limes datentechnik

FLAM ®

FRANKENSTEIN-LIMES-ACCESS-METHOD

(MVS®)

USER MANUAL

— Edition October 2009 Version 4.3 —

User Manual FLAM® V4.3 (MVS)

Preface

This Manual describes data compression and decompression with the Frankenstein-Limes-Access-Method. This method is implemented by the product FLAM.

FLAM compresses structurally related data using the algorithm upon which the Frankenstein-Limes-Access-Method is based. This method has been patented by the German, U.S. and European Patent Offices, registered by the inventor at the 19.7.1985.

This manual consists of the following chapters:

Introduction	This chapter explains FLAM's basic concepts and terminology and suggests areas for its use.
Functions	This chapter gives a general overview of the available functions.
Interfaces	This chapter contains the formal descriptions of all parameters and program interfaces.
Method of Operation	This chapter explains internal operative details to allow for optimal utilization of FLAM.
Application examples	This chapter illustrates by practical examples how to achieve satisfactory results quickly. It also contains hints and tips for FLAM users.
Installation	This chapter is the guide for installing and customizing FLAM.
Technical Data	This chapter describes the system environment required for FLAM operation. It also summerizes product features and characteristics.
Messages	This chapter contains a complete list of FLAM messages, their meanings, and required operator actions.
The FLAM user interface	TSO/ISPF users are supported with a user interface consisting of panels and CLIST procedures. This chapter describes control flow and principles of operation.
Appendix	The appendix includes the standard code conversion tables.

What knowledge is required?

You should be quite familiar with the MVS operating system, particularly with its command language.

This information is described in the following publications:

- JCL
- DATA Administration Guide
- VSAM Access Method Services

How to navigate through this manual:

All updates in their edition are summarized in the Documentation updates.

limes datentechnik® gmbh

FLAM (MVS)

User Manual

Documentation updates

Documentation updates 8 - FLAM V4.3

Changes in FLAM manual V4.2 from November 2007 by this supplement from October 2009 (FLAM V4.3).

FLAM V4.3 is a true superset of all previous versions, so any usage that has been working with earlier FLAM versions remains compatible with FLAM V4.3.

MODE=ADC as default

Compression mode ADC is introduced as default value. That means, that when the MODE parameter is omitted MODE=ADC ist used.

FLAM has been able to decompress this mode on all other platforms for about 10 years, so there should be no compatibility problem at all.

Remember: It is not allowed to mix MODE=VR8 and MODE=ADC (as with CX7 and CX8/VR8).
Take care when using DISP=MOD in the DD-statement of existing JCL and without the FLAM MODE-parameter. Then enter MODE=VR8 (the old default value) as FLAM parameter or start with a new file.

HW-AES

FLAM uses CPACF hardware for AES encryption, when present.

This is default for z10 systems and newer. FLAM automatically checks the availability, so no parameter other than CRYPTOMODE=AES is needed.

Encryption by hardware increases perfomance and decreases cputime. Up to 30 % may be saved. It depends on the compression ratio, less compression increases time savings. Particularly using MODE=NDC, packing files without compression.

Wildcard Syntax

Wildcard syntax may be used in file lists for compression, not only in the FLAMIN parameter.

This improves better assistance for compressing in Group-FLAMFILEs!

New Utilities

New utility programs for easier use of FLAM and FLAMFILEs are released.

FLAMCKV

Analyses a VSAM-KSDS FLAMFILE. The protocol allows checking for correct attributes of the cataloged file. Particularly performance oriented parameter of the VSAM-KSDS file are important for a fast key access.

FLAMCTAB

Creates a translation table load module from a 256 byte input stream. These bytes are read from a file of arbitrarily format or record size and stored into a load library.

With this program it is no longer neccessary to assemble and bind a module from an assembler source.

FLAMDIR

Lists a short summary of the table of contens of a FLAM-FILE like ISPF 3.4 or FLTOC of FLAM panels (or option ‘i’ in the FLAM start panel).

Documentation updates 7 - FLAM V4.2

Changes in manual FLAM V4.1 from April 2005 by this supplement from November 2007 (FLAM V4.2).

FLAM V4.2 is a true superset of all previous versions, so any usage that has been working with earlier FLAM versions remains compatible with FLAM V4.2.

This revision primarily improves the ease of use of FLAM.

New Parameters

New parameters (DATACLAS, MGMTCLAS, SPACE, STORCLAS, VOLUME, UNIT, and ODATCLAS, OMGMTCLAS, OSPACE, OSTORCLAS, OVOLUME, OUNIT) on command level or in a parameter file control data allocation of the output files (FLAMFILE, FLAMOUT). So its possible to store files on special volumes, units, and with SMS-classes without any JCL usage.

File allocation

During file allocation, FLAM tries to allocate a new file in one extend on a disk volume. If there is not enough space for this large extend, the primary space allocation will be reduced up to 1/16 of the total amount. This prevents the allocation of an out of storage situation.

Record Interface

The record interface has been improved by new parameters and functions.

FLMSET

Is extended by new parameters for file allocation (as introduced on the command line).

FLMFKY

Function ‘find key’ is now approved for compression mode ADC/NDC.

FLMGRN

Function ‘get record by number’ is now approved for ADC/NDC as well.

FLMUPD

The ‘update function’ is allowed for an update ‘in place’ for ADC/NDC compressed VSAM-KSDS FLAMFILEs, i.e. the record length must not change on rewriting the record.

Messages

Some messages are enhanced.

COMMENT

Comments in ASCII in the FLAMFILE are translated to EBCDIC for the output message with the internal A/E translation table during decompression. Entering a TRANSLATE parameter causes FLAM to use this table.

Nondisplaying characters are visualized by dots '.'.

File Allocation

Until now an error on file allocation was represented by the return code 31. Now additional system messages are displayed in the JCL-log or on the terminal. This is independent from using the utility or the record interface.

FLAM Panels

The interactive FLAM panels have been revised.

Start

An easier call to FLAM panels is introduced (but fully compatible to the older versions). The start procedure includes (or does not include) all necessary library allocation to ISPF). So it is easy to support your own TSO logon procedures.

Wildcards

Entering wildcards in the input filename (*, %) on compression lead to a selection of resulting filenames which may be edited.

Parameter

All parameters are now stored in a parameter file for execution. So, no double quotes are needed any more for C'- or X'-parameter (e.g. CRYPTOKEY=X'01AE94' instead of X''01AE94'').

LOAD library

Without any input for the FLAM LOAD library in the option panel the system concatenation (LINKLIST) will be used.

FLTOC

Duplicate filenames, 'exotic' (non z/OS compliant) or 'overlength' names are supported in Group-FLAMFILES for selection.

Documentations updates 6 - FLAM V4.1

Changes in manual FLAM V4.0 from April 2003 by this supplement from April 2005 (FLAM V4.1).

FLAM V4.1 is a true superset of all previous versions, so any usage that has been working with earlier FLAM versions remains compatible with FLAM V4.1.

Encryption

The AES (Advanced Encryption Standard) algorithm (introduced in FLAM V4.0) has become faster and saves up to 50 % CPU-time.

KMEXIT

A key management exit may be called by the parameter KMEXIT. This user written program supports FLAM with a key for en-/decryption. It is used as an interface to an existing key management system. Additional data may be stored as comment into the FLAMFILE during encryption, sent to the exit for decryption.

KMPARM

The KMEXIT routine receives control information from the caller by this parameter KMPARM.

COMMENT

Parameter COMMENT causes FLAM/FLAMUP to store these data into the FLAMFILE as a comment (user header) during compression. These data are protocolled during decompression.

File names

To avoid any conflicts with national character sets or naming conventions in other systems, all file names stored in ASCII character set are translated for message and selection in the following way:
all national characters are translated to ‘X’, a backslash ‘\’ to slash ‘/’, and blanks ‘ ‘ to underline ‘_’.
So it is easier to enter foreign file names that are unsupported in the z/OS environment. The file name itself remains unchanged in the FLAMFILE.

Entering ‘*DUMMY’ as a file name causes FLAM to use this file as dummy like the JCL command //ddname DD DUMMY. I.e. reading an input file leads to EOF (end of file), writing to an output file has no effect.
So DD-statements are not longer necessary for DUMMY files.

Enhancements of the Record Level Interface

New functions are added to the record level interface.

FLMEME

Ends a member in a Group-FLAMFILE..
With SECUREINFO=YES additional data (e.g. byte- and record counter) are written to the FLAMFILE member (member trailer) during compression. On AES encryption, the MAC of the ending member is stored into the member trailer.
A new member, started with FLMPHD, or FLMCLS, must follow.

FLMSET

New interface to set parameter without changing old interface calls (e.g. encryption mode, split mode, -size).

FLMQRY

New interface to receive parameter values without changing old interface calls (e.g. encryption mode, split mode, -size).

Messages

New messages (FLM0435, FLM0445, FLM0485, FLM0487) are provided for integrity of the FLAMFILE, for KMEXIT and COMMENT.

Documentations updates 5 - FLAM V4.0A

Changes in manual FLAM V3.0 from April 1999 by this supplement from April 2003 (FLAM V4.0).

First of all, FLAM V4.0 contains the complete predecessor version as a subset, so that it is possible to compress and decompress in the familiar way with MODE=CX7, CX8, VR8, and ADC.

OS/390 and z/OS

FLAM (MVS) V4.0 is usable in MVS as well as OS/390 or z/Os.

AES-Encryption

The National Institute of Standards (NIST) has defined the **Advanced Encryption Standard** (**AES**) for encrypting data. The method was described in the Federal Information Processing Standard (FIPS-197) in November 2001 and approved effective May 26, 2002.

FLAM uses this algorithm for encrypting compressed data. Keys of up to 64 characters can be specified (see also the description of the PASSWORD parameter in version 3). Internally, a key of 128 bits is derived (AES-128) and data security is enhanced by the insertion of verification fields created also with AES (hash-MACs).

This encryption method is activated by setting the parameters CRYPTOMODE=AES and CRYPTOKEY=key and is available with compression modes ADC and NDC (MODE=ADC or MODE=NDC). With CRYPTOMODE=-AES the compression mode defaults to ADC rather than the mode specified in the default settings.

FLAMFILE security

By specifying SECUREINFO=YES additional information is saved with the compressed file that allows verifying the integrity of the FLAMFILE without decompressing it. Changes to such a FLAMFILE (e.g. by updating, adding, or deleting members from an group FLAMFILE) are detected already by a formal check. This information is always added when encryption is used. It is ignored by FLAM 3.0 and does not cause decompression errors there.

SECUREINFO=IGNORE suppresses this security check. This may be desirable with secure FLAMFILEs that have been concatenated.

SECUREINFO=MEMBER limits application of these integrity checks just to specified compressed members in an group FLAMFILE rather than to the entire FLAMFILE.

FLAMFILE splitting

During compression a FLAMFILE can be splitted serially or in parallel into several parts, subject to the settings of

the parameters SPLITMODE, SPLITNUMBER, and SPLITSIZE.

Only the filename (or DD-name) of the first fragment of a split FLAMFILE must be specified at decompression and no additional settings are required. FLAM detects automatically whether and, if so, how a FLAMFILE has been splitted and searches by itself for the remaining fragments.

Splitting of FLAMFILEs is only available for binary compression modes (MODE=CX8,VR8,ADC,NDC). Binary informations have been added to every part of a splitted FLAMFILE.

Serial Splitting

Serial splitting (SPLITMODE=SERIAL) means that when the file currently used to store compressed data reaches a specified size limit it is closed and subsequent processing stores the compressed data into a newly created file (fragment). The number of fragments of a splitted FLAMFILE created is not limited. It only depends on the amount of data generated by the compression process. At decompression, FLAM verifies the order, the presence and the affiliation of all fragments.

This feature provides an efficient support for file size limitations (e.g. with e-mail attachments or file transfers). It can also improve system performance by allowing transmission of fragments over a network to begin before termination of the entire compression process

Parallel Splitting

With parallel splitting (SPLITMODE=PARALLEL) compressed data is stored into a specified number of fragments (SPLITNUMBER=number). The current version can handle up to 4 parallel fragments. The size of the fragments depends on the amount of data generated during compression. At decompression, FLAM verifies the order, the presence and the affiliation of all fragments. Decompression requires the accessibility of all fragments of a FLAMFILE.

None of the data can be recovered when one parallel fragment is missing.

One benefit of parallel splitting can consist in improved utilization of transmission capacities. Also, locally distributing FLAMFILE fragments may avoid unauthorized decompression without using encryption.

FLAMFILE examination

The parameter CHECKFAST effects a formal examination of the FLAMFILE. This examination includes verifying all checksums and assuring completeness and integrity of the FLAMFILE. These tests are performed without decompressing it. Specifying an additional parameter, CRYPTOKEY, causes FLAM to also decrypt the FLAMFILE and check all MACs.

