be recovered. Before altering the protection level, it is recommended to make a security copy of the file, in case of write I/O error. After altering the protection level, all existing copies should be updated so that they also have the new protection level.

If back-up protection is removed from a file, but the current back-up file contains records for that file, then all other TDFM files using that back-up file must be copied before destroying the current back-up file and generating a new one.

Errors

```
INVALID SECU PARAM VALUE
ASSIGN WORK FC. ERR. STAT. = <ssss>
WRITE IO ERR ON DESC. FILE STAT. = <ssss>
DELETE WORK FC. ERR. STAT. = <ssss>
```

DIAGNOSTIC AND OTHER COMMANDS

Two commands, the DUMP ad STAT commands, allow the user to print diagnostic information about a TDFM file. The SBUF command allows the user to alter the number of buffers in the pool used for access to TDFM files.

Two commands exist, EFEN and ABT, to terminate the EDF Processor; EFEN may only be used when all activity from other commands is complete, whereas ABT aborts any such activity.

DUMP COMMAND

```
Syntax:  DUMP FNAM=<filename>,USID=<userid>,DAD=<dadfc>,KNAM=<keyname>
         [,FROM=<keyvalue1>][,TO=<keyvalue2>][,PRNT=<filecode>]

         <keyvalue>  ::=  <integer>{A | D | H}<value>$$[<keyvalue>]
```

The DUMP command dumps data records from the TDFM file with name <filename>, whose Descriptor subfile is held in user area <userid> on DAD <dadfc>. The data records are dumped in ascending order of key <keyname>, in hexadecimal format with character equivalents.

If the FROM and TO parameters are both present, all records whose value of key <keyname> lies between the specified values are output. If the FROM parameter is omitted, dumping starts with the lowest key value in the file; if the TO parameter is omitted, dumping ends with the highest key value in the file. Thus, omitting both parameters causes the whole file to be dumped.

The key values defined in the FROM and TO parameters are made up of one or more subfields, each with length defined by <integer>. The <value> is as follows:

A - <integer> ASCII characters.
D - a decimal number, left-padded if necessary to a length of <integer>.
H - a hexadecimal number (two hexa digits per byte), left-padded with zeroes if necessary.

The key length defined in either parameter must be the same as that defined for key <keyname>.

The PRNT parameter, if present, gives the filecode of the print device; if omitted, filecode /02 is assumed.

Errors

```
ERR. WHEN ASS. WORK FC. TO DESC. FILE
/<ssss> READ IO ERR. ON DESC. FILE
UNKNOWN KEY NAME
/<ssss> DELETE DFM WORK FC. ERR.
/<ssss> ASS. WORK FC. TO EDFM FILE ERR.
/<ssss> TR. READY IO ERR.
TO VAL. ≥ PADDING KEY
FROM VAL. > TO VAL.
/<ssss> POSIT IO ERR.
/<ssss> TDFM READ IO ERR.
/<ssss> PRINT ERR. ON FC /44
/<ssss> READ NEXT IO. ERR.
/<ssss> TR. FINISH IO. ERR.
/<ssss> DELETE TDFM WORK FC. ERR.
INVALID PRINT FC
LENGTH OF FROM VALUE IS INCORRECT
LENGTH OF TO VALUE IS INCORRECT
FROM VALUE > GREATEST KEY VALUE
UNKNOWN DAD
UNKNOWN FILE NAME
UNKNOWN USER ON GIVEN DAD
USER OF 1 SUBFILE IS UNKNOWN
NAME OF ONE SUBFILE IS UNKNOWN
DAD OF ONE SUBFILE IS UNKNOWN
ONE PACK NOT ON LINE
DAD OF ONE SUBFILE NOT ASSIGNED
```

FILE ALREADY ASS. IN ANOTHER MACH. WITH DIFF. DAD FC. FOR SUBFILES
WORK FC. WAS PREVIOUSLY ASS. TO A TDFM FILE ON WHICH A TRANS. IS ALREADY
    OPENED
DYN. AREA OVF. IN SYST. MACH. DURING ASSIGN

STAT COMMAND

Syntax: STAT FNAM=<filename>,USID=<userid>,DAD=<dadfc>[,PRNT=<filecode>]

This command prints statistics concerning a TDFM file with name <filename>, whose Descriptor subfile is held in user area <userid> on DAD <dadfc>. Output is always to the line printer; if the PRNT parameter is supplied, the statistics will also be output to the specified <filecode>.

The statistics give the following information about the file as a whole:
- date of creation or last copy;
- protection level;
- number of records currently in the file;
- number of records which have been deleted;
- whether index or data overflow has occurred.

For each subfile, the name, userid, DADname and pack number are output. For all except the Descriptor subfile, the number of free sectors is output, together with an indication of overflow. For each data subfile the number of records it currently contains and the number which have been deleted are output.

If either an index or a data file has overflowed, or has only a very small number of free sectors so that overflow is imminent, the file needs to be reorganised. Refer to the section 'File Reorganisation' in this Chapter.

Errors

```
ASSIGN WORK FC. ERR. STAT = <ssss>
READ IO ERROR ON DISC. FILE STAT = <ssss>
DELETE WORK FC. ERR. STAT = <ssss>
```

SBUF COMMAND

Syntax: SBUF NBUF=<integer>

This command alters the maximum number of buffers in the TDFM pool, held in the CVT, to <integer>; the value must be a multiple of three, the number of buffers used to access a file.

Errors

SBUF NOT A MULTIPLE OF 3

Note: Each transaction requires three buffers for each TDFM file opened; one for the Descriptor subfile, one for the current key file and one for the current data file. The number of buffers required is therefore:

3 * T * F

where:

T	is the maximum number of simultaneous transactions, decided by the system manager (not more than 100);
F	is the number of TDFM files opened by each transaction.

EFEN COMMAND

Syntax: EFEN

This command exits from the EDF Processor and returns control to the Monitor. The command is rejected if the action of previous commands, such as disc space allocation or TDFM file copying, is still in progress.

