FL4REC-API

1

Generated by Doxygen 1.8.14

Contents

Chapter 1

FLAM4 Record Interface

This interface provides a platform independent record-oriented read and write access to **FLAM4** files. It contains all functions of the Windows, UNIX and HOST record interfaces to provide backward compatibility, required for sequential access. On Mainframes additional load modules are available for positioning, insert and update of records.

Each function is also available as a separate load module. On open platforms the function names are in lower case with '_' as prefix and 1 as postfix (_flmxxx1), on mainframes in upper case (FLMXXX). The libfl4recuc.dll/so provides the functions in upper case on Windows and Unix systems. This documentation use 'flmxxx' in lower case for each entry.

All parameters are call-by-reference and no return value will be used. There are 4 types of parameters:

```
POINTER:   pointer to an address (usually 32 or 64 bit)
INTEGER:   pointer to a 32 bit number in two's complement
STRING[x]: pointer to a byte (8 bit) array of length x
STRING:    pointer to a variable length byte (8 bit) array
```

It provides functions similar to the record-oriented file I/O functions used with COBOL, PL/I or assembler on mainframes. Only the file open function differs in that it gives 3 different open function and a few setter (or FLMSET()) which must be called in the right sequence, to open a FLAMFILE of version 4. To simplify the use of FKME (FLAM key management extensions) we have add the callable service FLMKME, which works like FLMPWD.

FL4REC consists of a number of subroutines that can be called by any programming language such as COBOL, PL/I, C, FORTRAN, etc., as well as by ASSEMBLER programs. Except for the key descriptions all parameters are implemented as elementary data types (INTEGER, STRING). Deliberately no control blocks are required to avoid alignment problems and additional copying of parameter values before and after a function call. Key descriptions are organized as a structured data type in order to shorten the parameter list.

Application programs written in C must include the header `FL4RECUC.h`. This C header file contains the definitions of the symbolic constants, as well as the structure of the key description. These definitions must be ported to other programming languages accordingly if C is not used.

1.1 Function arguments

All argument lists begin with an ID, which identifies the compression file uniquely. This flmid argument contains the address of the work area for the compressed file, which was specified by flmopn(), and must not be modified until flmcls(). All other arguments are only relevant to the function to which they are transferred.

The identification is followed by a return code that informs the caller about successful execution or occurring errors. Processing of a compressed file always starts with function FLMOPN that assigns the program to the compressed file and defines the operation mode. A file opened successfully must always be closed with function FLMCLS. There are no messages generated at the record level interface. During transfer of original data the parameter RECORD always contains the true data without any length fields or record delimiters. Or the parameter RECPTR points to a field with such a content. The parameter RECLEN always contains the length of the true data (exclusive length).

COBOL programs can be translated using the **DYNAM** option. As a result, the FLAM modules are loaded from the library only at the moment of execution. If a dynamic call is not wanted (**NO-DYNAM** option in COBOL or V constants in ASSEMBLER), the FLAM module FL4REC should be specified explicitly when linking.

1.2 Micro Focus EDZ support

The API is supported also in MF-EDZ environments. See install.txt on Windows or Unix for more information.

Functions passing pointer adresses, for example flmloc(), only work if the cobol compiler directiv AMODE is NOT used. With AMODE mainframe pointers are used in the cobol program which cannot be converted.

For use with Micro Focus Enterprise Server the following environment variables might be used.

FLAM4MF=unset: The filename needs to be specified as either `DD:name` or `//'name'` if the Micro Focus support is needed. Otherwise normal **FLAM4** use is done.
If `DD:name` or `//'name'` cannot be accessed an error is returned.

FLAM4MF=encoding: Micro Focus libraries will be used and an error is returned if they are not found.
The encoding is used to automatically convert all input character strings from this to the local system encoding. The output character strings are converted from the local system to this encoding.
The command `flcl info get.enc` prints a list of all supported encodings.
For convenience the strings `:EBCDIC` or `:ASCII` might be used. If the encoding strings begins with 'IBM' big endian is assumed and binary values will be byte swapped.

FLAM4MF=YES: Micro Focus libraries will be used and an error is returned if they are not found.
No character conversion is done.

FLAM4MF=NO: Micro Focus support is switched off, standard **FLAM4** use.
This must be done in order to work with filenames starting with `DD:`

For tracing:

FLAM4MF_TRACEFILE=unset Without trace file name tracing of library calls is omitted.

FLAM4MF_TRACEFILE=filename trace output of library function calls is written to `filename`

For system symbols:

FLAM4MF_STATIC_SYSVAR=filename to define static system variables as environment variables.

FLAM4MF_STATIC_SYSVAR=filename to use the JCL user exit to define dynamic system symbols

For dynamic system variables the DD:SYSVAR is also supported.