The same tests are performed when the parameter CHECKALL is specified. In addition, the FLAMFILE is decompressed without storing the decompressed data. With encrypted FLAMFILEs, the encryption key must be provided.

MODE=NDC

Data compression can be suppressed using MODE=NDC. Data are only formatted and, if requested, encrypted. This saves CPU time with data that do not compress efficiently (e.g. FLAMFILEs or compressed image files). The same security features are available as for compressed data.

MODE=NDC is downwards compatible with FLAM V3.0.

FLAM Panels

The interactive FLAM panels have been extended to support the newly added encryption options. In particular, the FLTOC-Overview allows specifying decompression parameters (see ch. 9.8.1).

Documentations updates 4 - FLAM V3.0A

Changes in manual FLAM V2.7E from April 1995 by this supplement from April 1999.

First of all, FLAM V3.0 contains the complete predecessor version as a subset, so that on the one hand it is possible to compress and decompress in the familiar way with MODE=CX7, CX8 and VR8; on the other hand creating the required compressed files is no problem if your partner has not yet changed over to FLAM V3.0, for example.

New Compression

A new high efficient compression method is implemented. With *MODE=ADC* (**A**dvanced **D**ata **C**ompression) the data are compressed *"straight forward".* The relative optimization of different search and presentation techniques is progressive (adaptive model). The code assignment changes continuously.

Autonomous data segments up to 64 KB in size are compressed. The maximum permissible number of records has been increased to 4095 (previously 255). MAXBUFFER is 64 KB static.

This method is independant from any record- or data structure and has a higher compression rate than the predeccesing versions.

New Compressing Syntax

With MODE=ADC any compression (FLAMFILE) differs from each other, even when the input data are identically. In other words, with MODE=ADC a unique FLAMFILE will be created.

A new checksum technique is introduced in this compression method. This is for security reasons (data manipulation) and to identify problems of file transfer products (loss or change of data during transmission).

Another modification is the so-called hardware ID. FLAM forms a 32-bit code from the hardware information about the environment. This code is incorporated in the compressed data to create the unique FLAMFILE. It is like a hard- and software stamp to identify the compressor but without knowing the name of the compressor itself (it is unique but anonymous).

Password

Data protection and data security, especially protection against unauthorized attackers, always has top priority. With MODE=ADC all compressed files can be enciphered with a password. Without the knowledge of the password it is impossible to decompress the FLAMFILE.

The PASSWORD itself is allowed to have up to 64 bytes (512 bits). It can be specified either in printable format with C'...' or as a hexadecimal string with X'...'.

Enhancements of the Record Level Interface

A new function is added to the record level interface.

FLMPWD

passes a password for compression or decompression using MODE=ADC.

Enhancements of the user interface

A new CLIST procedure is added to use in ISPF function 3.4:

FLTOC

Shows the directory of a group FLAMFILE like ISPF 3.4. It is possible to browse a member of this FLAMFILE (decompressing 'on the fly') or to decompress a member to store on to disk.

Documentation updates 3 - FLAM V2.7

Changes in manual from October 1992 (FLAM V2.6) by this supplement from August 1993 (FLAM V2.7).

FLAM V2.7 is a functional improvement of version 2.6. It is upward compatible to all previous versions of FLAM. Compressed data from versions 2.6 and 2.7 are identical and freely exchangeable as long as no new functions or file formats are used.

In addition to new functions being added to the record level interface, enhancements have also been made to the FLAM utility.

Support of other file formats:

VSAM Linear Data Set

LDS files can be assigned as input or output files.

For performance reasons, FLAM reads or writes 4 KB LDS blocks at a time in units of 64 KB by default. It is also possible to specify a logical record and block length in which the LDS file is to be read or written (e.g. FLAMIN=LDS.FILE, IRECSIZE=100, IBLKSIZE=65536, IDSORG=LDS), i.e. the LDS file has fixed records of 100 bytes length with blocking of 64 KB.

Since it is possible to decompress into LDS files from any file format and still set up a structure, FLAM is particularly well suited to loading for test purposes.

PO libraries

PO libraries can be compressed and decompressed either collectively or selectively (FLAMIN= USER.PO(MEM*)). The directory entries are left unchanged (this also applies to load libraries!), ALIAS members are supported. In particular, the FLAMFILE can itself be a PO file.

When compressed library data is transferred to a computer running under a different operating system (e.g. BS2000), it is still possible to create a library (e.g. LMS) again from this data.

Decompression can be performed completely or it is also possible to specifically select individual members for decompression.

Automatic creation of files when JCL is not specified

By entering parameters (FLAMIN=filename, FLAMFILE=filename, FLAMOUT=filename), the specified files are automatically allocated by FLAM if a JCL has not been specified. If FLAMOUT=<*> is specified, all values for decompression (such as the file name, file type, record format, record length, block length, file size (for data compressed under MVS)) are taken from the file header of the FLAMFILE. This means that the original file is completely reorganized and recreated with one extent on the disk.

A specified JCL takes precedence over parameters settings. SMS is required in order for files to be created (see chapter 3.1.2.1).

Compressing a number of files into one FLAMFILE in one run (creating a group file)

Until now, group files could be created only by appending (DISP=MOD in the JCL) the FLAMFILEs in several steps.

WILDCARD syntax in input file

By entering a partially qualified file name (e.g. USER.*.LIST, USER.A*.OBJ(FL*), ...) or specifying a list of files, all the files are stored in compressed format in one compressed file (group file) (N:1 relation).

Group file

All the files are assigned dynamically by FLAM, and the file type (PS, PO, VSAM-ESDS, VSAM-KSDS, VSAM-RRDS, VSAM-LDS), record format (F, V, B, S, M, A), and record and block lengths are detected automatically.

The files in this group file can be decompressed either individually (selected by name) or altogether.

Examples:

All files whose first qualifier is USER and third qualifier is LIST are to be compressed into the FLAMFILE called USER.CMP (see also chapter 3.1.4):

```
//..EXEC PGM=FLAM,PARM='C,FLAMIN=USER.*.LIST,
FLAMFILE=USER.CMP'
```

Here a DD name is assigned as the input. The file contains the names of files which are to be compressed in this call:

```
//..EXEC PGM=FLAM,PARM='C,IDDN=>DIRIN'
//DIRIN DD *
USER1.FILE.VSAMESDS
USER1.DATA.PSDATEI
USER2.DATA.POLIB
USER3.DATA.POLIB(MEMBER)
//FLAMFILE DD  DSN=...
```

Compressing a number of files into a number of compressed files

Conversion rules for file names (FLAMFILEs)

By entering a partially qualified file name (e.g. USER.*.LIST, USER.A*.OBJ(FL*), ...) or specifying a list of files, all the files are stored in compressed format in a number of compressed files (N:N relation).

The name of the FLAMFILE is then generated in accordance with a conversion rule to be specified (e.g. FLAMFILE=<*.LIST=*.CMP>, i.e. all files with the extension LIST are given the extension CMP). In this way, it is also possible to set all the compressed data of a run as members of a PO library (see chapter 3.1.4).

Example: In the FLAMFILE PO library, all members are given the name of the compressed list:

```
//... EXEC  PGM=FLAM,
PARM='C,FLAMIN=USER.*.LIST,
FLAMFILE=<USER.*.LIST=USER.PO(*)>'
```

Decompressing group files

As in the previous versions of FLAM, it is possible to decompress the entire group file into an output file specified in the JCL.

Conversion rules for file names (FLAMOUT)

By entering a conversion rule for the file name for decompression, it is now possible for FLAM to create all the files automatically.

It makes no difference if the compressed data has been created under a different operating system (VSE, DPPX, UNIX, OS/2, ...). All files are created using a file format appropriate to the MVS system.

The only prerequisite is that the file header exists in the FLAMFILE (parameter HEADER=YES (default setting)).

Unless specified differently the default procedure creates a PS file with LRECL=32756, BLKSIZE=32760 and RECFM=VB.

If the compressed file has been compressed on an MVS system, it can also be created in one extent on the disk.

Examples

A group file has been created on an MVS system; the same user entry exists on the computer currently being used (see chapter 3.1.4):

```
//.. EXEC PGM=FLAM,PARM='D,FLAMO=<*>,FLAMFILE=...'
```

This means that all the files are decompressed into their original names. If files have already been catalogued, these catalogue entries are used (regardless of the entry in the file header).

The same group file, but name changes are required:

```
//... PARM='D,FLAMO=< DATA.*=USER2.DECO.*>'
```

Selection rules

Herewith, all files with the prefix 'USER1.DATA.' are decompressed and given the new prefix 'USER2.DECO.'. If the group file contains other files with a different prefix, these files are not decompressed (individual file selection).

A group file has been created on a different computer. No more information about the file is available:

```
//... EXEC PGM=FLAM,PARM='D,SHOW=DIR'
```

This command displays the contents of the file header of the FLAMFILE. If, for example, the compressed data has been generated under UNIX and all file names begin with '/homeA/ag50/dasp.dat/' (followed by the "actual" name), these names can be converted:

```
//...PARM='D,FLAMO=</homeA/ag50/dasp.dat/*=USER.*>'
```

Internal file names

If a group file has been created with HEADER=YES but FILEINFO=NO (i.e. without saving the file name), each file can be accessed via the internal name from FILE0001 (for the first file) to FILE9999 (for the 9999th file) in a conversion rule.

The saved file name can on decompression generally be ignored by setting FILEINFO=NO. The internal names are then used for converting the file names.

Dynamically loadable record level interface

In the previous versions, the FLAM record level interface had to be rigidly linked to the calling programs.

If the user I/O interface is not used, it is now possible to load each FLAM call dynamically ('DYNAM' parameter when using the COBOL compiler). For inlinked purposes, the record level interface FLAMREC can be linked without any modification to the program.

The record level interface has been enhanced by new calls

FLMGRN	Read with record number
FLMGTR	Read backward
FLMFKY	Position record using key (Find Key)
FLMFRN	Position record using record number
FLMPUH	Write user data to file header (User Header)
FLMGUH	Read user data

This allows certain operations to be implemented with fewer function calls. The FIND functions can eliminate the need for buffer storage in the calling program. Furthermore, it is now possible to save self-defined data for each file in the compressed data.

User exits

When user exits (EXK10, ...) are called, their addressing mode is taken into account. The parameter lists are, however, stored in the high-order address space as long as FLAM has been linked with AMODE31. After returning from the exits, FLAM adjusts itself again to its own addressing mode, independent on how the return statement in the exit was programmed.

The user interface

CLISTs

More CLIST procedures have been created for the user interface, in particular for the ISPF panel 3.4:

FLDIR	This displays the directory information of the file
FLDISP	This displays the file (BROWSE). If it is a FLAMFILE, it is decompressed and written into a temporary file beforehand.
FLEDIT	This edits the file. If it is a FLAMFILE, it is decompressed and written into a temporary file beforehand. After editing, it is compressed again and written into the same FLAMFILE again.
FLCOMP	This compresses the file
FLDECO	This decompresses the file

Miscellaneous

Due to the many new features incorporated, FLAM can no longer be run under MVS/SP Level 1.

Empty VSAM files (i.e. files which contain no data) are handled in the same way as empty PS files for compression and are processed without an error message being output.

Chapter 5 has been supplemented by an example program in COBOL, which serves the entire record level interface of FLAM.

Chapter 8 (Messages) has been supplemented by the return codes of the subprogram and record level interfaces, as well as the condition codes from the call module FLAM.

In addition to all the examples cited in the manual, the library FLAM.SRCLIB contains the call module FLAM. This module can be modified by the user and thus adapted to suit special requirements (e.g. reentrancy, condition codes, ...). Example translation tables are also included.

Documentation updates 2 - FLAM V2.6

Changes in the previous manual from October 1991 (FLAM V2.5) by this supplement from January 1994 (FLAM V2.6).

FLAM V2.6 is a functional improvement of FLAM V2.5A. It is upward compatible to all previous versions of FLAM.

Compressed data from versions 2.5 and 2.6 are identical and therefore freely exchangeable as long as no new functions are used.

A new feature is the compression mode VR8 with FLAMCODE=ASCII. This compressed data can neither be read nor generated by FLAM (MVS) versions lower than V2.6.

In addition to this VR8 compression for ASCII files, the new features added mainly relate to an expansion of the FLAM record level interface in connection with VSAM/KSDS.

Record level interface

Compressed files in the VSAM KSDS format can be modified record by record. For this purpose, OPEN-MODE=INOUT has been implemented in the record level interface. Therefore, the FLMDEL (delete record), FLMPKY (write record using key) and FLMUPD (change current record) functions can also be used.

The function FLMFLU (enable matrix buffer) can also be used for determining an intermediate status for the statistics.