Errors

None.

ABT COMMAND

Syntax: ABT

This command exits from the EDF Processor and returns control to the monitor. If the action of previous commands, such as disc space allocation or TDFM file copying, is still in progress it is aborted.

Errors

None.

Input and output to and from the extended file is effected by means of LKM 1 requests issued by the program.

CALLING SEQUENCE

```
LDK       A7,C
LDKL      A8,B
LKM
DATA      [-]1
[DATA     L]
```

where:

L	is the address of a scheduled label routine
B	is the address of an event control block (ECB), whose format depends on the order code.
C	is a word having the following format:

bit 8 = 1 Implicit wait: The program is suspended until the I/O is complete.
bit 8 = 0 No implicit wait: Control is returned to the requesting program as soon as the transfer has been initiated.

bit 9 = 1 User error action: The requesting program will process all abnormal or error conditions; the hardware status is returned in this case.
bit 9 = 0 System error action: The system performs the standard error action and returns a status code to the ECB of the calling program.

bits 10 to 15 contain one of the following order codes:

/02	Read Next
/0A	Read on Key Value
/0B	Write
/12	Read Next and Attach
/1A	Read on Key Value and Attach
/1B	Read on Physical Coordinates
/1C	Read on Physical Coordinates and Attach
/25	Transaction Ready
/27	Abort Transaction
/28	Finish and Cancel Transaction
/29	Back-out Recovery
/2A	Detach a Record
/2B	Detach All Records Attached to a Transaction
/2C	Write and Attach
/2D	Delete
/2E	Replace Record
/3A	Read Previous and Attach
/3B	Transaction Finished
/3C	Position on Key Value
/3D	Replace Record and Detach
/3E	Delete and Detach
/3F	Read Previous.

TRANSACTION READY (Order Code /25)

This is the first request issued by a transaction and serves to define the transaction identifier and open all the files which may be accessed during transaction processing.

ECB Structure

The ECB layout is as follows:

```
      -------------------------------------------------
 -4   |    Transaction Number                         |
      |-----------------------------------------------
 -2   |    Linked Event Address (if L = 1)            |
      |-----------------------------------------------
  0   | E | L |               |        Filecode 1
      |-----------------------------------------------|
  2   |    Number of Files to be Opened               |
      |-----------------------------------------------|
  4   |Number of Free Sectors in the Back-up File     |
      |-----------------------------------------------|
  6   |                 Unused                        |
      |-----------------------------------------------|
  8   |                 General Status                |
      |-----------------------------------------------|     -----
 10   |                       |      Filecode 2       |     This block of
      -------------------------------------------------
 12   |    Opening Mode                               |     four words is
      -------------------------------------------------
 14   |    Zero                                       |     repeated for
      -------------------------------------------------
 16   |    Opening Status                             |     each file opened.
      |-----------------------------------------------|      -----
 18   |    etc.                                       |
      |-----------------------------------------------|
```

Where:

- `Transaction Number' is a unique identifier and must not be duplicated in this or any other machine.
- `E' (bit 0) is the event bit; it is set by MAS when the I/O is complete.
- `Filecode 1' is a dummy filecode, and should be set to any extended filecode in the application.
- `General Status' is returned by MAS and, if the request is rejected, gives the reason. These codes are described below under the heading `Returned Status'.
- `Number of Free Sectors in the Back-up File' is returned by MAS to allow the user to terminate processing before a file overflow occurs.
- `Linked Event Address' is the address of another ECB in a chain of ECB's which are all linked for event handling purposes (i.e. they are all awaiting the same event). If the location `L' of an ECB is set to 1, then the Linked Event Address will contain the address of the next ECB in the chain. If there are no linked ECB's (this is the only member of the chain), or this is the last ECB in the chain, then both the location `L' and the Linked Event Address should be set to zero.

- Bytes 10 to 17 form a four-word block, repeated for each file to be used during the processing of this transaction. Within each four-word block:

word 0	contains the TDFM filecode in bits 8-15.
word 1	contains the opening mode for the file (see below).
word 2	is unused (set to zero).
word 3	contains, on return, a status code which, if the request is rejected, identifies the file and the reason for the rejection.

Opening Modes

/2F - Exclusive Mode

This mode reserves the file for exclusive use by this transaction until a 'Transaction Finished' request is issued. If the file is already in use, this request is suspended until all current transactions have finished, and no new 'Transaction Ready' attempting to open this file will be accepted until this transaction finishes.

Following acceptance of this request any of the other order codes may be used, but any request to update or delete a record must still be done with 'Attach', despite the exclusive use.

/39 - Read Mode

In this mode, records may be not be deleted, updated, written or attached, but all other requests are accepted unless a record has been attached by another transaction, in which case this request will be suspended until the record is detached.

This 'Transaction Ready' request will be rejected if the file is already open in exclusive mode.

/3A - Update Mode

All order codes will be accepted following the successful opening of a file in this mode, but if records are to be deleted or updated, they must first be attached.

Notes

If security protection is in use on one or more of the files, the number of sectors remaining on the back-up file is returned to the user's ECB by the system and a special mark is written to the back-up file to indicate the start of a new transaction.

If back-out protection is in use, a granule of the the back-out file is allocated to this transaction. This granule is released on successful termination of the transaction.

A 'Transaction Ready' request will be rejected if there are already 100 transactions currently processing.

TRANSACTION FINISHED (Order Code /3B)

This is the last request issued by a transaction; it closes all the files opened during the processing of the Transaction Ready request, detaches all attached records and releases the exclusive use restriction on any files opened in that mode.

If the back-up file was in use during the transaction, an end-of-transaction mark is written.

The ECB layout is as follows:

-4	Transaction Number			
-2	Linked Event Address (if L=1)			
0	E	L		Filecode
2	Unused			
4	Unused			
6	Unused			
8	Status			

where:

'Filecode' is that of any extended file in use in this application.