To use the function in COBOL the libfl4recuc.dll/so must be copied in the working directory of the EDZ server under FLMOPN.dll/so and FLMERR.dll/so.

1.3 Return codes

Return codes are mainly the same between HOST and other platforms but in the higher values a few differences exists. This is mainly in cases where the return code is more the reason code for an error. To simplify the error handling a function (FLMERR) which provides the correct error message independent of the platform are add to this interface. Below you can find a list of important FLAM4 return codes. This list covers any return code to control the program, if another return code occur this will mainly a severe error to terminate execution.

0		No error, success
1		A record has been shortened to the length of the record buffer.
2		The end of the file has been reached while reading; no data is transferred.
3		A gap has been found in a relative file; the record length is zero.
4		When a record is converted to fixed format, it is padded with fill characters.
5		A key is missing when reading from or is invalid when writing to an index sequential file.
6		When reading in a group file, a new file is starting; no data is transferred.
7		Password / cryptokey missing on decompression.
8		Parameter or function not supported.
9		When compressing with the statistics switched on, FLAMUP or FLAM reports that the compressed file is larger than the original file (expansion).
10		During decompression, the input file has not been recognized as being a FLAM compressed file.
11		The format of the FLAMFILE is wrong.
14		The checksum of a compressed record is wrong.
22		Invalid compression method.
29		Password missing or invalid (passed by FLMPWD).
30		Input file is empty.
31		Input file does not exist.

1.4 Programming the Record Interface at Compression

The process of creating a FLAMFILE can be divided into three phases:

- open sequence
- one or more compression cycles
- termination (function flmcls())

The open sequence always starts calling the flmopn() function. Subsequently, flmopd() and/or flmopf() can be called if flmopn() was called with continue_param set. When both functions are called, flmopd() must come first and also have continue_param set.

When flmopd() or flmopf() are not called by the application program, FLAM uses implicit settings to generate such calls internally. Default values used with the flam command and flamup calls are not effective here.

Finally, if encryption is used, a password must be provided by a flmpwd() call.

In the second phase, one compression cycle must be done for each file being compressed.

Such a cycle would normally begin with a flmphd() call to generate a common FLAM fileheader. Only when a single file is compressed without encryption, this may be omitted. If a FLAM fileheader was created, a user-specific header

may be appended by invoking flmpuh(). With Secure FLAMFILEs, this invocation is mandatory even if the header data length is 0.

Now data can be submitted for compression by repeated execution of flmput(). Each such call passes a record in the FLAM terminology, which can be later replicated identically or with modified record attributes thanks to the structure information kept by FLAM. FLAM collects the data passed to it in the compression buffer and controls compression and the resulting output without involving the application program.

A compression cycle is terminated by a call to flmflu() which causes the remainder in the compression buffer to be compressed and the output of the result. This function also returns statistical information to the application. Following this, a new compression cycle may be initiated by another flmphd() or the FLAMFILE can be closed by calling flmcls().

Control is always returned to the invoking program. There are no error exits and no error messages are generated by the record interface. Rather, a return code is passed back to the application.

1.5 Programming the Record Interface at Decompression

As with compression, decompressing a FLAMFILE also consists of three phases:

- open sequence
- decompression cycle(s)
- termination (function flmcls())

The open sequence here consists of the same function calls as at compression. Only the direction of the flow of information is reversed for some of the arguments, e.g. the compression mode. They are returned to the application.

Every function used in the compression cycle has its decompression counterpart that performs the complementary operation. The complements for flmphd() and flmpuh() are flmghd() and flmguh(), respectively, that for flmput() is flmget(). A decompression cycle is also terminated with flmflu(). In addition, there is a function, flmpos(), which advances the read position in a FLAMFILE to the beginning of the next original file. This function has no counterpart with compression.

Typically, a decompression cycle initially invokes flmghd() which returns details about the original file such as file name, record format, etc. This may be followed by flmguh() to retrieve the user-specific header, if present. Subsequent flmget() calls retrieve one record per call, with the last record of an original file being signalled by a special return code. Skipping the remaining records of an original file can be done by flmpos() which positions to the next file header. A flmflu() terminates a decompression cycle that may be followed by another cycle or by flmcls(), which terminates the processing. One flmflu() or flmpos() per cycle is mandatory, all other calls are optional.

Some of the function are not documented. These functions are for internal use only. The setter for FLMSET are also not documented because these entries are not available as load module. To simplify C programming these functions can be used, the functionality is explained over FLMSET. FLMSET is only available on mainframes, to define additional host specific parameters after FLMOPN.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 key_description Struct Reference

Key elements for index sequential data access.

`#include <FL4REC.h>`

Data Fields

- U32 KEYFLG

  Key flag.
- U32 KEYCNT

  Number of key elements.
- FKE KEYELM [8]

  Array of key elements.

4.1.1 Detailed Description

Key elements for index sequential data access.