The function FLMGKY (read record using key) can be used for all compressed files created from index sequential original files. Compressed data from all previous versions can also be processed.

In particular, this also allows index sequential files archived sequentially with FLAM to be read record by record by means of keys. The compressed data can also be stored to tape or cassette.

Presentation of the compressed data

It is now possible to read and generate all compressed data in EBCDIC and ASCII code. This means also that CX7 compressed data from ASCII computers can still be processed even if it has not been recoded on the line.

The FLAMCODE parameter is now also a legal input parameter, thus allowing optimum presentation of compressed data to be selected for ASCII data on the host, too.

FLAMFILE in STREAM format

Problems frequently arise with respect to the record length when transferring binary files from MS-DOS, OS/2 and UNIX computers to host systems.

The reason is that the transmitting operating systems either do not support or do not uniformly support record lengths for binary files, or that the file transfer programs often do not allow for the record length to be specified.

As a result, a binary file is then cut up by file transfer into sections of equal length and these sections are stored as records on the host system. The original record length is lost in the process and FLAM is not able to detect the structure of the original compressed file.

This problem is remedied by the integrated decompression exit *STREAM, which is able to edit a wrapped, binary compressed file (CX8, VR8) in such a way that serial decompression is possible. This exit is automatically activated, if in the first record, an inconsistency is detected between the FLAM record length and the DVS record length when reading a sequential compressed file.

The STREAM exit can also be activated explicitly by the user with the statement EXD20=*STREAM, if the inconsistency is not detected automatically because it cannot be detected at the start of the compressed file.

If possible, compressed files in STREAM format should not be processed any further and they should not be sent by means of a file transfer, because repeated reformatting and wrapping of the files can destroy their ability to be processed. It is better to decompress such a file and then to compress it again afterwards.

Use of this exit is indicated by the following message: FLM0465 USED PARAMETER EXD20: *STREAM. The value '*STREAM' is returned in the parameter EXD20 on the record level interface.

Documentation updates 1

Changes in manual from 1989 (V2.0) by reissue from October 1991 (V2.5A)

FLAM V2.5A (MVS) is a completely new developed product. Compared with the previous version, it's functionality was enhanced in such a way that a new reference manual became necessary. However, the reference manual for version 2.0C. stays valid for all old functions and calls. In medium term, users should migrate to the new version.

Compatibility of compressed data

FLAM V2.5A (MVS) is compatible to the version 2.0 if only sequential organization is used for the compressed file.

Additionally FLAM V2.5A is upward compatible to all previous versions of FLAM.

The most important new features are:

Operating systems

FLAM V2.5A is available for:
MVS, VSE, DPPX/370, DPPX/8100, AIX/6000
BS2000, SINIX (all processors)
VMS, STRATUS, TANDEM
MS-DOS, OS/2
SCO-UNIX, SCO-XENIX, UNIX V
UNIX for HP, NCR and UNISYS systems
A FLAM nucleus on COBOL basis is available for OS/1100 from UNISYS.
Other implementations are planned for VM, AS/400, APPLE/MAC, CTOS, OS/3, VS and other UNIX systems.

Compatible interfaces

All implementations provide compatible subprogram interfaces. This allows not only to move compressed data in the FLAMFILE from system to system. It is also possible to do this with application programs containing FLAM calls. All call interfaces of previous versions are supported in an upward compatible way.

XS/ESA supported

On all /370-compatible systems (MVS, DOS/VSE, BS2000, etc.) the system independent program parts are identical. FLAM is completely reentrant and can run under all address modes (24 and 31 bit).

FLAMFILE

The restriction that the FLAMFIILE must be a sequential PS file is removed. All formats and file organizations for the original file are now supported for the compressed file, too. (PS, IS, VSAM-ESDS, KSDS, -RRDS)

Record level interface

In version 2.5A a record level interface is provided for the first time. This interface allows to process multiple files also. This interface provides the usual calls for file access, like OPEN, GET, PUT, CLOSE, etc., as provided by operating systems and higher level programming languages (such as COBOL) on mainframes.

Random access

The use of this record level interface and the new facility to store compressed data in index sequential files (VSAM KSDS) allows a fast random access to compressed data. This is well suited for setting up low frequency archives (documents and similar data) as online archives.

Integration

The record level interface enables FLAM to be integrated into applications with a low effort, provided the application source code is available. At the other hand, for a set of application packages prewritten interfaces are available. This allows to use compressed files transparently within these packages (in the same way as uncompressed files would be used). The concept of the record level interface allows the integration of FLAM into an application within a few days or weeks.

Portability

The idea of integration and the portability of FLAM in heterogeneous system environments is based on a consequent separation of the components in system specific and system neutral elements. All interfaces use the standards for subroutine linkage. This allows to exchange all system specific components (memory management, I/O, time evaluation, etc.) easily.

User I/O

Independently from the record level interface for original data, a user interface for I/O from and to files is supplied. This interface can be activated dynamically via a parameter (DEVICE=USER) for all file I/O (original file input, compressed file output, compressed file input, original file output).

Only one program

Compression and decompression are now united in one program. This was done in respect to future function, especially for update of index sequential compressed files (OPEN=INOUT/OUTIN, PUTKEY, DELETE).

Generation

All parameter default values can be defined via a generation step in a comfortable way. For this generation it is not necessary to compile program parts. All message texts, all parameter default values, and the syntax for parameter input are contained in one data module (FLAMPAR). This enables an easy adaption to other languages.

File formats

The FLAMFILE can now be created and read in all file and record formats, previously only supported for uncompressed files.

This makes the transfer of compressed files much easier.

Conversion

During creation and conversion of files the user is released from the task of observing the specific properties of a given data management system as much as possible. (E.g., the relationship between block length and record length automatically considered and adapted to the requirements of the specific DMS.)

Keys

During conversion between sequential and index sequential files it is possible to generate or delete keys if required. The key position of index sequential files is adapted automatically if a file is converted from fixed to variable record format or vice versa. The key position is stored in a system independent way and is independent from the record format, too.

Protocol

The parameter protocol has been improved and unified. At the other hand the message layout was kept as close to the old form as possible.

During decompression the old FLAM version is displayed now and the size of the matrix buffer and the compression method are documented as well. The function INFO=HOLD can now be used, too, with compression to obtain the specified parameters.

Statistics

Statistical data is evaluated based on true data (without length fields and delimiter strings). So the resulting values are independent from record format and operating system.

User interface

A user interface under TSO/ISPF has been developed to support the user. It enables the use of FLAM without having to deal with JCL statements for TSO or batch processing.

During the redesign some changes were necessary:

The message, that an original file is already a FLAMFILE, has been removed. This statement could only be made with a certain probability but not with absolute certainty.

Modification of the code conversion table via PATCH parameter is not longer supported.

The CLIMIT parameter is only evaluated with INFO=YES. For efficiency reasons no statistic is evaluated for INFO=NO.

Parameters from previous versions are always accepted and mapped on the new parameters if possible (e.g., SANZ=1 is equivalent to MAXRECORDS=1). Other parameters are simply ignored (e.g., PATCH).

The program size has been increased due to functional improvements and the combination of compression and decompression. On the other hand, FLAM can run entirely in the high-order address space.

The dynamic memory requirements for the matrix buffer have been doubled. This memory may now also be allocated in the high-order address space.

The CPU work load did not change or was reduced by 15%.

The receiving and sending of compressed data at the KOFLAM/DEFLAM interface is not longer supported. This has been replaced by the repeatedly usable, reentrant-, and XA-capable record level interface FLAMREC. For return of compressed data the user interface for file access USERIO is provided.

List of significant changes made with this version:

The modules for compression/decompression are now combined into only one program. The program call for the FLAM utility is FLAM.

The according subprogram call for the utility function is FLAMUP. The subprogram interface has been changed.

The FLAM record level interface replaces the programs KOFLAM/DEFLAM in a new but incompatible form. The record level interface is completely reentrant and can run under XA/ESA.

The FLAM interfaces and all FLAM parameters of previous versions are supported in an upward compatible manner.

Calls from FLAM V1.x (like FLKOMP, FLKOMPV, ...) are accepted and emulated in this version for the last time.

The KOFLAM/DEFLAM interface can still be used but only with the functionality of version V2.0 and without returning compressed files.

Beside the record level interface for original data an additional user interface for I/O (USERIO) is supplied. This user interface can be activated for uncompressed as well as for compressed files via parameters.

All user exit interfaces have been expanded with a work area of 1 KB in a compatible way. This improves the reentrancy of the exits considerably.

The limitation of the FLAMFILE on sequential PS files has been removed. Now the FLAMFILE can have all formats and file organisations of the original file (PS, IS, VSAM-ESDS, -KSDS, -RRDS).

Especially important is that now a FLAMFILE can be created in the VSAM KSDS format. This allows direct access (retrieval) to compressed data using the original keys!

On utility level new parameters were introduced. Parameters of previous versions are accepted and are mapped onto the new parameters. E.g., the CODE parameter is equivalent to the TRANSLATE parameter. Other parameters are simply ignored, like PATCH.

A FLAM protocol is now only issued by the programs FLAM and FLAMUP. (The new record level interface does not return a protocol, only return codes.) The protocol has been enhanced while keeping as much from the old layout as possible. It has been homogenised and contains more information. Now the used CPU time is displayed in addition to the elapsed time. After decompression the FLAM version used for compression, the size of the matrix buffer, and the compression mode (MODE) are displayed.

The dynamic memory requirement for the FLAM matrix buffer (MAXB parameter) has been increased by more than 100 percent.

FLAM (MVS)

User Manual

Contents

FLAM (MVS)

User Manual

Chapter 1:

Introduction

1. Introduction

FLAM is a software product for data compression typically used in applications in banking, wholesale and retailing, industry and public administration. FLAM is best suited for tabled data.

FLAM compresses the standardized data formats as used in banking with a typical compression rate of 4:1. For lists of material the compression rate can be as high as 95% (20:1).

Although that FLAM was not specially developed for banking applications, it is now accepted as the optional standard in data compression within electronic funds transfer. FLAM is used because of its flexibility and the proven short turnaround time.

With each new FLAM implementation, new benefits arise for each user without additional costs. It is in the interest of each user to ask for the support of FLAM by hardware manufacturers and third party software houses, and to support the installation of FLAM at sites of a business partner with whom electronic data is exchanged. That is the special benefit of FLAM regarding cost effectiveness.

FLAM uses the algorithm of the Frankenstein-Limes-Access-Method for structure oriented data compression. This method has been patented in Germany, Europe and USA beginning with 19.7.1985.

FLAM works without pre analysis of the data and without additional tables. This ensures that decompression can be always performed based on the syntax of the compressed file (FLAMFILE) and the FLAM program. This also ensures upward compatibility that allows for long time archiving.

FLAM does not need any additional specifications about the data to be compressed. The compression method is invariant to file, record, and field formats.

However, the compression effect depends on the actual data. Structural distortion mostly leads to worse compression results.

FLAM is the only product that caters for the following principles:

Transparency

On online data storage media files compressed with FLAM can be used in connection with both sequential and index sequential access methods without additional intermediate conversions. The same transparency applies for data exchange (file transfer) in a heterogeneous network between computers with different hardware and different operating systems.

Portability

Formatting of the compressed files can be controlled in a way that all requirements are met for maximum memory usage and for portability on any type of transmission lines with any file transfer product. This is especially true for punched card formats (80 columns) and for FTAM formats. Compressed records can be created in both fixed and variable format.

Convertibility

FLAM is even able to generate compressed files in a printable format. This allows to convert the compressed file from EBCDIC to ASCII and vice versa at any time. At the other hand, code conversions can be performed in combination with compression or decompression.

Compatibility

Optionally FLAM can convert file and record formats. This allows FLAM to solve compatibility problems between heterogeneous systems or version dependent file management systems. Restrictions regarding record format (fix), duplicate key, etc. are neutralized using FLAM as an access method.

System independency

A *FLAMFILE* can be used on all computer systems where FLAM is available. The FLAMFILE is the base for the access method FLAM without sacrificing the different system specific access methods of the particular file management system.

Continuity

A *FLAMFILE* can be converted during decompression into any file or record format as specified by the user. This guarantees continuity. An archived FLAMFILE can always be processed (especially decompressed) on any system.

This insures independency from a particular operating system. However, it must be made sure, that the storage medium can be read by the hardware (e.g. tape unit). Also the FLAMFILE should not be converted into a system dependent format of any given archiving product.

Data security

FLAM encrypts data and seals the compressed files using checksums for better protection. The FLAMFILE has internal synchronisation points, which allow to restart decompression after an erroneous data block caused by physical defects. Requirements for revision and controlling are fully supported.

Interfaces

FLAM provides a variety of interfaces derived from a real file management system with index sequential access. FLAM can be executed as a subprogram under control of an application. User exits within FLAM allow pre- and postprocessing of uncompressed records as well as of FLAMFILE records.