'Status' is a code returned by the system indicating the reason for rejection if this request should fail.

This request is performed only when all outstanding requests issued by this transaction have been completed. It is rejected if any request issued during this transaction was rejected; in this case, the user must issue an 'Abort Transaction' (order code /27) request.

After the Transaction Finished request has been successfully processed, any requests from other transactions awaiting access to records which were attached to this transaction will be actioned.

Note: When a user fails to send a 'Transaction Finished' request for any reason (e.g. he forgets or is stopped by abnormal termination), and he is using unprotected files only, the files opened by this transaction may be damaged and must no longer be used. If protected files are used this is detected by the system, and the user is forced to recover before doing further work with the damaged file.

ABORT TRANSACTION (Order Code /27)

This request is used to terminate a transaction for which the normal terminations (Transaction Finished or Finish and Cancel) have been forbidden or have failed. It is designed to prevent a blocking of the system, and for this reason any Read requests queued for records attached to these faulty transactions are serviced with an appropriate error status. In the case of protected files further queueing of requests for these records is rejected, while for unprotected files they are detached in the normal manner.

The ECB layout is as follows:

```
       |----------------------------------------|
 -4    |           Transaction Number           |
       |----------------------------------------|
 -2    |     Linked Event Address (if L=1)      |
       |----------------------------------------|
  0    | E | L |       |       Filecode         |
       |----------------------------------------|
  2    |                 Unused                 |
       |----------------------------------------|
  4    |                 Unused                 |
       |----------------------------------------|
  6    |                 Unused                 |
       |----------------------------------------|
  8    |          Returned Status               |
       ------------------------------------------
```

<u>Filecode</u>	is any extended filecode in use in the application.
<u>Status</u>	is a code returned by the system; these codes are described below under the heading 'Returned Status'.

FINISH AND CANCEL TRANSACTION (Order Code /28)

This request is actioned in the same way as the Finish Transaction request, but has the additional function of undoing all the modifications that have been performed on files protected in back-out mode betwen the time the transaction started and the time this request was issued. It can only be used if all the files opened in update or exclusive mode are back-out protected.

This request is rejected if any of the file-modifying requests on protected files were incorrectly performed. The user must then use the Abort Transaction request.

ECB Layout

```
      |------------------------------------------|
 -4   |          Transaction Number              |
      |------------------------------------------|
 -2   |     Linked Event Address (if L=1)        |
      |------------------------------------------|
  0   | E | L |       |          Filecode        |
      |------------------------------------------|
  2   |                Unused                    |
      |------------------------------------------|
  4   |                Unused                    |
      |------------------------------------------|
  6   |                Unused                    |
      |------------------------------------------|
  8   |       Returned Status                    |
      |------------------------------------------|
```

Filecode	is any extended filecode in use in the application.
Status	is a code returned by the system; these codes are described below under the heading 'Returned Status'.

BACK-OUT RECOVERY (Order Code /29)

This request must be the first one issued in the first TDFM run following a system failure. It performs a multiple Finish and Cancel Transaction operation on all the transactions which were incomplete when the system failure occurred.

If the recovery succeeds, the transaction numbers of all cancelled transactions are returned to the user in a Result Block - an area within the user's program which he must provide for this purpose. The user enters the address of this block in his ECB prior to issuing the request; he must ensure that sufficient room is provided, otherwise the request is rejected and error status /A603 is returned.

This request is only accepted if all the files opened in update or exclusive mode are back-out protected.

Recovery

Recovery may be performed in two cases:
1) A complete system failure requiring IPL occurred.
2) One or more transactions cannot be terminated by a Transaction Finished or Finish and Cancel order.

In case 1, an IPL is performed and the EDF `INSE' command is used.

In case 2, the user must delete all TDFM filecodes assigned in all machines and then use the EDF `INSE' command.

ECB Layout

-4	Zero			
-2	Linked Event Address (if L=1)			
0	E	L		Filecode
2	Result Block Word Address			
4	Result Block Length			
6	Length of Information Returned in Result Block by the System			
8	Returned Status			

Filecode is any extended filecode in use in this application.
Status is a code returned by the system; these codes are described below under the heading `Returned Status'.

Result Block Layout

```
|--------------------|   ---
|         L          |
|--------------------|
|-       Run       --|
|-     Identity    --|
|--------------------|
| Transaction No. 1  |
|--------------------|    L
| Transaction No. 2  |
|--------------------|
|     etc.           |
|                    |
|--------------------|
| Transaction No. n  |
|--------------------|   ---
|         -          |
|--------------------|
```

The unused portion of the block is zero filled.

POSITION ON KEY VALUE (Order Code /3C)

This request 'positions' the file so that the next 'Read Next' request issued by this transaction will read the first data record whose key value immediately exceeds that specified in this request.

Similarly, if this request is followed by a 'Read Previous' request, the record having the next lower key is delivered.

Both these rules hold even if records are added to or deleted from the file between the 'Position' request and either of the read requests.

ECB Structure

```
       |-------------------------------------------|
  -4   |           Transaction Number              |
       |-------------------------------------------|
  -2   |    Linked Event Address (if L=1)          |
       |-------------------------------------------|
   0   | E | L |           |        Filecode       |
       |-------------------------------------------|
   2   |    Unused                                 |
       |-------------------------------------------|
   4   |    Unused                                 |
       |-------------------------------------------|
   6   |    Unused                                 |
       |-------------------------------------------|
   8   |    Status                                 |
       |------------------------------------------ |
  10   |                                           |
   -   ~               Unused                      ~
  18   |                                           |
       |-------------------------------------------|
  20   |                                           |
       |--                                       --|
  22   |          Key Name (three words)           |
       |--                                       --|
  24   |                                           |
       |-------------------------------------------|
  26   |                                           |
   -   |               Key Value or                |
   -   |               Padding Key                 |
   -   |                                           |
  44   |                                           |
       |-------------------------------------------|
```

If the padding key or a lower value is entered in the ECB when issuing this request, the record with the lowest key value in the file is delivered.