Defines up to 8 key elements for index sequential data access. Such access methods are only supported on mainframe systems (BSAM/VSAM).

4.1.2 Field Documentation

4.1.2.1 KEYFLG

```
U32 key_description::KEYFLG
```

Key flag.

Options:

- 0 = No duplicate key (DEFAULT)
- 1 = Duplicate keys allowed

4.1.2.2 KEYCNT

```
U32 key_description::KEYCNT
```

Number of key elements.

The number of elements in key_description::KEYELM (normally 1 (VSAM), maximum 8 (BSAM))

4.1.2.3 KEYELM

```
FKE key_description::KEYELM[8]
```

Array of key elements.

The documentation for this struct was generated from the following file:

- FL4REC.h

4.2 key_element Struct Reference

Defines a key element for index sequential access methods (VSAM-KSDS)

```
#include <FL4REC.h>
```

Data Fields

- U32 KEYPOS
  Key position.
- U32 KEYLEN
  Length of the key in the record (minimum 1, default 8)
- U32 KEYTYP
  Data type:

4.2.1 Detailed Description

Defines a key element for index sequential access methods (VSAM-KSDS)

4.2.2 Field Documentation

4.2.2.1 KEYPOS

```
U32 key_element::KEYPOS
```

Key position.

Byte offset where the key begins (starting at 1, default 1)

4.2.2.2 KEYLEN

```
U32 key_element::KEYLEN
```

Length of the key in the record (minimum 1, default 8)

4.2.2.3 KEYTYP

```
U32 key_element::KEYTYP
```

Data type:

- 0 = printable characters
- 1 = binary data

The documentation for this struct was generated from the following file:

- FL4REC.h

Chapter 5

File Documentation

5.1 FL4REC.h File Reference

Specification of FLAM4 Record Interface (with SSH support)

Data Structures

- struct key_element

  Defines a key element for index sequential access methods (VSAM-KSDS)
- struct key_description

  Key elements for index sequential data access.

Typedefs

- typedef int I32

  signed 32 bit integer
- typedef unsigned int U32

  unsigned 32 bit integer
- typedef long long int I64

  signed 64 bit integer
- typedef unsigned long long int U64

  signed 64 bit integer
- typedef char C08

  signed 8 bit integer
- typedef unsigned char U08

  unsigned 8 bit integer
- typedef struct key_element FKE

  Defines a key element for index sequential access methods (VSAM-KSDS)
- typedef struct key_description FKD

  Key elements for index sequential data access.

Enumerations

- enum flmset_parameter {
  FLMSET_SPLITMODE = 1, FLMSET_SPLITNUMBER = 2, FLMSET_SPLITSIZE = 3, FLMSET_PRIMARY_SPACE = 4,
  FLMSET_SECONDARY_SPACE = 5, FLMSET_VOLUME = 6, FLMSET_UNIT = 7, FLMSET_DATA_CLASS = 8,
  FLMSET_STORAGE_CLASS = 9, FLMSET_MANAGEMENT_CLASS = 10, FLMSET_DISPOSTION_STATUS = 11, FLMSET_DISPOSITION_NORMAL = 12,
  FLMSET_DISPOSITION_ABNORMAL = 13, FLMSET_CRYPTOMODE =2001, FLMSET_SECUREINFO =2002 }

  The flmset_parameter enum defines constants for the flmset() function.
- enum flmget_parameter {
  FLMGET_SPLITMODE = 1, FLMGET_SPLITNUMBER = 2, FLMGET_SPLITSIZE = 3, FLMGET_CRYPTOMODE =2001,
  FLMGET_SECUREINFO =2002, FLMGET_MODE =4001, FLMGET_RLCOMPMODE =6001 }

  The flmget_parameter enum defines constants for the flmqry() function.

Functions

- void flmopn (char **flmid, I32 *rc, I32 *lstpar, I32 *opmode, C08 *filename, I32 *statis)

  open a FLAMFILE
- void flmopd (char **flmid, I32 *rc, I32 *lstpar, I32 *namelen, C08 *filename, I32 *org, I32 *recf, I32 *recs, U08 *recdel, FKD *keydesc, I32 *blks, I32 *cldisp, I32 *dev)

  set/get FLAMFILE organisation attributes
- void flmopf (char **flmid, I32 *rc, I32 *version, I32 *flcode, I32 *modus, I32 *maxbuf, I32 *header, I32 *maxrec, FKD *keydesc, I32 *blkmode, U08 *exk20, U08 *exd20)

  set/get FLAMFILE attributes
- void flmflu (char **flmid, I32 *rc, U32 *cputime, U32 *recin, U32 *bytin, U32 *bytoflin, U32 *recout, U32 *bytout, U32 *bytoflout)

  flush FLAMFILE
- void flmcls (char **flmid, I32 *rc, U32 *cputime, U32 *recin, U32 *bytin, U32 *bytoflin, U32 *recout, U32 *bytout, U32 *bytoflout)