Operating systems

FLAM is currently available for a variety of operating systems:

FSC	BS2000/OSD Sinix Reliant Unix
HP	HPUX Windows OpenVMS (DEC) True64 UNIX (DEC) Non Stop OS (Tandem) OSS (Tandem)
IBM	z/OS, OS/390, MVS, MVS-Subsystem Linux (S/390, z-Series) VM, VSE OS400 AIX
Microsoft	Windows (9x, NT, 200x, Server)
NCR	Unix
SCO	SCO-Open Server SCO-UnixWare
FSC	BS2000 SINIX (all processors)
SUN	SOLARIS
PCs	Windows (9x, NT, 200x, XP, Server) Linux

Other versions can be made available on demand.

Standards

FLAM is an optional compression standard for different applications used in German banking, like BCS, EAF (LZB), DTA, and others.

Manufacturer

limes datentechnik gmbh

Philipp-Reis-Passage 2

D-61381 Friedrichsdorf/Ts.

Telephone ++49 6172 / 5919-0

Telefax ++49 6172 / 5919-39

eMail: info@flam.de
eMail: info@limesdatentechnik.de

Internet: http://www.flam.de
http://www.limes-datentechnik.de

Marketing

Bank Verlag GmbH (BCS modules)
limes datentechnik gmbh (other systems)

Furthermore we refer to the ISIS reports (NOMINA).

Cooperations

The following products support FLAM via interfaces:

BCS	Bank Verlag GmbH
CFS	OPG Online Programmierung GmbH
MultiCom	CoCoNet AG
NATURAL	Software AG
SFIRM	BIVG Hannover GmbH & Co.KG
SIRON	Ton Beller AG

Licence fees apply to some of these interfaces.

For electronic banking (BCS) some banks and their partners provide complete solutions for PC users.

For the manufacturers of FLAM each new cooperation with software houses based on the FLAM standard is welcome. This allows maximum benefit for all partners.

The advantages of FLAM in key words:

Data transmission

- Cost reduction by volume reduction (e.g., packet switching)
- Faster transmission by virtualisation
- Implicit acceleration of other transmissions
- Change to slower physical lines possible
- Lower access and operation costs
- Less transmission faults due to physically slower transmission
- Solution for technological bottlenecks
- Increase of potential transmission frequency
- Reduction of network node and buffer workload
- More efficient reaction possible to line breakdowns, transmission faults, or operating mistakes
- FLAMFILE in parking position saves space, allows immediate restart of transmission and can be archived
- Compatibility of the FLAMFILE in heterogeneous networks
- Portability of the FLAMFILE due to format options
- Convertibility of the FLAMFILE for printable data by pre- and post character conversion possible
- Conversion of record and file formats possible (Utility)
- Transparency of FLAMFILE to other applications
- More remote controlling possible due to volume reduction
- More data exchange via line possible due to volume reduction
- More swapping to emergency computer centres possible due to volume reduction

- Automated remote archiving possible
- Automated remote restore possible
- Better data revision due to automated procedures
- More data integrity due to check sum technique
- More data security due to FLAM typical data encryption
- Higher efficiency of additional cryptographic methods

Data storage

- Reduction of data storage on all media
- Fewer requirements for physical space
- Less multi volume files (Disc, Tape, Floppy)
- Fewer requirements for power, air conditioning, protection
- Less fixed capital required
- Less overhead in archive and more continuity
- Less I/O, less work load for channels
- Probably fewer controllers, I/O ports, buffers
- Acceleration of batch copy processes and of backup-/restart processes, thus more resources for automation of computer centre
- Shorter processing times and shifts
- Additional data protection due to FLAM access
- Integrated protection against manipulation due to FLAM syntax
- Process typical data encryption
- Effective protection of logical deleted data
- Innovative (combined) access methods for index sequential and logically blocked data in heterogeneous environments.

1.1 Introduction to FLAM® with MODE=ADC

Since version 3 FLAM offers 3 fundamental enhancements:

- A universal MODE=ADC (Advanced Data Compression)
- A new, sophisticated FLAM syntax (Frankenstein-Limes-Access Method)
- An extremely efficient PASSWORD enciphering system.

First of all, FLAM contains the complete predecessor version as a subset, so that on the one hand it is possible to compress and decompress in the familiar way with MODE=CX7, CX8 and VR8; on the other hand creating the required compressed files is no problem if your partner has not yet changed over to FLAM V3.0, for example. This applies not only to interfaces and user exits, but also to the elegant MVS subsystem. (User exits are needed for inserting/deleting and for editing records/fields before/after the compression/decompression procedure.)

The above-mentioned compression modes have yielded extremely good results for the kind of data that typically arises with commercial applications on mainframes. It is up to each user to decide whether or not to continue using this technique, which often already permits compression scores of 85% or more.

The increasing penetration of PC and UNIX systems into commercial data processing has led to significant changes in data structures. The FLAM compression technique, which is based on structural redundancies, has had to be extended to take account of context-related views.

FLAM was originally - and still is - an access method designed to facilitate efficient working with compressed data. By definition, this philosophy forbids FLAM from creating or using any kind of temporary files. Preliminary analyses for determining the most suitable compression technique and/or multiple-step procedures are quite simply irreconcilable with demands for a high-performance, direct-access method (for autonomous segments), with a concept that is essentially invariable across almost all platforms (from PCs to mainframes).

The user should be allowed to compress as early as it appears useful to do so and decompress as late as necessary - in isolated cases (retrieval) where possible only locally and if appropriate selectively. The FLAMFILE® should be capable of being used consistently across all platforms for storage, archiving and file transfers, including backups (transferring to external storage), as a standard tool for every situation.

MODE=ADC (Advanced Data Compression) compresses in a very straightforward way. The relative optimization of different search and presentation techniques is progressive (adaptive model). The code assignment changes continuously.

Autonomous data segments up to 64 KB in size are compressed. The only way in which the user can influence this size is via the number of records (MAXRECORDS). The maximum permissible number of records has been increased to 4095 (previously 255). MAXBUFFER is 64 KB static (ADC).

The term "record" refers to a logical unit that is defined in the user's data management system. Record formats can be either fixed or variable. Some systems use a length field for records, while others have a delimiter. This is important if, from the point of view of an application or when data is exchanged, records are crucial as a logically invariable access basis (similar to the record interface in FLAM during compression/decompression).

On systems that have no file catalog containing information about what is to be interpreted as a record, it is perfectly possible to simply read in 64 KB; this will not have any adverse effects on compression with MODE=ADC.

If a file with delimiters is read on a PC or UNIX and these delimiters are not interpreted as such, problems may be encountered if the file needs to be exchanged in a heterogeneous environment after decompression and adapted to this new environment.

FLAM allows problems such as these to be precluded right from the start by setting certain parameters, providing the record format is known and utilized. It permits a neutral, future-proof presentation, which can be matched automatically to the new situation when the file is decompressed (format conversion).

Certain files or file groups (e.g. libraries) are inevitably reorganized whenever a file is compressed and decompressed again with FLAM.

The compressed file - the FLAMFILE - can only be formatted individually with FLAM, because this "temporary" file may have to satisfy completely different requirements from the original file, for instance in connection with file transfers (portability).

Example: IBM's RJE is only capable of transferring files with a fixed record format. FLAM compresses the file in question and turns it into a FLAMFILE in RJE format. When it is decompressed, another format conversion takes place without the user even noticing. FLAM can also bundle so-called load modules stored in an MVS library together in a FLAM group file (i.e. concatenate the compressed files) and export them to a PC. If this data is later transferred back to an MVS system, decompressed there with FLAM and re-saved in a library, it can be called up and loaded from the MVS system in the usual way.

If printable data is coded in such a way that it can be converted unambiguously (1:1) from EBCDIC to ASCII or vice versa, this conversion can be activated when the file is compressed or decompressed. The tables that are supplied together with the software are only suggestions, as there are far too many possible variants for them all to be reproduced. You can adapt these tables easily to suit your own particular needs. We recommend converting the code on the same system that you intend to use for decompression, because going on past experience the relevant table settings are likely to be most reliable on that system. 1:1 convertibility and compatibility are then guaranteed.

If you need to exchange data in a printable format using a file transfer method that converts the code "en route", you must use the predecessor version with MODE=CX7. Experience has shown that code conversions with file transfer products are far too unpredictable. We can only advise you not to attempt them in the first place. The safest procedure is to exchange binary data and to convert the code either before or (preferably) afterwards. With edited listings there is also the problem of the control function of the first byte in each record (print control characters).

If the transfer has to take place in ASCII, there are automatic mechanisms in many file transfer products for recoding binary data temporarily into what appears to be printable data and then restoring it to its original state after the transfer. You could write your own 3:4 routine for this purpose and activate it in a FLAM user exit (portability).

Format errors, which are reported by FLAM as checksum errors, are a relatively common occurrence in connection with file transfers involving FLAMbéd data. All the parties concerned can thus be certain that the transfer was error-free from the user's point of view (beyond the limits of the FT protocol). Some PC products do not even have a checksum for the compressed file, but instead just a single checksum for the complete original file, which may be up to 4 GB in size. (FLAM does not impose any restrictions on the file's type or size.)

What an irony: without FLAM, this kind of error would often not even be noticed. It is thus easy to gain the false impression that the error would never had occurred if FLAM had not been around to interfere. The combination of FTP and FLAM exhibits particularly remarkable synergy effects, which are indispensable on account of the inadequate security and stability of the FTP. FLAM is also very important as a pre-post-process for file transfers with checkpoint restart.

There is actually a whole series of problems related to file transfers that can only be solved using FLAM. In the very few instances where this is not the case, the reason lies in the problem itself and not in FLAM. Major problems are encountered, for example, when character sets are recoded if umlauts continue to be employed, even though the stock of special characters has been largely used up.

It is not possible to compress without creating a working memory for temporary information. FLAM requires around 160 KB for MODE=ADC over and above the I/O areas. From the algorithmic point of view, this basic memory must always be available, if the necessary CPU time is to be kept within reasonable limits. Compared with other models, this memory size is relatively small for an adaptive model.

If we were to compare the compression effects fairly with those of other products (usually PC products), we would have to split the original file into segments (small files) of 64 KB each beforehand and then add the individual results together. Also, a FLAMFILE has a certain amount of "packaging", which inflates the compressed file by up to 2%, both for security reasons and on account of the innovative access techniques.

One advantage of retaining the segmentation principle is that in the event of a serious data error only a single segment will be affected. Every segment in a FLAMFILE is considered independently of the others (in the same way as for a transaction) and is also saved as such (packed). This makes synchronization easy; you can start at any segment half-way through.

If no compression effect whatsoever is evident after the compression procedure has completed approximately 16 KB of a particular segment, the compression is aborted for this segment with MODE=ADC and the original input of up to 64 KB (segment) is used instead 1:1. If the compression effect in a segment does not begin until after 16 KB, it will no longer be detected, because the logic function that compares the "cost" and the "benefit" will come to the conclusion that this segment is probably not compressible.

Reason: the poorer the compression effect, the greater (unfortunately) the amount of CPU time that is needed (ultimately out of all proportion). This is an inherent drawback of the general principle.

FLAM owes its capability for serving multiprocessor systems to its use of a layer model: one process reads, forms the segments and distributes them to other processes for the purpose of compression; another process collects the compressed segments, formats them as a FLAMFILE and writes them.

Although this technique is not actually critical at the present moment, FLAM's model will be ready when the time comes.

FLAM does not simply refuse to work if the input is itself a FLAMFILE. This can sometimes be an extremely useful characteristic. Let us assume, for instance, that you have a library containing a large number of small elements which first of all need to be compressed autonomously and saved as a group file, so that the library can be reconstructed correctly on the basis of the element names and their attributes. In this case, you cannot expect an excessive amount of compression.

If you use FLAM V2.x with MODE=CX8 and MAXRECORDS=1 for this purpose, all you will achieve with this leader will be to create the above-mentioned group file, in which greater importance is attached to the diverse information than to the compression effect. This "flat" file can be compressed by FLAM V3.0 with MODE=ADC. You could also use a utility that is capable of performing a similar function (group file) instead of the leader with FLAM V2.x.

In exceptional cases, you may even have very highly structured files which you can compress relatively effectively beforehand with FLAM V2.x, MODE=CX8 and MAXRECORDS=255; you can then compress the resulting compressed file further with FLAM V3.0 and MODE=ADC. Generally speaking, however, FLAM V3.0 with MODE=ADC and MAXRECORDS=4095 always performs better than the predecessor version or than a two-step variant with it. There is no compulsion to change to a different mode if you are satisfied with the old compression technique and syntax of FLAM V2.x. All new features (e.g. PASSWORD enciphering) require at least FLAM V3.0 with MODE=ADC though, especially since the FLAMFILE syntax has been significantly improved.