If the maximum key value or a key higher in value than all existing keys is used, the record having the highest value in the file is delivered.

Example

```
                Position       Read Next      Delete 3
                  on 7         (delivers 9)
Transaction A ----|-------------|--------------|----------------> time

                   Write 7        Position          Read Previous
                                     on 6           (delivers 1)
Transaction B -------|-------------|------------------|----------> time

              1      1                               1
File Key      3      3                               6
Structure     6      6                               7
              9      7                               9
                     9
```

Returned Status

See below.

READ ON KEY VALUE

Order Code /0A (Without Attach)
Order Code /1A (With Attach)

This request is used to read the record with the specified key value. The attach is required only if the record is to be replaced or deleted, and may only be requested if the file was opened in update or exclusive mode.

If no records exist with the specified key value, this request is equivalent to 'Position on Key Value'.

If several records exist with the specified key value, the first record in the homonymous chain is read and the number of remaining records in the chain is returned to the user in word 10 of the ECB. This number is not updated if records are added to, or deleted from, this homonymous chain during the processing of this transaction, either by this or any other transaction.

If the Attach request is used, the record becomes exclusive to this transaction and any requests for the record from other transactions are queued until the record is detached. This can be done:

explicitly by:
1) a Detach request,
2) a Replace and Detach request,
3) a Delete and Detach request;

or implicitly by:
a Transaction Finish request.

Notes:

1) A transaction may read a record attached by itself.
2) Requests for Attached records, the queueing of which would lead to dead-lock, are rejected.
3) A transaction may issue more than one 'Read and Attach' request for the same record.
4) If a transaction is reading a homonymous chain, the remaining count returned by MAS may be incorrect if other transactions are adding records to, or deleting records from, this homonymous chain. However, the warning status 'Beginning of Homonymous Chain', returned to the user, is always correct.
5) A Read on Key request for a non-existent record is equivalent to a Position on Key for that key value.

ECB Structure

```
     |-----------------------------------------------|
-4   |              Transaction Number               |
     |-----------------------------------------------|
-2   |        Linked Event Address (if L=1)          |
     |-----------------------------------------------|
 0   | E | L |           |         Filecode          |
     |-----------------------------------------------|
 2   |        Buffer (character) Address             |
     |-----------------------------------------------|
 4   |        Requested Length                       |
     |-----------------------------------------------|
 6   |        Actual Length                          |
     |-----------------------------------------------|
 8   |        Returned Status                        |
     |-----------------------------------------------|
10   |        Remaining Count                        |
     |-----------------------------------------------|
12   |        Unused                                 |
     |-----------------------------------------------|
14   |                                               |
     |--                                           --|
16   |        Record Co-ordinates                    |
     |--                                           --|
18   |                                               |
     |-----------------------------------------------|
20   |                                               |
     |--                                           --|
22   |                  Key Name                     |
     |--                                           --|
24   |                                               |
     |-----------------------------------------------|
26   |                                               |
--   |                                               |
--   |                  Key Value                    |
--   |                                               |
44   |                                               |
     |-----------------------------------------------|
```

where:

- Record Coordinates are:
    the data file number (from zero)
    the sector number
    the record displacement within the sector.

  These co-ordinates are returned by the system.

- Remaining Count is the remaining number of records in the homonymous chain, and is returned by the system.

READ NEXT

Order Code /02 (Without Attach)
Order Code /12 (With Attach)

This request is used to read the data record having the next ascending key value to that specified in the user's ECB.

The request may only be issued after a 'Position on Key Value' or 'Read on Key Value' request, or after a previous 'Read Next' request.

'Attach' may only be requested if the file was opened in update or exclusive mode, and is mandatory if the record is to be deleted or replaced (updated).

If a 'Read on Key Value' request has been given and the key specified in that request was homonymous, the 'Read Next' command can be used to access the remaining records of the homonymous chain.

ECB Structure

```
      |----------------------------------------------|
 -4   |          Transaction Number                  |
      |----------------------------------------------|
 -2   |          Linked Event Address (if L=1)       |
      |----------------------------------------------|
  0   | E | L |            |     Filecode            |
      |----------------------------------------------|
  2   |          Buffer (character) Address          |
      |----------------------------------------------|
  4   |          Requested Length                    |
      |----------------------------------------------|
  6   |          Actual Length                       |
      |----------------------------------------------|
  8   |          Returned Status                     |
      |----------------------------------------------|
 10   |          Remaining Count                     |
      |----------------------------------------------|
 12   |          Unused                              |
      |----------------------------------------------|
 14   |                                              |
      |--                                          --|
 16   |      Record Co-ordinates (if Attached)       |
      |--                                          --|
 18   |                                              |
      |----------------------------------------------|
```

Returned Status

See below.

READ PREVIOUS

Order Code /3F (Without Attach)
Order Code /3A (With Attach)

Once the file has been `positioned' (by a `Read on Key' or `Position on Key' request), the `Read Previous' request can be used to access the record with the key value immediately lower than that specified in the user's ECB.

The `Attach' request may only be used if the file was opened in update or exclusive mode, and is mandatory if the record is to be deleted or updated.

ECB Structure

The ECB layout is as for the `Read Next' request.

Returned Status

See below.

REPLACE RECORD

Order Code /2E (Replace Only)
Order Code /3D (Replace and Detach)

This request is used to update the data part of a record; the record must already be Attached. File Positioning is not affected.

The record is written back to the identical position on the disc from which it was read, and its key values are unchanged. The request is rejected if the user has changed any key value, or if the record length differs from that of the original record.