  close FLAMFILE
- void flmghd (char **flmid, I32 *rc, I32 *namelen, C08 *filename, I32 *org, I32 *recf, I32 *recs, U08 *recdel, FKD *keydesc, I32 *blks, I32 *prnctr, C08 *system)

  get file header
- void flmphd (char **flmid, I32 *rc, I32 *namelen, C08 *filename, I32 *org, I32 *recf, I32 *recs, U08 *recdel, FKD *keydesc, I32 *blks, I32 *prnctr, C08 *system, I32 *cont)

  put file header
- void flmguh (char **flmid, I32 *rc, I32 *hdrlen, C08 *header)

  get user header
- void flmpuh (char **flmid, I32 *rc, I32 *hdrlen, C08 *header)

  put user header
- void flmpwd (char **flmid, I32 *rc, U32 *pwdlen, C08 *passwd)

  Sets a password for encryption/decryption while compressing/decompressing.
- void flmkme (char **flmid, I32 *rc, U32 *kmclen, U08 *kmc, const U32 *liblen, const C08 *lib, const U32 *fuclen, const C08 *fuc, const U32 *parlen, const C08 *par, U32 *msglen, C08 *msg, U32 *inflen, C08 *inf)

  load and use FKME
- void flmpos (char **flmid, I32 *rc, I32 *pos)

  set read position
- void flmget (char **flmid, I32 *rc, I32 *reclength, U08 *record, I32 *buflength)

  get next record

- void flmloc (char **flmid, I32 *rc, I32 *reclength, U08 **record)
  locate record
- void flmput (char **flmid, I32 *rc, I32 *reclength, U08 *record)
  put record
- void flmerr (I32 *rc, I32 *msglen, const C08 **errmsg)
  provides an error message
- void flmqry (char **flmid, I32 *rc, I32 *parameter, void *value)
  query extended FLAMFILE attributes
- void flmset (char **flmid, I32 *rc, I32 *parameter, void *value)
  set extended FLAMFILE attributes

5.1.1 Detailed Description

Specification of FLAM4 Record Interface (with SSH support)

The FLAM4 record interface only uses variables whose types are given in the Typedef section

Author

limes datentechnik gmbh

Date

24.2.2016

Copyright

5.1.2 Typedef Documentation

5.1.2.1 I32

```
typedef int I32
```

signed 32 bit integer

5.1.2.2 U32

```
typedef unsigned int U32
```

unsigned 32 bit integer

5.1.2.3 I64

```
typedef long long int I64
```

signed 64 bit integer

5.1.2.4 U64

```
typedef unsigned long long int U64
```

signed 64 bit integer

5.1.2.5 C08

```
typedef char C08
```

signed 8 bit integer

5.1.2.6 U08

```
typedef unsigned char U08
```

unsigned 8 bit integer

5.1.2.7 FKE

```
typedef struct key_element FKE
```

Defines a key element for index sequential access methods (VSAM-KSDS)

5.1.2.8 FKD

```
typedef struct key_description FKD
```

Key elements for index sequential data access.

Defines up to 8 key elements for index sequential data access. Such access methods are only supported on mainframe systems (BSAM/VSAM).

5.1.3 Enumeration Type Documentation

5.1.3.1 flmset_parameter

```
enum flmset_parameter
```

The flmset_parameter enum defines constants for the flmset() function.

Enumerator

FLMSET_SPLITMODE	split mode 0: none, 1: serial, 2: parallel
FLMSET_SPLITNUMBER	split number 2 - 4
FLMSET_SPLITSIZE	split size in MiB 1 - 4095
FLMSET_PRIMARY_SPACE	size of primary in MiB 1 - 4095
FLMSET_SECONDARY_SPACE	size of secondary space in MiB 1 - 4095
FLMSET_VOLUME	volume name 1-8 character
FLMSET_UNIT	unit name 1 - 8 character
FLMSET_DATA_CLASS	data class name 1 - 8 character
FLMSET_STORAGE_CLASS	storage class name 1 - 8 character
FLMSET_MANAGEMENT_CLASS	management class name 1 - 8 character
FLMSET_DISPOSTION_STATUS	disposition status 0: Default, 1: NEW, 2: OLD, 3: SHR, 4: MOD
FLMSET_DISPOSITION_NORMAL	normal disposition 0: Default, 1: DELETE, 2: KEEP, 3: CATLG, 4: UNCATLG
FLMSET_DISPOSITION_ABNORMAL	abnormal disposition 0: Default, 1: DELETE, 2: KEEP, 3: CATLG, 4: UNCATLG
FLMSET_CRYPTOMODE	mode of encryption 0: none, 1: FLAM, 2: AES
FLMSET_SECUREINFO	secure info 0: no, 1: yes, 2: ignore, 3: member

5.1.3.2 flmget_parameter

```
enum flmget_parameter
```

The flmget_parameter enum defines constants for the flmqry() function.