The new syntax ensures firstly that data which cannot be compressed despite the ADC technique is not expanded by more than 2%, and secondly that the originals - which in such cases are merely copied - are rendered unrecognizable.

The reason for this is the checksum method, which is the only one of its kind in the world. At the time the four checksums (!) are formed, the third of these encrypts the compressed input in such a way that the procedure can be reversed by applying the checksum function twice. If the compressed data in a segment has been mutilated (data errors, manipulation), the defect will spread to the remainder of the compressed segment "like the plague". The defective data is then useless. The decompression procedure is not even started! This CRC routine can moreover only be activated in FLAM if the complete compressed segment is available for "decryption".

Certain PC products allow the original to be "read" if it has not been compressed. CRC errors are not reported until the decompressed file is closed, because the checksum is based on the original data. The decompression is not aborted despite the checksum error. The decompressed file may contain many different kinds of error, including size errors, even though the number of bytes that appears in the header of the compressed file is correct.

Since FLAM V3.0 with MODE=ADC the segment checksums are linked together by means of a connector. Providing compression and decompression are always serial, the integrity of this sequence can be verified.

The connector is additionally given a time-dependent color code, so that if the same segment is compressed again later on it has a different appearance. The compression effect is identical.

Another modification is the so-called hardware ID. FLAM forms a 32-bit code from the hardware information about the environment. This code is incorporated in the connector. If exactly the same file then happens to be compressed twice at times that do not result in different connector settings, even though different hardware environments are used, the connector - and thus also the external appearance of the compressed file - is modified automatically.

The aim of these techniques is that, insofar as possible, every data segment compressed with FLAM should be unique in terms of its contents (original), its environment and its time of compression. The sum of the checksums for the various layers is a signature that can be used by an addressee as unequivocal confirmation of reception (complete and with no loss of integrity).

The FLAMFILE itself is written record-by-record for format-related reasons, in the same way as in the predecessor version (e.g. fixed 512 bytes). Each record in the FLAMFILE has a simple checksum, which allows you to verify that no formatting errors have occurred during the transfer. This is still a relatively common type of user error (regardless of whether or not FLAM is involved). The compressed segment is not "assembled" until after the format check.

Every compressed segment has a header. This allows its exact position to be located in a FLAMFILE (synchronization). The header must not be encrypted (and indeed it is not encrypted!) for this reason. It is however protected by means of a separate checksum, so that you can be sure that the information it contains is always correct.

You can find our product name FLAM in ASCII code at the end of every compressed segment. This is useful for synchronization in the event of a defect or if you reading backwards.

A special, hidden checksum refers directly to the enciphered PASSWORD. If this checksum is not correct and the PASSWORD enciphering FLAG is set, an attempt has been made to decode with an invalid PASSWORD. If the PASSWORD FLAG is not set, but somebody else is using a PASSWORD, the decoding and decompression functions will stop with mentioning this input error.

A segment decompression procedure never starts if any of the four checksums are invalid. Apart from anything else, there are technical reasons for this. Decompression presupposes a certain, constantly changing interpretation of the code. A defect will cause the decompression function to "go spinning out of control". FLAM prevents this by using a layer model with four checksums. If you attempt to subvert this principle even though an error has been reported (error message, return code), for example by manipulating the data with program patches, you are likely to provoke extremely serious consequential errors.

Data protection and data security, especially protection against unauthorized attackers, always has top priority, even without PASSWORD enciphering (see V3.1 Outlook at the end of this document).

The PASSWORD itself must not be longer than 64 bytes = 512 bits. It can be specified either in printable format with C'...' or as a hexadecimal string with X'...'. The number of what might be termed "half" bytes in a hexadecimal input must work out as a pair. If you enter a PASSWORD with C'...', you should remember that the binary conversion is dependent on the system generation. The same C PASSWORD in conjunction with a different character conversion to binary code will result in a different internal PASSWORD. You could find this useful if you are working in this environment yourself and do not alter anything. Using apostrophes as delimiters means that any trailing blanks are also considered to be part of the PASSWORD. A PASSWORD defined with C'...' must be entered again exactly before you can decode. It is advisable to test both sides in advance whenever you specify a new PASSWORD.

If you enter a PASSWORD incorrectly, you are allowed exactly one attempt at utility level, because this is all that is permitted by the internal transfer method in FLAM. You must start FLAM again and either enter or assign a new PASSWORD before you are allowed another attempt. The most you can do is to automate or optimize this restart procedure if you are using FLAM's internal interfaces.

The PASSWORD is processed internally in FLAM in such a way that it is impossible to infer any information about it. All attempts to analyze it with a view to securing a personal advantage are doomed to failure. We, the manufacturers, will be unable to help you in any way if you forget your PASSWORD. It is not even possible for an outsider to determine the length of your forgotten PASSWORD, nor whether you entered it with 'C...' or with 'X...'. It is highly unlikely that you will ever find any useful information on the Internet from hackers about how you could cut corners.

Before the first segment of a FLAMFILE can be deciphered at all, certain preparatory steps must be completed internally; they take up CPU time but are unavoidable. This means that there is a basic amount of work in every PASSWORD attempt that it is not possible to optimize. The enormous number of possible solutions that can be reconstructed mathematically is the user's surefire guarantee that nobody will manage to "crack" a PASSWORD which has been specified for enciphering a FLAMFILE even remotely easily. There is no such thing as a higher-level PASSWORD that performs the role of a master key. A PASSWORD which is hierarchically structured from the user's point of view is not recognized as such. Even the small difference between one blank more or less at the end of the PASSWORD results in completely different internal keys, which are what ultimately determine the actual procedure (2 * 4 KB key data internally).

If you then also give your PASSWORD an attribute that refers to your company or to some other feature of your environment, and thus extend the PASSWORD length artificially, the amount of effort needed by an outsider to work it out will reach astronomical proportions:

If the full 512 binary bits are used, the total possible number of variants will have 155 digits. Even if you only want 96 printable characters to be allowed per byte, you will still be left with a figure with 127 digits. The length alone, which is part of the PASSWORD, is enough to put other people off, because they have no specific information about it.

Example of a PASSWORD with attributes:

C'limes datentechnik gmbh, Ruskbakercity Friedrichsdorf/Ts.'

This makes 57 out of 64 bytes (between the two apostrophes). As an alternative to "Ruskbakercity", we could also take Huguenots, Mormons, Philipp Reis or any other attribute that is typical of Friedrichsdorf in the German Taunus region. The remainder (7 bytes in this example) is used for the actual personal PASSWORD (e.g. a blank followed by 6 bytes of variable binary code = 2.8 * 10**14 variants if the length, structure and attribute are static).

With a PASSWORD such as that described above, and without any personal modifications, you can create a "company-specific", compressed FLAM file that can only be decompressed within your company. Instead of "Ts." you could also write "Taunus" or leave off this attribute entirely and replace it with the zip code: "D-61381". Upper or lower-case notation affects the binary code, as do any structural changes. Be careful not to make input errors in hidden dialogs or with lower-case letters on mainframes.

With PASSWORD enciphering you must allow an additional 2.5% on average on top of the time for compression/decompression with FLAM V3.0 and MODE=ADC; this is an immense advantage simply on account of the restriction to compressed data. The same also applies to protection against hackers, as ownership of FLAM V3.0 is an essential prerequisite of any "attack". In addition, each compressed segment must be made available, complete and undamaged, in the right "envelope" (i.e. synchronized).

Our PC version is not available as shareware or in a similar form, and we consider it a near-on impossibility for anyone else to rewrite even the decompression part of the program and - as is common practice on the Internet - publish it "for private use" as their own, home-made product. We have endeavored to achieve a healthy measure of complexity to protect our own interests, apart from anything else. Of course, we can never protect ourselves 100% against pirate copies, nor against disloyal employees with inside know-how. Yet even then, there is no way that a person could secure any kind of advantage whatsoever for themselves by attempting to "crack" a PASSWORD. The non-optimizable portion of CPU time would still remain, even if we were to publish the sources! You determine this time yourself through the PASSWORD rules you define within your company (see PS). Please remember that there is an enormous difference between wanting protection inside your company as well and protection "just" against "unauthorized third parties", for example when files are transferred. Somebody who already works in the same building as you is likely to have access to the data you are desperate to protect through other sources too. This is a far more difficult problem to solve.

The majority of FLAM's processes are automated. We recommend saving the PASSWORD in a separate file and then getting FLAM to read it via this file. You can protect access to the file in the usual way.

The parts of the syntax that are critical for synchronization and positioning are neither enciphered nor encrypted in FLAM. This data is of no use to anybody; it can however help to speed up direct access considerably, since those parts of the compressed file whose only interest to authorized users is that they contain formatting information do not need to be deciphered or decrypted, nor do they have to be decompressed unnecessarily.

Some users may of course prefer to start by "compressing, encrypting and sealing" with FLAM and then use a stipulated enciphering technique. There is nothing to be gained, on the other hand, by enciphering the original data before compressing it with FLAM. It is however useful to form signatures and other authorization data relevant to the original before compressing it with FLAM, providing you do not otherwise alter the original data in doing so. The FLAMFILE can also be modified to make it personal and incompatible by swapping bytes via FLAM user exits. The effect is basically the same as enciphering (symmetrical exit modules).

Instead of personal keys, you can also use ready-made key systems with generation/administration functions, etc.; these keys must be symmetrical for enciphering with FLAM however (the same PASSWORD on both sides from the binary point of view).

PS: If you would like to work out exactly how many different PASSWORDs are possible, you must start with the length in bits as a power with base "2" for purely binary codes (X input); this number must be divisible by "8" with no remainder, so that the input length is an integer number of bytes. PASSWORDs in X format are always invariable for heterogeneous applications.

In the case of PASSWORDs entered with C'...', it depends how many characters are allowed. In ASCII, for instance, there are 96 printable characters (not including extended character sets). Only 52 of these characters are Roman letters, while 10 are digits, etc., etc. If the PASSWORD has a length of "k" bytes and there are up to "n" permissible characters per byte, the total number of variations will be "n**k" (power "k", base "n"). There will always be a few "dregs" left over that an attacker will exclude straight away. It is therefore important to choose a length "k" with a sufficient margin (see example of PASSWORD with attributes). A C PASSWORD often depends to a very large extent on the character sets and their binary conversion, e.g. if special characters or umlauts are used! As far as FLAM is concerned, only the binary conversion of the string that is transferred during compression/enciphering with C'...' is relevant. This may well be a different binary code on the very next screen.

1.2 FLAM and AES

The National Institute of Standards (NIST) has defined the **Advanced Encryption Standard** (**AES**) for encrypting data. The method was described in the Federal Information Processing Standard (FIPS-197) in November 2001 and approved effective May 26, 2002.

FLAM uses this algorithm for encrypting compressed data. Keys of up to 64 characters can be specified (see also the description of the PASSWORD parameter in version 3). Internally, a key of 128 bits is derived (AES-128) and data security is enhanced by the insertion of verification fields created also with AES (hash-MACs).

This encryption method is activated by setting the parameters CRYPTOMODE=AES and CRYPTOKEY=key and is available with compression modes ADC and NDC (MODE=ADC or MODE=NDC). With CRYPTOMODE=-AES the compression mode defaults to ADC rather than the mode specified in the default settings.

This fast algorithm, combined with the ADC compression, enables the user to encrypt large amounts of data with a worldwide accepted algorithm.

For further details see the manual FLAM & AES.

FLAM (MVS)

User Manual

Chapter 2:

Functions

Content

2. Functions

2.1 The FLAM Utility

FLAM is a utility that is able to compress and encrypt whole files or to expand compressed and to decrypt files.

By using the parameters COMPRESS respectively UNCOMPRESS/DECOMPRESS you select the operation mode:

Compression of an uncompressed file or expansion of a compressed file.

2.1.1 File compression using FLAM

As FLAM compresses a file it will write the result into a sequential or index sequential file, the FLAMFILE. This file may have a header that will store information about the original file.

FLAM is able to process all PS, PO and VSAM files.

To adapt the compression process to the requirements of the user, it is possible to specify parameters during the program call interactively. It is also possible to define the parameters using a parameter file or while generating the system.

FLAM creates a job protocol for each execution (on screen or into a log file).

During compression FLAM processes a set of 1-4095 (logical) records within one block (Matrix).

FLAM can process input and output from both disc and magnetic tape or cartridge. This is also true for the compressed file, the FLAMFILE.

FLAM always compresses a set of records in one step. The size of the intermediate buffer can be specified via the parameter MAXBUFFER. FLAM will read only as many input records in one step as can be stored within the intermediate buffer.