ECB Structure

```
      |---------------------------------------------|
 -4   |              Transaction Number             |
      |---------------------------------------------|
 -2   |      Linked Event Address (if L=1)          |
      |---------------------------------------------|
  0   | E | L |              |     Filecode         |
      |---------------------------------------------|
  2   |          Buffer (character) Address         |
      |---------------------------------------------|
  4   |          Requested Length                   |
      |---------------------------------------------|
  6   |          Actual Length                      |
      |---------------------------------------------|
  8   |          Returned Status                    |
      |---------------------------------------------|
 10   |                                             |
      |---------------------------------------------|
 12   |                                             |
      |---------------------------------------------|
 14   |     Record Co-ordinates (as returned by     |
      |--   a previous Read and Attach request    --|
 16   | (file No./ sector No./ recd displacement)   |
      |--                                         --|
 18   |                                             |
      |---------------------------------------------|
```

Filecode	is that of the TDFM file to be updated, which must have been opened in update or exclusive mode.
Actual Length	is returned by the system.

Notes:

To change the key value or length of a record, the following procedure should be used:

Read with Attach
Delete
Modify the record in the user buffer
Write the record.

READ ON PHYSICAL CO-ORDINATES

Order Code /1B (Without Attach)
Order Code /1C (With Attach)

This request may be used if rapid access to a record is required and the co-ordinates are known - either because they were returned by a previous Read request, or they were computed by the user.

The requested length must be equal to the actual (effective) length of the record.

File positioning is not affected by this request.

ECB Structure

```
          |--------------------------------------|
  -4      |          Transaction Number          |
          |--------------------------------------|
  -2      |    Linked Event Address (if L=1)     |
          |--------------------------------------|
   0      | E | L |            |     Filecode    |
          |--------------------------------------|
   2      |    Buffer (character) Address        |
          |--------------------------------------|
   4      |         Requested Length             |
          |--------------------------------------|
   6      |         Actual Length (returned)     |
          |--------------------------------------|
   8      |         Returned Status              |
          |--------------------------------------|
  10      |         Unused                       |
          |--------------------------------------|
  12      |         Unused                       |
          |--------------------------------------|
  14      |               Record                 |
          |--                                  --|
  16      |            Coordinates               |
          |--                                  --|
  18      |  (File No. / Sector No. / Recd. Disp.)|
          |--------------------------------------|
```

Notes:

No remaining count is returned.

Returned Status

See below.

DELETE A RECORD

Order Code /2D (Without Detach)
Order Code /3E (With Detach)

This request causes a record to be flagged as deleted in both the data and index files. The remaining count of homonymous records is decremented as necessary.

The request will be rejected if the record is not attached to the transaction.

ECB Structure

```
        |---------------------------------------|
  -4    |      Transaction Number               |
        |---------------------------------------|
  -2    |    Linked Event Address (if L=1)      |
        |---------------------------------------|
   0    | E | L |      |         Filecode       |
        |---------------------------------------|
   2    |             Unused                    |
        |                                       |
   4    |                                       |
        |                                       |
   6    |                                       |
        |---------------------------------------|
   8    |            Returned Status            |
        |---------------------------------------|
  10    |               Unused                  |
        |                                       |
  12    |                                       |
        |---------------------------------------|
  14    |         Record Coordinates            |
        |--                                   --|
  16    |     (as returned by a previous Read   |
        |--            with Attach)           --|
  18    | (File No./ Sector No./ Displacement)  |
        |---------------------------------------|
```

Notes:

1) This request does not affect file positioning for the requesting transaction.
2) A `Delete and Detach' request is rejected for a protected file.
3) If the `Delete and Detach' request is used, an automatic detach is executed following the the delete. Otherwise, if the detach option is not used, the record remains attached and may have Reads queued on it.
4) There is no recovery of disc space by this request and an overflow condition arising from a previous Write operation is not removed following a Delete request. The overflow condition can be removed by file reorganisation or back-out recovery.

Returned Status

See below.

WRITE A RECORD

Order Code /0B (Without Attach)
Order Code /2C (With Attach)

This request is used to add or insert a new record into a TDFM file, the indexes being updated as necessary. The request is rejected if the file is open in Read mode. In the case of a homonymous key for which one or more records already exist, the new record becomes the last of the homonymous chain and the homonymous count is incremented accordingly.

If `Attach' is not used, the record becomes accessible to any other transaction. A Write without `Attach' is rejected for a protected file.

ECB Structure

-4	Transaction Number			
-2	Linked Event Address (if L=1)			
0	E	L		Filecode
2	Buffer (character) Address			
4	Requested Length			
6	Effective (Actual) Length			
8	Returned Status			
10	Nbk (returned by MAS)			
12	fn			

where:

Nbk	is the number of the KEY command defining the index file which was bijective but, as a result of this command, has become homonymous by the addition of this record.
fn	defines the data file to which this record is to be added; possible values are:
	/8000 Add to the first data file in which there is room.
	/C000 Add to the data file within whose criterion key range the criterion key of this record lies.
	Data file number; these run from 0 upwards, corresponding to the order in which the files were defined.

Remarks

1) This request destroys file positioning.
2) If he uses file numbering, the user's program becomes dependent on the physical file organisation. This can be a serious inconvenience if it becomes necessary to change the file organisation later.

Returned Status

See below.

DETACH ONE OR ALL RECORDS

Order Code /2A (Detach One Record)
Order Code /2B (Detach All Records)

This request will detach one or all records attached to the requesting transaction. If there are requests queued for these records, the following may occur:

1) If the record has been deleted, a status is returned to the requesting transaction.
2) If the record has not been deleted, all the read requests are serviced until a 'Read With Attach' is encountered or the end of the queue is reached.

ECB Structure

```
         |--------------------------------------------|
   -4    |            Transaction Number              |
         |--------------------------------------------|
   -2    |       Linked Event Address (if L=1)        |
         |--------------------------------------------|
    0    | E | L |               |     Filecode       |
         |--------------------------------------------|
    2    |              Unused                        |
         |                                            |
    4    |                                            |
         |                                            |
    6    |                                            |
         |--------------------------------------------|
    8    |         Returned Status                    |
         |--------------------------------------------|
   10    |                                            |
         |         Unused                             |
   12    |                                            |
         |--------------------------------------------|
   14    |           Record Co-ordinates              |
         |---                                      ---|
   16    |         (for detach one record)            |
         |---                                      ---|
   18    |    (File No./ Sector No./ Recd. Disp.)     |
         |--------------------------------------------|
```

Notes:

1) In the case of request code /2B (Detach All Records), 'Filecode' in word 0 of the ECB refers to any TDFM filecode in the user's application.
2) Request order code /2A (Detach a Record) is rejected if the specified file is protected.
3) Request code /2B will only work for those files which are not protected.