Enumerator

FLMGET_SPLITMODE	split mode 0: none, 1: serial, 2: parallel
FLMGET_SPLITNUMBER	split number 2 - 4
FLMGET_SPLITSIZE	split size in MiB 1 - 4095
FLMGET_CRYPTOMODE	mode of encryption 0: none, 1: FLAM, 2: AES
FLMGET_SECUREINFO	secure info 0: no, 1: yes, 2: ignore, 3: member
FLMGET_MODE	general mode 0: deco/read 1: comp/write
FLMGET_RLCOMPMODE	real compression mode 3: ADC 8: ZEDC

5.1.4 Function Documentation

5.1.4.1 flmopn()

```
void flmopn (
            char ** flmid,
```

```
                I32 * rc,
                I32 * lstpar,
                I32 * opmode,
                C08 * filename,
                I32 * statis )
```

open a FLAMFILE

The flmopn() function opens a FLAMFILE. It can be followed by a flmopd() and/or flmopf() call, in order to set or query the attributes of the FLAMFILE or the FLAM settings.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
lstpar	Set to a non-zero value if you will call flmopd() and/or flmopf() afterwards
opmode	0=INPUT(read), 1=OUTPUT(write), 2=INOUT(read/write, Host-only)
filename	Zero-terminated filename of file to open (or DD name on mainframes)
statis	Indicates whether to collect statistical data that is returned on flmcls() or flmflu() (0=disable, other=enable)

5.1.4.2 flmopd()

```
void flmopd (
                char ** flmid,
                I32 * rc,
                I32 * lstpar,
                I32 * namelen,
                C08 * filename,
                I32 * org,
                I32 * recf,
                I32 * recs,
                U08 * recdel,
                FKD * keydesc,
                I32 * blks,
                I32 * cldisp,
                I32 * dev )
```

set/get FLAMFILE organisation attributes

flmopd() allows to set and/or get file organistation attributes of the FLAMFILE. Use of flmopd() is optional but has to be done after flmopn#() and before flmopf().

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
lstpar	Set to a non-zero value if you will call flmopf() afterwards
namelen	Pass the length of the `filename` buffer; contains the filename length after this call

Parameters

filename	Buffer used to output the filename of the file to be opened (on mainframes the file name allocated to the DD name)
org	organisation of FLAMFILE 0,8,16=sequential(PS,ESDS) 1,9,17=index-sequential(IS,KSDS) 2,10=RRDS 3,11=LDS
recf	Record format for FLAMFILE 0,8,16=variable 1,9,17=fixed(default) 2,10,18=undefined
recs	maximum record length for FLAMFILE 80 - 32760, 512=default (for CX7 the maximum is 4095)
recdel	Record delimiter for FLAMFILE records (only for CX7 on WIN/UNIX)
keydesc	pointer to key description for the original records see #struct kd (mainframe only)
blks	Blocksize for the FLAMFILE (only for mainframes) 0=unblocked or 80-32760
cldisp	Close disposition (only for mainframes) 0=rewind(default) 1=unload 2=leave/retain
dev	Device type 0=disc(default) 7=user defined I/O functions are used (for mainframes: 0,8,16=disc(default) 1,9,17=tape 2,10,18=floppy 3,11,19=streamer 7,15,23=user defined I/O functions are used)

5.1.4.3 flmopf()

```
void flmopf (
              char ** flmid,
              I32 * rc,
              I32 * version,
              I32 * flcode,
              I32 * modus,
              I32 * maxbuf,
              I32 * header,
              I32 * maxrec,
              FKD * keydesc,
              I32 * blkmode,
              U08 * exk20,
              U08 * exd20 )
```

set/get FLAMFILE attributes

Can be used to define (when compressing) or retrieve (when decompressing) the attributes of the compressed file. flmopf() can be called as second function after flmopn(), when flmopd() is not used, or as third function after flmopd(). For encrypted FLAMFILES on mainframes the function FLMSET() must be used, other platforms use bit 4 and 5 of the `modus` parameter to specify the encryption method: FLAM and AES

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
version	of FLAMFILE: always 2
flcode	During compression, specifies the character set to use to write to the FLAMFILE. This information is returned during the decompression procedure. 0=EBCDIC, 1=ASCII(default)
modus	compression method: 0=CX8 1=CX7 2=VR8 3=ADC 5=NDC
maxbuf	size of matrix buffer 2048 - 262144 default=32768
header	option flag for header generation/existence: 0=no 1=yes