For compatibility reasons to all FLAM versions the size of the intermediate buffer is restricted by 32 kB. However, if the target computer (i.e., the computer on which the compressed file will be decompressed) is known and the FLAM version installed on this computer allows for it, a bigger buffer size for optimal compression can be defined.

It is also possible to restrict the number of records within one set using the MAXRECORDS parameter. If you define MAXRECORDS=1, FLAM will use a serial context free compression method that is only useful with long records.

The typical compression ratios are usually achieved with 16-32 records within one set. With a bigger set a slightly better compression ratio can be achieved (which also leads to less CPU consumption!), but the intermediate buffer has to be bigger.

The better the compression ratio is, the less CPU time is needed!

The compression method is always the same - based on the Frankenstein-Limes-Access-Method. Only the treatment of the matrix columns and the representation of the compressed files differs from case to case. This is controlled with the MODE parameter.

With MODE=CX8 FLAM will only compress repeating characters (horizontally and vertically). With MODE=VR8 the remaining data is compressed in addition using the FL-B(4) code. This process first translates the characters into a special 8-bit code and then homogenizes them using specially designed logical operations. The results are bit strings that can be compressed efficiently. One reason for this is that the data resulting from the Frankenstein-Limes-Access-Method can be grouped into partial equal character classes.

In both cases, the compressed file, the FLAMFILE, is a sequence of 8-bit combinations that are written into sequential or index sequential files. Record size, record format, block size can be defined by the user. Each record of a file is in addition protected with a check sum. Code conversions within the FLAMFILE are not allowed. During transmission the file has to be treated as a binary file.

For files that contain only printable characters and shall be transmitted using a 7-bit-line, FLAM offers MODE=CX7. This mode creates a compressed file which will "behave" during transmission in the same way as the uncompressed file. FLAM does not check, if the original file is suitable for transmission, but will create a compressed file using a very restricted character set that is neutral to the different code conversion utilities on the marketplace.

Using this mode one can create compressed files that can be converted from EBCDIC to ASCII and vice versa (e.g., during a file transfer).

However, it is necessary that the code conversion is reversible without any changes. Otherwise FLAM will signal a syntax error in the compressed file due to differences in the byte numbers and will stop decompression. Such cases are possible, if the original file contains printer control characters or tabulator characters, which are not converted 1:1.

Apart from this, code conversion can be performed integrated with the compression step. FLAM offers the possibility of code conversion (before compression, or after decompression). Code conversion is controlled via standard code tables or user defined code tables. For special cases where a 1:1 code conversion is not possible for all characters, user exits are provided.

2.1.2 Decompression of files using FLAM

FLAM reads a compressed file (FLAMFILE), decompresses the content and writes the result into a target file. FLAM automatically configures itself according to the parameters (e.g., buffer size or record set size) used during compression. The general layout of the compressed file is described in a separate chapter.

FLAM version 2.7 can decompress all FLAMFILES created with FLAM V1.0 - 2.6 (upward compatibility). FLAM V2.0/ V2.1 can decompress sequential FLAMFILES created with FLAM V2.5 - V2.7 (downward compatibility).

To adapt the decompression process to the requirements of the user, it is possible to specify parameters during the program call interactively. It is also possible to define the parameters using a parameter file or while generating the system.

FLAM creates a job protocol for each execution (on screen or into a log file). During decompression the characteristic parameters of the original files are restored depending on the information in the file header.

By specifying certain parameters it is possible to generate a target file differing from the original file. All conversions are possible provided that the target system caters for the appropriate access method.

That the FLAMFILE eventually was created in a different environment (e.g., different operating system) lacks influence on the operation of FLAM. Data is decompressed into equivalent file formats or in user defined file formats.

By specifying code conversion tables it is possible to change code systems after compression.

To be even more flexible, FLAM provides user exits that allow the user to apply user defined post processing.

2.2 Subprogram FLAMUP

FLAMUP differs from FLAM in that respect, that it can be called as a subprogram out of an application. All accesses to data are performed via FLAMUP.

All parameters can be passed to FLAMUP using the CALL interface or methods as provided by FLAM (interactive parameter definition or parameter file) can be used.

Using FLAMUP within a driver program allows for example to select a certain set of files and to compress/decompress all the selected files in one step. A typical example is, to select only files that were modified after a certain date (archiving).

2.3 Interface on record level: FLAMREC

The Frankenstein-Limes-Access-Method as a hardware and operating system independent and compressing access method is realized via the record level interface.

This interface allows sequential, relative or index sequential access to individual records. These records may be contained in compressed files stored on and interchanged between different devices and different operating systems.

The record level interface is provided by a set of different subprograms that may be called from all programming languages such as COBOL, FORTRAN, C, and from ASSEMBLER.

These subprograms are identical on all those operating systems where FLAM was released.

FLMCLS FLMCLS (Close) closes the current processing after all records have been received with FLMPUT, or after all records have been read for decompression.

FLMDEL FLMDEL (Delete) deletes the last read record from an index sequential FLAMFILE.

FLMFKY FLMFKY (Find Key) positions the record pointer in an index sequential FLAMFILE (that has been created from an index sequential file) in such a way that the record with the specified or the following key can be read when FLMGET is subsequently activated.

FLMFLU FLMFLU (Flush) writes any compressed data left over from the last records transferred for compressoin and still present in the memory of the FLAMFILE and requests the statistical data. In contrast to FLMCLS, the FLAMFILE is not closed, i.e. another piece of compressed data can be appended.

FLMFRN FLMFRN (Find Record Number) positions the record pointer in an index sequential FLAMFILE (that has been created from a relative or sequential file) in such a way that the record with the specified record number can be read when FLMGET is subsequently activated.

FLMGET FLMGET (Get Record) reads one and only one decompressed record from a specified buffer.

FLMGHD Using FLMGHD (Get File Header) all header information containing the file format of the original file can be read. If more than one file header is contained in the compressed file, the information is valid for that records that are read next (FLMGET, FLMLOC).

FLMGKY With FLMGKY (Get KEY) it is possible to read a record by key out of an index sequential FLAMFILE. In addition the record pointer is positioned to the record with the nearest greater key to allow sequential read with FLMGET or FLMLOC.

FLMGRN FLMGRN (Get Record Number) reads the record with the specified record number from an index sequential FLAMFILE that has been created from a relative or sequential file.

FLMGTR FLMGTR (Get Reverse) reads the next decompressed original record, progressing towards the start of the file, into a specified buffer.

FLMGUH Information that has been added to the compressed data with FLMPUH during compression can be read by means of FLMGUH (Get User Header) during decompression.

FLMIKY With FLMIKY (Insert Key) a new record with new key will be taken on into the compressed file. The indicated key must not exist in the file.

FLMLCR FLMLCR (Locate Reverse) reads the next decompressed original record towards the beginning of the file in Locate Mode.

FLMLOC Instead of FLMGET function FLMLOC can be used to access to a decompressed record. FLMLOC will not pass a record to the caller, but an address pointer to the record.

FLMOPN The function FLMOPN (Open) has been partitioned into three subfunctions (FLMOPN, FLMOPD, FLMOPF) because of the large number of parameters. FLMOPN caters for the basic parameters (e.g., Compression/Decompression). FLMOPD defines the file formats of the FLAMFILE. FLMOPF defines the compression parameters. If the FLMOPD and FLMOPF subfunctions are not put to use, fixed values are used.

FLMPHD Function FLMPHD (Put File Header) will use its parameters to create a file header. The file header describes the format of the original file. The file header is valid for all following records from the original file received with FLMPUT until processing is closed or function FLMPHD is called again.

FLMPKY With FLMPKY (Put Key) it is possible to insert or update a record by key within an index sequential FLAMFILE.

FLMPOS FLMPOS (Position) is used for relative positioning in any file and for creating gaps when writing relative files.

FLMPUH FLMPUH (Put User Header) can be used to append another character string of any content to the information stored with FLMPHD. It can only be called immediately after FLMPHD has been called.

FLMPUT FLMPUT (Put Record) passes one record of the original file to FLAM.

FLMPWD FLMPWD gives in a password to FLAM for compression or decompression.

FLMQRY FLMQRY asks for special parameter that are not returned in other calls.

FLMSET FLMSET sets special parameter that are not set by other calls.

FLMUPD FLMUPD (Update) updates the last record read from an index sequential FLAMFILE.

2.4 I/O User Interface

This interface allows the user to integrate own access methods into FLAM.

One possibility is to pass the compressed records immediately to postprocessing routines without creating a compressed file. Vice versa, during decompression the compressed records may be received from a preprocess instead of reading them from a file.

A direct application could be the integration of FLAM with a file transfer application avoiding the creation of intermediate files.

Generally this interface allows to intercept all input and output data of both FLAM and FLAMUP. This allows the user to adapt FLAM easily to specific access methods.

2.5 User exits

2.5.1 Original data input EXK10

This user exit interfaces the record passed to FLAM for compression.

Special processing can be defined for: Start of file, record level, end of file. Records can be passed on, modified, deleted or inserted. This exit can be used to modify records in a structure dependent way.

EXK10 is only available in FLAM and FLAMUP and corresponds with EXD10 during decompression.

2.5.2 Compressed data output EXK20

This exit interfaces the compressed data before it is written into the FLAMFILE.

Special processing can be defined for: Start of file, record level, end of file. This exit can be used to modify records in a structure independent way.

With this exit it is possible, to modify the data with an own encryption routine, or a special code translation can be applied if a non-transparent file transmission method shall be used.It is possible to insert own records in front of the compressed records, for example archiving control records or origin information.

Another possibility is the extension of records to append specific revision information.

EXK20 is available in FLAM, FLAMUP and FLAMREC and corresponds with EXD20 during decompression.

2.5.3 Original data output EXD10

This exit interface the decompressed record immediately before it is written into the target file.

Special processing can be defined for: Start of file, record level, end of file. Records can be passed on, modified, deleted or inserted. This exit can be used to modify records in a structure dependent way.

EXD10 is only available in FLAM and FLAMUP and corresponds with EXK10 during compression.

2.5.4 Compressed data input EXD20

This user exit interfaces the compressed data immediately after it is read from the FLAMFILE.

Special processing can be defined for: Start of file, record level, end of file. This exit can be used to modify records in a structure independent way.

With this exit it is possible, to decrypt the data with an own decryption routine or to apply the reverse code translation as used during compression.

For a proper operation of FLAM it is indispensable that all changes applied to the compressed data are reversible. User exit EXD20 must deliver exactly the same data as user exit EXK20 received. All modifications applied to compressed data with EXK20 must be undone with EXD20.

EXD20 is available in FLAM, FLAMUP and FLAMREC and corresponds with EXK20 during decompression.

2.5.4 Compressed data input EXD20

This user exit interfaces the compressed data immediately after it is read from the FLAMFILE.

Special processing can be defined for: Start of file, record level, end of file. This exit can be used to modify records in a structure independent way.

With this exit it is possible, to decrypt the data with an own decryption routine or to apply the reverse code translation as used during compression.

For a proper operation of FLAM it is indispensable that all changes applied to the compressed data are reversible. User exit EXD20 must deliver exactly the same data as user exit EXK20 received. All modifications applied to compressed data with EXK20 must be undone with EXD20.

EXD20 is available in FLAM, FLAMUP and FLAMREC and corresponds with EXK20 during decompression.

2.5.5 Key management KMEXIT

This user exit returns a key to the FLAM utility for en-/decryption of a FLAMFILE.

So it is possible to enter any PASWORD/CRYPTOKEY in a secure way without notice to the JCL or the protocol.

This exit is implemented as an interface to special key management systems, without influence to the FLAM utility programs.

2.6 Bi-/serial compression with BIFLAMK

Bi-/serial compression does not employ FL-Matrices. The compression effect is achieved by comparing the original data with a sample and/or by using serial compression techniques.

BIFLAMK works synchronously. Each call results immediately in output data. It does not need a memory- compression is done on a call-by-call or record-by-record basis. Bi-/serial compression is especially suitable for integration into other products or applications.

The compression effect is considerably lower than with the compression using the FL-matrix. The advantage is the independence of each record. In many environments is this independency a necessary requirement for integration.

BIFLAMK offers beside compression two more functions that become more and more important as requirements for data security and data integrity are rising. All compressed records are encrypted and protected against modification using checksums over both original and compressed file.

BIFLAMK offers several compression modes that can be selected via a function code.

First, a vanilla serial compression mode is provided that does not require a sample record. All compressed records are independent from each other and can be decompressed individually.

Second, a bi-serial mode is provided that can be adapted to different environments. This compression mode is based on a comparison of the record with a sample record byte by byte.

The result is a bitmap that denotes all positions where characters are equal and the remaining differing characters.

The first option allows to control the postprocessing of the remainder. A serial compression can be applied to the remainder or the remainder is simply encrypted.