RETURNED STATUS

Apart from the general LKM 1 error codes, the user may receive the following codes which are specific to TDFM:

Warning Status

Value	Meaning
/0008	The requested length for a read operation is less than the record length: the transferred length = the requested length. the effective length = the record length.
/1000	The record just read is the start of a homonymous chain.
/1008	Both the warnings for /1000 and /0008 apply.
/1001	The record to be deleted has already been deleted.
/1002	Position on Key request with key higher than all existing keys; file is positioned at EOF.
/1003	No transactions for back-out recovery.
/D000	Internal system status for back-up recovery - ignore.

Error Status

Value	Meaning
/A000	Requested file not opened by requesting transaction.
/A001	Detach forbidden because requested file is back-out protected.
/A002	Detach forbidden, file was damaged by incorrectly performed modifying operation.
/A004	Dynamic Area overflow in the system machine; cannot allocate buffer for Transaction Table, EFT or file buffer. If this error occurs with Cancel, the Cancel may be retried.
/A005	Back-out recovery refused because run without security.
/A006	Back-out recovery not first request of run.
/A007	Unknown order code in A7.
/A008	Buffer address not in user's area.
/A009	Requested length is zero.
/A00C	Back-out recovery compulsory; any other request refused.
/A00D	Unknown transaction number.
/A00F	Transaction Ready refused because back-out recovery failed.
/A010	Transaction Ready refused because transaction already exists.
/A011	Transaction Finished already received.
/A012	Overflow of system request number for logging.
/A013	Transaction already aborted.
/A024	Transaction cannot be cancelled because none of the files opened in update mode is protected in back-out mode.
/A025	Transaction Finished forbidden because one modifying request issued by the transaction on a protected file was incorrectly performed.
/A026	Cancel Transaction forbidden (see /A025).
/A027	Abort refused because normal end of transaction allowed.
/A060	Transaction Ready for zero or negative number of files.
/A061	Two entries for the same file in Transaction Ready ECB.
/A062	Transaction Ready uses unknown filecode.
/A063	Transaction Ready on a file under exclusive access for another transaction.
/A064	Date of one protected file greater than run date.
/A06C	Transaction Ready in Read mode for an empty file.
/A06D	File still locked after back-out recovery failure.

/A06E	Transaction Ready with unknown opening mode.
/A070	Transaction Ready refused because back-out file not declared, or not on-line.
/A071	Transaction Ready refused because back-up file not declared, or not on-line.
/A072	Transaction Ready on a file damaged by previous modification incorrectly performed.
/A120	Unknown key name.
/A123	File empty.
/A150	Erroneous coordinates or wrong requested length.
/A180	Sequential read on a file not previously positioned.
/A181	Read and Attach forbidden on a file on a file opened in read mode; file positioning not destroyed.
/A183	Read Next refused because file is positioned at EOF.
/A185	Record attached to aborted transaction; queuing forbidden.
/A186	Record attached to unknown transaction.
/A187	Record to be read attached to another transaction; queueing refused to avoid deadlock.
/A188	Read without queuing on attached record.
/A243	Read on Key Value less than or equal to padding key value.
/A244	Read on non-existent key value; file positioned.
/A245	Read on key value greater than all existing key values - file positioned at EOF.
/A246	Record deleted.
/A2A0	Write without Attach attempted on a protected file.
/A2A1	Modification attempted on a file opened in Read mode.
/A2A2	Index overflow; Write not performed.
/A2A3	One key of written record less than or equal to padding key.
/A2A4	Requested length for Write greater than 4095 bytes.
/A2A5	Data overflow - Write not performed.
/A2A7	No criterion key defined for Write on requested file.
/A2A8	Written criterion key value greater than highest key declared at generation for last data file.
/A2B0	Write performed, but secondary bijective key becomes multiform (key number returned in ECB Word 10).
/A2B1	Written primary key value already exists.
/A2B6	Write uses an invalid data file number.
/A2B7	One of the written keys lies outside the user's buffer.
/A2B8	Written primary key value exists in deleted record attached to a still running transaction which may be cancelled.
/A301	Modification of a non-attached record.
/A302	Modification of a record attached to another transaction.
/A304	Detach forbidden on a protected file.
/A305	Length of replacing record not equal to record length on disc.
/A306	Not all key values are identical in the replaced and replacing records.
/A36A	Delete Record attempted using a key value not in the index.
/A36B	Record coordinates not found in index; Delete undone.
/A371	The record in data file is already deleted; file is probably corrupted.
/A372	Key value in data record not found in index.
/A373	Record coordinates not found in index; I/O error when releasing.
/A420	Detach uses wrong record co-ordinates in the ECB.
/A482	Read Previous attempted on a file positioned on the first key value; file positioning is destroyed.
/A4F0	Previous logging error forbids further use of back-up file.
/A4F4	Back-up file overflow.
/A520	Previous logging error forbids further use of back-out file.
/A525	Back-out file overflow (Disc GRANTB).
/A526	Overflow of DAD containing the back-out file.

/A532	Transaction Ready refused because too many simultaneous transactions.
/A580	One file of a transaction to be undone is not assigned.
/A590	One of the files involved in the undoing of a transaction has been damaged after the Cancel request.
/A596	Undo Write/Delete failed because a key value in the data record was not found in the corresponding index.
/A5F3	Back-out impossible because end of back-out file met before end of one transaction.
/A602	Back-out recovery stopped because inconsistency detected in back-out file.
/A603	Result block for back-out recovery is too small.
/A620	In ECB for back-out, user's buffer address not word-aligned.
/FFFF	Inconsistency detected in one index sector.