Parameters

maxrec	maximum number of records in matrix: limits are 255 for CX8,CX7,VR8 and 4095 for ADC
keydesc	pointer to key description for the original records see #struct kd (mainframe only)
blkmode	method of matrix storage: 0=unblocked 1=blocked(default, data from more than 1 matrix might be in 1 block)
exk20	name of user exit for output of compressed data (unused / z/OS only)
exd20	name of user exit for input of compressed data (unused / z/OS only)

5.1.4.4 flmflu()

```
void flmflu (
              char ** flmid,
              I32 * rc,
              U32 * cputime,
              U32 * recin,
              U32 * bytin,
              U32 * bytoflin,
              U32 * recout,
              U32 * bytout,
              U32 * bytoflout )
```

flush FLAMFILE

flmflu() finalizes the compression procedure for a file The remaining records in the block are compressed and the last FLAM record is padded where necessary. The compressed file is not closed after flmflu() and the statistics counters are not reset, i.e. another piece of compressed data can be appended. With Secure FLAMFILEs, a member trailer is appended.

Statistical information is returned if it has been requested at the corresponding flmopn() call (statis != 0).

When compressing Secure FLAMFILEs, flmflu() is permitted only after flmpuh() or flmput(). When decompressing Secure FLAMFILEs, flmflu() must be preceded by either flmget(), flmghd(), or flmguh().

Parameters

flmid	Identifies the FLAMFILE to be flushed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
cputime	Consumed CPU time in units of time as returned by clock()
recin	If flmopn() was called with `statis` != 0, this argument returns the number of decompressed records which have been processed by flmput() or flmget(). The accumulated values of this counter are only significant if complete files are processed.
bytin	If flmopn() was called with `statis` != 0, this argument returns the number of bytes of the decompressed records which have been transferred with flmput() or flmget(). The accumulated value of this counter is only significant if complete files are processed.
bytoflin	If the accumulated value of the `bytin` argument exceeds 2,000,000,000, this counter is used for multiples of 2,000,000,000.
recout	If flmopn() was called with `statis` != 0, this argument returns the number of compressed records which were created during the compression procedure or read during the decompression procedure. This value is only significant if complete files are processed.
bytout	If flmopn() was called with `statis` != 0, this argument returns the number of bytes which were
	created during the compression procedure or read during the decompression procedure. This value is only significant if complete files are processed.
bytoflout	If the accumulated value of the `bytout` argument exceeds 2,000,000,000, this counter is used for multiples of 2,000,000,000.

5.1.4.5 flmcls()

```
void flmcls (
             char ** flmid,
             I32 * rc,
             U32 * cputime,
             U32 * recin,
             U32 * bytin,
             U32 * bytoflin,
             U32 * recout,
             U32 * bytout,
             U32 * bytoflout )
```

close FLAMFILE

The flmcls() function finalizes the record interface.

If compression is closed, the last matrix is compressed and the compressed data is written to the FLAMFILE. Additional information (byte-, record counter, MACs) are stored as an 'ending record', if required (SECUREIN↩ FO=YES). Then the FLAMFILE is closed.

When decompression is closed, only the FLAMFILE is closed. Additional records not yet read from the FLAMFILE are discarded.

Statistical information is returned if it has been requested at the corresponding flmopn() call (statis != 0).

With Secure FLAMFILEs, flmcls() may only be invoked immediately after flmflu() or flmpos().

Parameters

flmid	Identifies the FLAMFILE to be closed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
cputime	Consumed CPU time in units of time as returned by clock()
recin	If flmopn() was called with `statis` != 0, this argument returns the number of decompressed records which have been processed by flmput() or flmget(). The accumulated values of this counter are only significant if complete files are processed.
bytin	If flmopn() was called with `statis` != 0, this argument returns the number of bytes of the decompressed records which have been transferred with flmput() or flmget(). The accumulated value of this counter is only significant if complete files are processed.
bytoflin	If the accumulated value of the `bytin` argument exceeds 2,000,000,000, this counter is used for multiples of 2,000,000,000.
recout	If flmopn() was called with `statis` != 0, this argument returns the number of compressed records which were created during the compression procedure or read during the decompression procedure. This value is only significant if complete files are processed.
bytout	If flmopn() was called with `statis` != 0, this argument returns the number of bytes which were created during the compression procedure or read during the decompression procedure. This value is only significant if complete files are processed.
bytoflout	If the accumulated value of the `bytout` argument exceeds 2,000,000,000, this counter is used for multiples of 2,000,000,000.

5.1.4.6 flmghd()

```
void flmghd (
              char ** flmid,
              I32 * rc,
              I32 * namelen,
              C08 * filename,
              I32 * org,
              I32 * recf,
              I32 * recs,
              U08 * recdel,
              FKD * keydesc,
              I32 * blks,
              I32 * prnctr,
              C08 * system )
```

get file header

flmghd() allows retrieving the file attributes of the original file during decompression if they were stored in the FLAMFILE during compression. If the FLAMFILE contains several file headers (see flmphd()), the last file header found by FLAM is returned.