The serial compression of the remainder can be bypassed if the CPU-overhead is considered too high or the compression ratio is satisfactory.

The second option controls the processing of the sample. If a dynamic sample is used a checksum is computed for each sample record and is inserted into the compressed record. This reduces slightly the compression effect and increases CPU overhead. On the other hand data security is improved because modifications are detected more easily. For an exact error analysis it is even possible to decide whether the compressed record or the sample has been

modified. If a static sample is used, no checksum of the sample is computed. Modifications in the sample can in this case only be detected by using the checksum over the original file.

The third option allows the storage of samples within the compressed file. During decompression these records are restored again as samples. With this method BIFLAMK can create sequences of records that can be decompressed by BIFLAMD without additional information (samples).

One possibility would be to provide one sample first. Then all records are compressed against this sample and biserial compression is used to compress the remainders as well.

Another possibility is to use dynamic samples by using the processor of each record as the sample for the current record. This sequence result in good compression ratios if adjoining records are similar (Reports, data entry lists). The disadvantage is, that the individual records are not independent from each other. This sequence must be decompressed as a whole. However, additional information (like samples) is not necessary.

It does not make much sense, to use a different sample for each record. Sample records can only be compressed with simple serial compression. In addition the compressed data of the original data has to be stored as well.

2.7 Bi-/serial decompression BIFLAMD

BIFLAMD decompresses compressed records created with BIFLAMK.

For the serial decompression no sample record (including length) is used. This implies that BIFLAMD has to parameters less. Therefore it is necessary to inform BIFLAMD during the call, which mode (serial or bi-serial) shall be used for decompression.

For a proper decompression it is necessary that BIFLAMD receives exactly the same compressed records and sample records as created with BIFLAMK. Modification (Code translations) must not be applied to the compressed data and the sample records. If compressed data shall be transmitted between different computer the file transmission must be transparent.

BIFLAMD automatically detects if a record was compressed serial or bi-serial and will report an error message, if the compression syntax does not correspond with the call function code. Also modifications in compressed data, sample records and original data are detected via the check sums.

FLAM (MVS)

User Manual

Chapter 3:

Interfaces

Content

3. Interfaces

FLAM provides a set of interfaces that allow the use of the product within different environments and for different applications.

The simplest application is the execution of FLAM via the EXEC command. This allows to compress or decompress complete files.

In addition FLAM provides a set of subprogram interfaces for integration with other programs and products. This allows also to develop tailored applications where FLAM is embedded in dedicated control programs.

User exits provide the pre- and postprocessing of the original data as well as of the compressed data without the additional step via intermediate files.

All interfaces are designed in regard to higher level programming languages like COBOL. Only in cases where the usage of address pointers is a conditio sine qua non, assembler (or equivalent, e.g. C) interfaces must be used.

3.1 FLAM Utility

FLAM is able to compress complete files and to reconstruct complete files from compressed data.

For original files all file and record formats of type PS, PO or VSAM for disc or tape are supported.

The user interface for file IO (DEVICE=USER) allows to support additional access methods.

The user exits allow to pre- and post process original data as well as compressed data. The user exits are implemented as subprograms that are loaded dynamically during execution time from a module library (STEPLIB).

Using predefined or dynamically loaded conversion tables, character conversion can be applied to the original data.

It is possible to change file and record formats during decompression. Conversion from variable to fixed format or from sequential to index sequential organization is possible. The compressed data (FLAMFILE) can be stored in sequential or index sequential format in any record or file format. The record and file format of the compressed data is independent from the record and file format of the original data. An index sequential FLAMFILE provides efficient random access to original data by using the record level interface. At the other hand, sequential organization of the FLAMFILE is better suited for file transfer between computers with different operating systems.

FLAM compressed data is always heterogeneously compatible. This means that compressed data, which was generated under a certain operating system, can be decompressed on all operating systems supported by FLAM. Depending on the available record and file formats on the target systems, record and file conversions may be necessary.

FLAM can be executed online as well as in batch. It can be adapted in a flexible way to the requirements of the users. Several parameters are supplied for this purpose.

These parameters may be entered or may be stored via the PARM interface. An additional parameter file is also possible. Default values for the parameters may be created during installation (see: Generation of Default Values). File attributes can be also defined via the DD command.

During processing, parameters are evaluated in the following sequence:

First the installation parameters are used.

During decompression this values are substituted by the values from the file header of the FLAMFILE, if available.

Then the values from the parameter file are used.

The input PARM in the EXEC statement overwrites these values.

File attributes defined in the DD or ALLOCATE command overwrite again.

According to this hierarchy, a very flexible mode of operation is possible. Please be aware that the sequence is not always chronological:

E.g., it is possible to select a parameter file via PARM input. This parameter file is read in after the input despite of the fact, that the PARM inputs will overwrite the specifications of the parameter file.

3.1.1 Parameters

Independently from the input medium, parameters are evaluated always using the same syntax. Only upper case letters are allowed. Parameters may be passed in one or more lines or records. In each line the parameter interpretation ends with the first blank position. After the blank any type of comment may follow. Single parameters must not be separated by end of line.

Evaluation of the parameters is ended at the keyword END or by an empty input line (length = 0) or by EOF for the input medium.

There are parameters with or without key words. Key words and values can be abbreviated. Within brackets [ ] alternative key words are presented as used in other operating systems (MVS, DOS/VSE, BS2000). Key words from previous versions are presented within sharp brackets. This key words should not be used any longer.

Because of compatibility reasons all parameters are presented although not all parameters are evaluated under MVS.

Key word parameters can be specified in two different modes as usual under MVS:

```
parameter0,parameter1=value1,parameter2=value2,...
```

or:

```
parameter0,parameter1(value1),parameter2(value2),...
```

A string value (file names, module names, password, ...) may be passed as C'.....' (characters) or X'....' (hexadecimal).

All string parameters are assigned with blanks if "(NONE)" or no value is specified:

```
parameter=(NONE),   or   parameter(NONE),...
```

or:

```
parameter=,...  or parameter(),...
```

The sequence of parameters is arbitrary if not otherwise specified.

Only parameters that differ from the default values must be specified.

In the following we describe all parameters in alphabetic order:

Parameter may be abbreviated as long as they stay unique. Otherwise the first matching entry of the following summary is used.

ACCESS

Access method for the input or output file.

ACC

Possible values:

LOG	logical access by record
PHY	physical access by block
MIX	physical access by block with resolution into records
Default:	LOG
Valid for:	compression, decompression

Note: Ignored under MVS.

All files are read and written logically.

BLKSIZE

Logical block length for compressed file.

BLKS

Possible values:

0 - 32760

Default:	0 bytes
Valid for:	compression, decompression

Note: This parameter is not necessary for catalogued files in MVS.
A value of zero leads to a system determined blocksize.

CLIMIT

Minimal compression in per cent

CLI

Possible values:

0 - 90

Default:	0 no limit
Valid for:	compression

Note: If the compression result is worse than the predefined limit, FLAM will generate a message and will set condition code 80.

The compression is finished properly anyway. This parameter is only evaluated for INFO=YES.

CLOSDISP Final processing for compressed file on tape.

CLO Possible values:

REWIND	Rewind of tape to tape start.
UNLOAD	Rewind of tape and unload.
LEAVE	No rewind.
Default:	REWIND.
Valid for:	compression, decompression

Note: Currently ignored.

Control of final processing may be done via JCL (DD card).

COMMENT A comment to the FLAMILE.

COMM Will be stored into the user header of the FLAMFILE.

Possible values:

1 - 256 characters starting with A'...', C'...', X'...' or a string

Using A'..' all characters are translated to ASCII with the internal translation table A/E (ch. A).

Default:	No comment
Valid for:	compression

COMPRESS Compress

C Possible values:

1 - 64 characters starting with A'...', C'...', X'...' or a string

Using A'..' all characters are translated to ASCII with the internal translation table A/E (ch. A).

Default:	No values
Valid for:	compression

CRYPTOKEY Key to encrypt or decrypt a FLAMFILE

CRYPTOK This parameter activates the crytographic method, entered with parameter CRYPTOMODE.

Possible values:

1 - 64 characters starting with A'...', C'...', X'...' or a string

Using A'..' all characters are translated to ASCII with the internal translation table A/E (ch. A).

Default: no key

Valid for: compression, decompression

Note: Please take care of the different code tables or national character sets used on the different platforms.

E.g. using the password FLAM both on Windows systems (ASCII) and on MVS (EBCDIC) leads to a password error. You have to pass X'464C414D20' (this is 'FLAM ' in ASCII) or A'FLAM' on MVS instead.

We recommend to use the hex input for a heterogeneous environment.

CRYPTOMODE Choose the algorithm for encryption.

CRYPTOM Possible values:

AES Advanced Encryption Standard

FLAM the internal FLAM algorithm

Default: FLAM

Valid for: Compression.

Note: AES was introduced in FLAM V4.0 and is not compatible to older versions.

The encryption will be activated by the parameter CRYPTOKEY. The encryption mode is stored in the FLAMFILE, only the key is necessary on decompression and decryption.

Encryption implies MODE=ADC or NDC. Without entering a MODE-parameter ADC is used.

DATACLAS
DATAC

Data storage class for allocation of the FLAMFILE.

Possible values:

name	name of the SMS data storage class
Default:	none
Valid for:	compression

DECOMPRESS
D

Decompression.

[UNCOMPRESS]

No values

Valid for: decompression

DEVICE
DEV

Assignment of device assignment for compressed file.

Possible values:

DISK	disc unit
TAPE	tape unit
FLOPPY	floppy disc drive
STREAMER	tape streamer
USER	user specific I/O
Default:	DISK
Valid for:	compression, decompression

Note: This parameter is not needed for catalogued files under MVS. The device type is automatically assigned by the operating system.

If the user exit for I/O shall be activated, DEVICE=USER must be specified (see also: User I/O interface).

DSORG
DS

File organization for compressed file.

[FCBTYPE]

Possible values:

PS	sequential

ESDS	VSAM-ESDS
KSDS	VSAM-KSDS
LDS	VDSAM-LDS
RRDS	VSAM-RRDS
Default:	PS
Valid for:	compression

Note: If the compressed file shall be generated as index sequential file, a VSAM-KSDS-file must be created.

EXD10 Activate user exit for processing decompressed data.

EXD1 Possible values:

name	name of the module (max. 8 characters)
Default:	no user exit
Valid for:	decompression

The module is loaded dynamically.

EXD20 Activate user exit for processing compressed data.

EXD2 Possible values:

name	name of the module (max. 8 characters)
Default:	no user exit
Valid for:	decompression

The module is loaded dynamically.

EXK10 Activate user exit for processing original data.

EXK1 Possible values:

Name	name of the module (max. 8 characters)
Default:	no user exit
Valid for:	compression

The module is loaded dynamically.

EXK20 Activate user exit for processing compressed data.

EXK2 Possible values:

name	name of the module (max. 8 characters)
Default:	no user exit
Valid for:	compression

The module is loaded dynamically.

FILEINFO Transfer file name of original into file header.

FI Possible values:

YES	Transfer file name of original into/out of FLAM file header.
NO	Don't transfer file name (when compression). On decompression, a file name is generated (FILE0001-FILE9999) which can be used for conversion rules.
Default:	YES
Valid for:	compression, decompression

FLAMCODE Code for FLAM syntax.

FLAMC Possible values:

EBCDIC	FLAM syntax is created in EBCDIC code.
ASCII	FLAM syntax is created in ASCII code.
Default:	EBCDIC
Valid for:	compression

Note: If the original data are in ASCII character code, a better compression ratio is obtained using FLAMCODE=ASCII.

FLAMDDN Symbolic file name for compressed file.

FLAMD [FLAMLINK]

Possible values:

DD-NAME with max. 8 characters

> DD-NAME with max. 7 characters (decompression)

Default:	FLAMFILE
Valid for:	compression, decompression

Note: This parameter allows to change the DD name of the DD command. For decompression, '>' before the DD name means the file contains a list of FLAMFILE names.

FLAMFILE File name for compressed file.

FL Possible values:

File name	with max. 54 characters.
> File name	with max. 53 characters.

*DUMMY

Default:	no name
Valid for:	compression, decompression

Note: Specifying the file name is an alternative to assigning the file by means of a DD statement. The specification can contain a conversion rule for file names (see chapter

3.1.4). The name can be specified in wildcard syntax for decompression.

A '>' before the file name means the file contains a list of files to be decompressed.

*DUMMY acts like the DD-statement //.. DD DUMMY

FLAMIN

File name for the input file.

FLAMI

Possible values:

File name with max. 54 characters.

> File name with max. 53 characters.

*DUMMY

Default: no name

Valid for: compression

Note: Specifying the file name is an alternative to assigning the file by means of a DD statement. The file name can be specified in wildcard syntax (see chapter 3.1.4).