Disc I/O Errors

All the status code values relevant to disc I/O errors are of the form /Bxxx:

/B0xx	Disc not damaged.
/B1xx	Requested file(s) damaged (back-up logging not performed if file protected).
/B2xx	Back-up logging incorrectly performed.
/B3xx	Back-out logging incorrectly performed.
/B4xx	I/O error during Cancel or Back-out recovery.

Note: In the case of status codes /B441 or /B443 the back-out recovery is successfully performed, but the back-out file is damaged and a new one must be generated.

Error Code Cross Reference Table

The table below contains a list of error codes, together with the requests which may give rise to them. The requests are in coded form and these codes are listed here for convenience:

Code	Request
a	Back-out
b	Detach All Records
c	Detach One Record
d	Write and Attach
e	Write
f	Delete and Attach
g	Delete
h	Replace and Detach
i	Replace
j	Read on Physical Co-ordinates and Attach
k	Read on Physical Co-ordinates
l	Read Previous and Attach
m	Read Previous
n	Read Next and Attach
o	Read Next
p	Read on Key and Attach
q	Read on Key
r	Position
s	Abort
t	Cancel
u	Transaction Finished
v	Transaction Ready.

Cross Reference Table

Error Code	Affected Request
/A000	c to r inclusive
/A001	c
/A002	d to r inclusive
/A004	a to r and t, u, v
/A005	a
/A006	a
/A00C	b to r and t, u, v
/A00D	b to u inclusive
/A00F	v
/A010	v
/A011	b to u
/A012	d to i and t, u, v
/A013	b to v
/A024	t
/A025	u
/A026	t
/A027	s
/A060	v
/A061	v
/A062	v
/A063	v
/A064	v
/A06C	v
/A06D	v
/A06E	v
/A06F	v
/A070	a, v
/A071	v
/A072	v
/A120	p, q, r
/A123	p, q, r
/A180	j, k
/A181	l, m, n, o
/A183	j, l, n, p
/A2B6	d, e
/A2B7	d, e
/A2B8	d, e
/A301	f, g, h, i
/A302	f, g, h, i
/A304	f, h
/A305	h, i
/A306	h, i
/A36A	f, g
/A371	f, g
/A420	c
/A482	l, m
/A4F0	d to i and t, u, v
/A4F4	d to i and t, u, v
/A520	d to i and t, u, v
/A525	d to i and v
/A526	d to i and v
/A532	v

/A580	a and t
/A590	t
/A596	a, t
/A5F3	a
/A602	a
/A603	a
/A620	a
/FFFF	a, d, e, f, g, t

DEFINITIONS

An access is any access to a TDFM file at the record level, i.e. any read, update, delete an existing record or add a new one.

A transaction consists of all the accesses, plus other processing, required to perform some task (preparing an invoice, for example, or booking a passenger on a flight).

A run consists of all the transactions between two points in time, decided by the user, or until a system breakdown.

Back-up is the process of recording all the accesses in a transaction so that, in case of system breakdown, all the modifications to a TDFM file may be re-performed on a security copy of the file, thus restoring it to its condition at the time of breakdown.

Back-out is the process of recording all the accesses in a transaction so that, if the transaction is incomplete at the time of system breakdown, all the modifications to a TDFM file made by the incomplete transaction may be undone, thus restoring the file to its condition before the transaction was started.

TRANSACTIONS

A transaction is initiated by a 'Transaction Ready' call, which establishes the mode of access to a TDFM file (read-only, update or exclusive). If the file is protected it also writes a header into the back-up file, and into the back-out file if appropriate.

Each transaction has a transaction number, which must be unique in the system at any given time; that is, no two simultaneously running transactions may have the same number. The user allocates transaction numbers in any manner he chooses. The system identifies each transaction by its number.

Each access that causes a modification to the TDFM file is recorded in the back-up file, as the "after" image of the affected record. Where back-out recovery is applicable, each modifying access is recorded in the back-out file as the "before" image of the affected record.

A transaction is terminated in one of the following ways:
1) a 'Transaction Finished' call, which deletes all back-out information for the transaction, unless some error ocurred during a modifying access, in which case the back-out file needs to be kept and the user must use the 'Abort' call.
2) a 'Finish and Cancel Transaction' call, which may only be used if all the files opened in update/exclusive mode have back-out protection. It cancels all changes made by this transaction and then acts like a 'Transaction Finished' call. The 'Abort' call must be used if an error occurred during a modifying access.
3) an 'Abort' call, which detaches all the records attached to the transaction, thus allowing existing queued reads to be performed by other transactions. However, for protected files no new queueing on these records is allowed, and an error status is returned with each read performed in an existing queue. Back-out information (if any) is kept.

4) a System Breakdown, i.e. an error of such severity that the system must be restarted. All files are left in the condition they were in when the breakdown occurred.

Recovery is clearly required in the last two cases.

RUNS

A run is initiated by the user giving the EDF Processor Command INSE, and is terminated in one of three ways:
1) issuing another INSE command (thus starting a new run);
2) system closedown and power-off;
3) system breakdown.

Each run is identified by a run identifier, supplied with the INSE command. All run identifiers must be unique within the system, i.e. back-up (and possibly back-out) information relating to two runs with the same identifier is not allowed. For each run the identifier and other information is written into the back-up file.

ESTABLISHING RECOVERY MECHANISMS

The user will, for each TDFM file in the system, specify what level of protection he requires for that file; he will do this in the FILE command when the file is first created. If he wishes, he may change this level of protection at a later date by means of the SPRO command.

If the user is going to access protected files (other than in read-only mode), he will need a back-up file and possibly also a back-out file; these are created with the BUGN and BOGN commands, respectively, and may thereafter be used until they are full or until an unrecoverable error occurs. At any one time the system recognises one back-up and one back-out file, as specified (by their filecodes) in an INSE command; each file provides the appropriate protection for all the TDFM files known to the system.