The first file header is usually available immediately after flmopn() (see flmopf() header=1). When FLAM recognizes additional file headers, it will inform the user via the return code (RETCO=6) of flmget() or flmloc().

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
namelen	Pass the length of the `filename` buffer in which the name of the original file will be returned. Is set to the actual length when function returns.
filename	Buffer area in which to store the original filename. If the buffer is too small, the filename will be truncated to the length of the buffer.
org	returns a value for the organization of the original file. Possible values (not all are supported on Unix): 0=sequential, 1=index-sequential, 2=relative, 3=direct access, 4=no record structure, 5=library 6=physical
recf	returns a value for the record format of the original file. Possible values (not all are supported on Unix): 0/8/16/24 = variable / blocked / blocked/spanned / VFC; 1/9/17 = fix / blocked / blocked/Standard; 2/10/18 = undefined / / EAF; 3/11/19 = stream / text delimiter / length fields
recs	This argument returns the record size of the original file. For variable record sizes and the stream record format, the value is 0.
recdel	This argument specifies the record delimiter for the stream record format.
keydesc	Key description (mainframe only)
blks	Block length of original file (0=unblocked)
prnctr	Printer control characters (0=none, 1=ASA, 2=Machine specific)
system	This argument consists of two bytes and is used to return a code for the operating system in which the FLAMFILE was created.

5.1.4.7 flmphd()

```
void flmphd (
```

```
            char ** flmid,
            I32 * rc,
            I32 * namelen,
            C08 * filename,
            I32 * org,
            I32 * recf,
            I32 * recs,
            U08 * recdel,
            FKD * keydesc,
            I32 * blks,
            I32 * prnctr,
            C08 * system,
            I32 * cont )
```

put file header

The flmphd() function is only allowed during compression and if header=1 was set with flmopf(). It generates a common file header, which describes the file format of the original records that are transferred subsequently. If several different files are compressed into a single FLAMFILE, a separate file header can be generated for each file with the flmphd() function. FLAM returns this file header information during the decompression procedure if it has been requested to do so (flmghd()).

Calls to flmphd() with `continue_param!=0` or after flmpwd() must be followed immediately by flmpuh().

Using SECUREINFO=YES, function flmphd() is mandatory!

Note: The parameter in flmphd() also controls the construction of an index-sequential FLAMFILE. On DSORG=0 (sequential data), a record number is created and used as a record key; on DSORG=1 (index-sequential) the original key is used.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
namelen	Length of the `filename`
filename	Buffer containing the original filename
org	Code for the organization of the original file. For permissible values, see the flmghd() function.
recf	Code for the record format of the original file. For permissible values, see the flmghd() function.
recs	The maximum record size of the original file
recdel	Record delimiter
keydesc	Key description (only for mainframes)
blks	Block length of original file (0=unblocked)
prnctr	Printer control characters (0=none, 1=ASA, 2=Machine specific)
system	This argument consists of two bytes and is used to return a code for the operating system in which the FLAMFILE was created. For a list of valid values, see flmghd().
cont	Indicates whether a call to flmpuh() will follow (0=no, 1=yes)

5.1.4.8 flmguh()

```
void flmguh (
            char ** flmid,
```

```
            I32 * rc,
            I32 * hdrlen,
            C08 * header )
```

get user header

Retrieves the data from the user-specific FLAM file header. Calling this function is only allowed during decompression and immediately following an flmghd() call.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
hdrlen	Pass the length of the buffer in which the user-specific file header will be returned. Is set to the actual length when function returns.
header	Buffer to write the user-specific file header data to

5.1.4.9 flmpuh()

```
void flmpuh (
            char ** flmid,
            I32 * rc,
            I32 * hdrlen,
            C08 * header )
```

put user header

The flmpuh() function adds a user-specific header with information of arbitrary structure to the FLAMFILE. This information is added in binary form and can be retrieved during decompression with flmguh() (the complementary function).

Calling this function is only allowed during compression and immediately following an flmphd() call.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
hdrlen	Length of the user-specific `header` data (max. 3500 (Host, 8-bit), 1750 (Host, 7-bit, CX7), 32732 bytes (other))
header	User-specific header data

5.1.4.10 flmpwd()

```
void flmpwd (
            char ** flmid,
```

```
            I32 * rc,
            U32 * pwdlen,
            C08 * passwd )
```

Sets a password for encryption/decryption while compressing/decompressing.

It is the first call after the last FLMOPx function during encryption.

The encryption mode is set by the function flmset(). On decryption, the information is read from the FLAMFILE and can be obtained by calling flmqry().