Finally, the user needs a security copy of each TDFM file, since the recovery mechanism is predicated on this. Security copies should be taken at times when the user is sure that the file is consistent, i.e. just after recovery, and at regular intervals such as at the start of each day, depending on the amount of update activity expected. Either the DEL and SAVE commands or a series of COPY commands may be used to make the copy.

RECOVERING TDFM FILES

Recovery is required either after a system breakdown or when a significant percentage of transactions on a file have had to be aborted. It is always performed at the start of a run; no other concurrent TDFM activity is allowed.

Recovery operates on all protected TDFM files simultaneously; it is not possible to recover just one of a number of files.

The user begins recovery by restoring each TDFM file from its latest copy; see "TDFM File Housekeeping" in Chapter 3 of this Part for the various methods of restoring files. A filecode is assigned to the back-up file, and to the back-out file if required, in the normal way (ASG command). Then the EDF Processor RBUP command is issued; this first performs back-up recovery on a TDFM file, then if back-out security applies to the file the relevant changes are undone. This action is repeated for all files. (In fact all files are recovered concurrently, and unnecessary back-up changes are not performed; the end effect is the same.)

When the RBUP command has finished processing, the back-out file is empty and the information in the back-up file, relating to update accesses which have been backed-out, is erased. Both files may be used for further runs, if desired, unless some error has occurred while accessing one of these files, in which case it should be deleted and recreated.

If the recovery procedure has had to be redone with the NRUN parameter to the RBUP command, the only option open to the user is to repeat all transactions which could not be recovered. The back-up and back-out files should both be deleted and recreated before starting this procedure.

The following example illustrates the various procedures required to create, load, copy and restore a TDFM file.

The TDFM file is to contain at most 200 records, each with length 50 characters and the following keys:

1) BKEY, a binary number held in two bytes;
2) NAME, the name of the person, held in 12 bytes;
3) COUNTR, the country in which the person resides, held in 12 bytes;
4) PROF, the person's profession or trade, held in 12 bytes.

BKEY is the primary key, and bijective; the remaining keys are homonymous. Data fields also exist, containing the person's age, address etc.

Creating and Loading a TDFM File

In the system machine:

```
DCB 16                    - declare a batch machine
FCD /01,TY
FCD /E0,TY
FCD /C0
FCD /C1
FCD /F0,/C0,SUPERV
FCD /F1,/C1,WB200
FCD /F2,/C0,WBR
DEN
DCF WBT,1                 - declare a foreground machine
CMA 2,2000
SEG 1,10
FCD 1,TY0A
FCD 2,LP
FCD /E0,TY0A
FCD /C0
FCD /C1
FCD /F0,/C0,SUPERV
FCD /F1,/C1,WB200
FCD /F2,/C0,WBR
DEN
BYE BATCH
SB
MACH-ID:BATCH      DATE:
:JOB USID=WB
ASG FCOD=/D6,DAD=/F1,TYPE=UF,NBGR=2
ASG FCOD=/D7,DAD=/F1,TYPE=UF,NBGR=2
EDF
FILE FNAM=FEXD,DAD=/F1,USID=WB,NKEY=4,NDAT=1,MREC=200,SECU=BU
KEY KNAM=BKEY,DAD=/F1,USID=WB,KPOS=48,KLGT=2,KPAD=/00,BIJ
KEY KNAM=NAME,DAD=/F1,USID=WB,KPOS=0,KLGT=12,KPAD=/20
KEY KNAM=COUNTR,DAD=/F1,USID=WB,KPOS=25,KLGT=12,KPAD=/20
KEY KNAM=PROF,DAD=/F1,USID=WB,KPOS=37,KLEN=12,KPAD=/20
DATA FNAM=FEXDAT,DAD=/F1,USID=WB,NBGR=3
LOAD ONAM=FEXD,ODAD=/F1,USID=WB,ICOD=/05,TYPE=CONT,SEP='!!',IGEN=YES,FREE=50
EFEN
```

Saving and Using a File

In (the same) batch machine:

```
SCR FCOD=/D6               - delete filecodes no longer required
SCR FCOD=/D7
ASG FCOD=/10,DAD=/F2,TYPE=UF,FNAM=BACKUP
ASG FCOD=/13,DAD=/F2,TYPE=EF,FNAM=FEXD
EDF
SAVE FNAM=FEXD,IDAD=/F1,IUSI=WB,ODAD=/F2,OUSI=WB
BUGN FNAM=BACKUP,DAD=/F2,USID=WB,NBGR=6
INSE IDEN='19AUG 10:15',BUFC=/10
EDF=FREE B-UP SEC.= nnnnn
EFEN
:EOB

SM WBT
MACH-ID:WBT         DATE:
...
RUN <program> -     Interactive access to TDFM file FEXD, using filecode /13 for
                    the purpose.
...
BYE BATCH
SB
:JOB USID=WB
EDF
INSE IDEN='19AUG 11:30',BUFC=/10    - end previous run, start new run.
EDF=FREE B-UP SEC.= nnnnn
EFEN
:EOB
SM WBT
...
RUN <program> - as before.
...
```

Restoring a File

If an IPL was necessary, the machine definitions given above should be repeated.

In batch machine:

```
EDF
REST FNAM=FEXD,IDAD=/F2,IUSI=WB,ODAD=/F1,OUSI=WB
INSE IDEN='19AUG REC1',BUFC=/10
EDF=FREE B-UP SEC.= nnnnn
RBUP BUFC=/10
<messages from RBUP>
INSE IDEN='19AUG 13:20',BUFC=/10
EDF=FREE B-UP SEC.= nnnnn
DEL FNAM=FEXD,DAD=/F2,USID=WB
SAVE FNAM=FEXD,IDAD=/F1,IUSI=WB,ODAD=/F2,OUSI=WB
EFEN
:EOB
SM WBT
...           - continue running foreground program.
```