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
pwdlen	The password's length in characters
passwd	The password string

5.1.4.11 flmkme()

```
void flmkme (
            char ** flmid,
            I32 * rc,
            U32 * kmclen,
            U08 * kmc,
            const U32 * liblen,
            const C08 * lib,
            const U32 * fuclen,
            const C08 * fuc,
            const U32 * parlen,
            const C08 * par,
            U32 * msglen,
            C08 * msg,
            U32 * inflen,
            C08 * inf )
```

load and use FKME

Works like SETPWD but use FKME parameters. It dynamically loads the FKME module, runs the function code INFO, if `inflen` provided and execute function code COMP or DECO with the provided values.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
kmclen	length of key management context
kmc	pointer to key mangement context
liblen	length of the library name (not relevant on z/OS)
lib	pointer to the library name
fuclen	length of the function name

Parameters

fuc	pointer to the function name (for dynamic load)
parlen	length of the parameter string
par	pointer to the paramter string
msglen	size/length of return message
msg	point to the return message
inflen	size/length of return info string (could be 0 to prevent call of function code INFO)
inf	point to the info string

5.1.4.12 flmpos()

```
void flmpos (
             char ** flmid,
             I32 * rc,
             I32 * pos )
```

set read position

With flmpos() the read position can be changed. The `pos` parameter is used as follows:

On mainframes: -MAXINT -> Start of file +MAXINT -> End of file -99999999 -> Start of file for Cobol +99999999 -> End of file for Cobol -99999998 -> Start of previous file in a group file +99999998 -> Start of next file in a group file -N -> N records backward +N -> N records forward

otherwise only one relative position is supported: +99999998 -> Start of next file in a group file

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
pos	new position to use on next read

5.1.4.13 flmget()

```
void flmget (
             char ** flmid,
             I32 * rc,
             I32 * reclength,
             U08 * record,
             I32 * buflength )
```

get next record

flmget() reads the next original record sequentially. The data is transferred to the supplied `record` buffer. A special return code indicates when a new file header is encountered.

It is possible to position to a certain record in the compressed file using flmpos() and then to continue with sequential reading.

flmget() may only be used at decompression. With encrypted FLAMFILEs, a password must be provided via flmpwd() before the first invocation of flmget().

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
reclength	Returns the length in bytes of the `record` which has been read
record	Record buffer used to put record data into
buflength	Length of the `record` buffer in bytes

5.1.4.14 flmloc()

```
void flmloc (
             char ** flmid,
             I32 * rc,
             I32 * reclength,
             U08 ** record )
```

locate record

The function flmloc() is equivalent to flmget(), except that it does not put a read record into a supplied buffer. Instead, a pointer to the record is returned (LOCATE mode).

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
reclength	Length of the record pointed to by `record`
record	Contains pointer to the read record after call

5.1.4.15 flmput()

```
void flmput (
             char ** flmid,
             I32 * rc,
             I32 * reclength,
             U08 * record )
```

put record

The flmput function compresses one original record at a time.

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
reclength	Record length in bytes
record	Buffer containing the buffer

5.1.4.16 flmerr()

```
void flmerr (
              I32 * rc,
              I32 * msglen,
              const C08 ** errmsg )
```

provides an error message

With flmerr() a return code can be translated into a talking error message.

Parameters

rc	Return code
msglen	at input space for the error message, at output length of the error message
errmsg	pointer to the error message buffer

5.1.4.17 flmqry()

```
void flmqry (
              char ** flmid,
              I32 * rc,
              I32 * parameter,
              void * value )
```

query extended FLAMFILE attributes

With flmqry() parameters can be obtained during decompression. It may be called at any time after flmopn(), but the results depend on the moment it is called. E.g. SPLIT.. are first known after flmopd(), CRYPTOMODE after flmopf().

Note: In opposite to the other function calls the return code `rc` was expanded into two words (2 x 4 byte). The first word is still the return code, the second word is the info code. The info code is the parameter in error on return.

See also

flmset() and flmset_parameter

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
parameter	identifies the extended parameter to query
value	will contain the value of the specified `parameter` after return

5.1.4.18 flmset()

```
void flmset (
              char ** flmid,
              I32 * rc,
              I32 * parameter,
              void * value )
```

set extended FLAMFILE attributes

flmset() sets parameter that open functions do not support. It must be called before flmopd() and/or flmopf(). A later call will fail with return code 90.

Note: In opposite to the other function calls the return code `rc` was expanded into two words (2 x 4 byte). The first word is still the return code, the second word is the info code. The info code is the parameter in error on return.

Besides FLMSET_CRYPTOMODE Extended attributes are only meaningful on the mainframe. FLMSET_CRYPTOMODE is used to set the encryption mode when writing a FLAMFILE

See also

flmqry() and flmset_parameter

Parameters

flmid	Identifies the FLAMFILE for which the function is executed. It is set by the flmopn() function and must not be changed until calling flmcls().
rc	Return code
parameter	identifies the extended parameter to set
value	will be used to set the specified `parameter`

Index