A '>' before the file name means the file contains a list of files to be decompressed.

*DUMMY acts like the DD-statement //.. DD DUMMY

FLAMOUT

File name for the output file.

FLAMO

Possible values:

File name with max. 54 characters

*DUMMY

Default: no name

Valid for: decompression

Note: Specifying the file name is an alternative to assigning the file by means of a DD statement. The specification can contain a conversion rule for file names (see chapter 3.1.4).

*DUMMY acts like the DD-statement //.. DD DUMMY

HEADER

Create file header.

HE

Possible values:

YES Create file header.

NO Don't create file header.

Default: YES

Valid for: compression

Note: The header consists of three parts. The first part is independent from the operating system and contains file attributes that are compatible. The second part depends on the operating system and contains file attributes specific for the according operating system. The third part is optional and contains the file name when specified with the parameter FILEINFO.

FLAM and FLAMUP evaluate the file header in order to create a file with the most similar characteristics. The simplest case is when the file is reconstructed in the original system environment. In this case the second (operating system specific) part of the file header can be used. In all other cases only the first part can be used and the system neutral attributes are mapped on system specific attributes of the target environment.

HELP

Help, output parameters.

No values.

Valid for: compression, decompression

Note: If the Help function is requested in the first input line, the generated FLAM parameters and their values are output and the program is then terminated.

IBLKSIZE

Logical block length for the input file.

IBLK

Possible values:

0 - 32760

Default: 32760 byte

Valid for: compression

Note: This parameter is not necessary for catalogued files in MVS.

ICLOSDISP

Final processing for tape input file.

ICLO

Possible values:

REWIND Rewind to tape start.

UNLOAD	Rewind and unload tape.
LEAVE	No rewind.
Default:	REWIND
Valid for:	compression

Note: Currently ignored. Control of final processing may be done via JCL (DD card).

IDDN

Symbolic file name for the input file

[ILINK]

Possible values:

DD-NAME with max. 8 characters

> DD-NAME with max. 7 characters

Default:	FLAMIN
Valid for:	compression

Note: This parameter allows to change the DD-NAME of the DD command.

A '>' before the file name means the file contains a list of files to be compressed.

IDEVICE

IDEV

Device assignment for input file.

Possible values:

DISK	disc unit
TAPE	tape unit
FLOPPY	floppy disc drive
STREAMER	tape streamer
USER	User specific I/O
Default:	DISK
Valid for:	compression

Note: This parameter is not needed for catalogued files under MVS. The device type is automatically assigned by the DMS. If the user exit for I/O shall be activated,

DEVICE=USER must be specified (see also: User I/O interface).

IDSORG

File organization for input file.

[IFCBTYPE]

Possible values:

PS	sequential
ESDS	VSAM-ESDS
KSDS	VSAM-KSDS
LDS	VSAM-LDS
RRDS	VSAM-RRDS
Default:	PS
Valid for:	compression

Note: This parameter is not necessary for catalogued files in MVS.

IKEYLEN

IKEYL

Key length of input file.

Possible values:

0, 1 - 255

Default:	No key
Valid for:	compression

Note: This parameter is not necessary for catalogued files in MVS. The key length specified in the catalogue is used.

IKEYPOS

IKEYP

Key position of input file.

Possible values:

0, 1 - (record length minus key length)

Default:	1 if key exists, 0 otherwise
Valid for:	compression

Note: This parameter is not necessary for catalogued files in MVS. The key position specified in the catalogue is used.

FLAM defines the position of the record key always as the relative position within the record data part - independently from the specific properties of the operating system. The first byte has position 1.

INFO

Control of protocol.

I

Possible values:

YES	Messages and statistics - generate and display.
NO	Don't display messages.
HOLD	Display the parameters for compression or decompression but do not execute compression or decompression.

Default: YES

Valid for: compression, decompression

Note: The INFO parameter should be contained in the first input line. Otherwise it has no effect on displaying the parameter inputs. The statistics inform about elapsed time and CPU time needed. Also the number of bytes and records for both input and output is computed. If relative files are decompressed the number of records after the removal of gaps is computed in addition. For conversion into fixed record format the resulting number of bytes is displayed.

INFO has been replaced by the more powerful parameter SHOW.

IRECDEL

Record delimiter for input original file.

IREC

Possible values:

String of up to 4 characters.

Default: no record delimiter

Valid for: compression

Note: Is not evaluated by FLAM under MVS.

IRECFM

Record format for input file.

[IRECFORM]

Possible values:

F	fix record length
V	variable record length
U	record length undefined
FB	fix blocked
VB	variable blocked
VBS	variable spanned
FBS	fix standard
Default:	VB variable blocked record format
Valid for:	compression

Note: This parameter is not necessary for catalogued files in MVS.

IRECFORM Record format for input file.

IRECF [IRECFM]

Possible values:

FIX	fix record length
VAR	variable record length
UNDEF	record length undefined
FIXBLK	fix blocked
VARBLK	variable blocked
VARSPAN	variable spanned
FIXS	fix standard
Default:	VARBLK variable blocked record format
Valid for:	compression

Note: This parameter is not necessary for catalogued files in MVS.

IRECSIZE Record length of input file (true lenght, without record length field.)

IRECS [ILRECL]

Possible values:

0 - 32760

Default: 32752

Valid for: compression

Note: This parameter is not necessary for catalogued files in MVS.

KEYDISP

KEYD

Key processing during decompression.

OLD	The records of the original file are reconstructed as they were read in (key + data).
DEL	If the original file has a key length unequal 0, the key is removed.
NEW	If the output file has a key length unequal 0, a key is generated at the key position in the specified key length. The key is generated from a record sequence number (16 characters max.).

Default: OLD

Valid for: decompression

Note: This parameter allows or simplifies the automatic conversion of sequential files into index sequential files and vice versa.

KEYLEN

KEYL

Key length of an index sequential compressed file.

Possible values:

0, 1 - 255

Default: 0 (no key)

Valid for: compression, decompression

Note: In an index sequential compressed file the key must be positioned at the beginning of the record. The key length should be equal to the sum of the length of all partial keys of the original file + 1. However, it is allowed to violate this rule.If sequential data is compressed into an index sequential compressed file, a key length of 5 bytes is sufficient.

KMEXIT Activate the key management exit.

KME Possible values:

name — name of the module (max. 8 characters)

Default: no user exit

Valid for: en-/decryption

The module is loaded dynamically.

Note: This Parameter overrules CRYPTOKEY

KMPARM Parameter used for the KMEXIT.

KMP Possible values:

Any input up to 256 characters in the form A'...', C'...', X'...', or as a string.

Default: no parameter

Valid for: en-/decryption

Note: This parameter overrules COMMENT, if any.

MAXBUFFER Maximum size of matrix.

MAXB Either a value between 0 and 7

value:	0	1	2	3	4	5	6	7
kByte:	32	32	64	128	256	512	1024	2048

or the matrix size in kBytes:

Minimal value: 8, maximal value: 2047

The value is rounded upwards according to the following table:

			8	10	12	14	16
32	48	64	80	96	112	128	144
176	224	256	288	320	352	384	416
512	640	768	896	1024	1536	2048	

or matrix size in bytes:

Minimal value: 2048

This value ist rounded upwards by kBytes according to the following table or is rounded downwards to 2560 kBytes:

2	4	6	8	10	12	14	16
32	48	64	80	96	112	128	144
176	224	256	288	320	352	384	416
512	640	768	896	1024	1536	2048	2560

default: 64 kByte

Valid for: compression, mode=cx7,cx8,vr8

Note: Because during decompression a buffer of the same size as during compression is needed, the compressed file is only heterogeneously compatible, if this buffer size is valid on the target system.

ADC/NDC use 64 kb, unchangeable.

The information is stored into the FLAMFILE, so it is well known during decompression.

MAXRECORDS

Maximum number of records for compression in one matrix.

MAXR

Possible values:

1 - 255 MO=CX7/CX8/VR8
1 - 4095 MO=ADC

Default: 255, 4095 depending on MODE

Valid for: compression

MAXSIZE

Maximum record length for compressed file.

(Net (true), without record length fields)

MAXS

Possible values:

80 - 32760

Default: 512 bytes

Valid for: compression

Note: The record length of the compressed file is independent from the record length of the original file. This pa-

rameter should be chosen according only to the aspect of efficiency and functionality. To avoid padding in blocks, for fixed record length the block length should be a multiple of the record length.

File transfer may require other record length (e.g.: 80 bytes fix for RJE (IBM) or 2036 bytes fix for transfer between SINIX and BS2000 with FT-BS2000).

MGMTCLAS Management class for allocation of the FLAMFILE.

MGMTC Possible values:

name	name of the SMS management class
Default:	none
Valid for:	compression

MODE Compression mode.

MO Possible values:

ADC	8 bit compression of highest efficiency
CX7	transformable 7 bit compression
CX8	8 bit compression (CPU time optimised)
VR8	8 bit compression (space optimised)
Default:	VR8
Valid for:	compression

Note: The compression mode is especially important for file transfer. For local use only the 8 bit modes should be used (ADC/CX8/VR8) for better efficiency.

For file transfer on transparent lines also the 8 bit modes should be used. For compressed text data (only printable characters - no control or tab characters) shall be transferred via non transparent lines the 7 bit mode (CX7) is appropriate.

MSGDDN Symbolic file name for message file.

MSGD [MSGLINK]

Possible values:

DD-NAME with max. 8 characters.

Default: FLPRINT

Valid for: compression, decompression

Note: With this parameter the DD-NAME in the DD command can be changed.

MSGDISP

MSGD

Device selection for message output.

Possible values:

TERMINAL	Currently not supported.
MSGFILE	Output into List file.
SYSTEM	Output on console with WTO, ROUTCDE=11

Default: MSGFILE

Valid for: compression, decompression

Note: The MSGDISP parameter should be contained in the first input line. Otherwise it has no effect.

MSGFILE

MSGF

File name for message file.

Possible values:

File name with max. 54 characters.

Default: no name

Valid for: compression, decompression

Note: Specifying the file name is an alternative to allocating the file by means of a DD statement.

OBLKSIZE

OBLK

Block length of the output file.

Possible values: 0 - 32760

Default: 0

Valid for: decompression

Note: A value of zero leads to a system determined blocksize for the output data set.

OCLOSDISP Final processing of tape output file.

OCLO Possible values:

REWIND	Rewind to tape start.
UNLOAD	Rewind and unload tape.
LEAVE	No rewind.
Default:	REWIND.
Valid for:	decompression

Note: Currently not supported. Please use JCL (DD statement) to control final processing.

ODATACLAS Data storage class for output file

ODATAC Possible values:

name	name of the SMS data storage class
Default:	none
Valid for:	decompression

ODDN Symbolic file name for output file.

[OLINK]

Possible values:

DD-NAME with max. 8 characters.

Default:	FLAMOUT
Valid for:	decompression

Note: With this parameter the DD-NAME in the DD command can be changed.

ODEVICE Device assignment for output file.

ODEV Possible values:

DISK	disc unit
TAPE	tape unit

FLOPPY	floppy disc drive
STREAMER	tape streamer
USER	User I/O interface
Default:	DISK
Valid for:	decompression

Note: This parameter is not needed for catalogued files in MVS.

If the user interface for I/O shall be activated, ODEVICE= USER must be specified (see also: User I/O interface).

ODSORG File organization for output file.

ODSO [OFCBTYPE]

Possible values:

PS	sequential
ESDS	VSAM-ESDS
KSDS	VSAM-KSDS
LDS	VSAM-LDS
RRDS	VSAM-RRDS
Default:	PS
Valid for:	compression

OKEYLEN Key length of output original file.

OKEYL Possible values:

0, 1 - 255

Default:	8 or the value from the file header
Valid for:	decompression

Note: This value must only be specified if the key length differs from the original file.

OKEYPOS

Key position of output file.

OKEYP

Possible values:

0, 1 - (record length minus key length)

Default: 1 or the value from the file header

Valid for: decompression

Note: This value must only be specified if the key length differs from the original file. The position of the record key is defined independently from the properties of a particular operating system as the relative position within the data part of the record. The first byte has position 1.

OMGMTCLAS

Management class for allocation of the output file

OMGMTC

Possible values:

name name of the SMS management class

Default: none

Valid for: decompression

ORECDEL

Record delimiter for output original file.

ORECD

Possible values:

String with max. 4 characters

Default: no record delimiter

Valid for: decompression

Note: Not evaluated by FLAM in MVS.

ORECFM

Record format for output file.

Possible values:

F fixed record length

V variable record length

U undefined record length

FB fix blocked

VB variable blocked

VBS	variable spanned
FBS	fix standard
Default:	VB or value from the file header
Valid for:	decompression

Note: This value must only be specified if the record format differs from the original file.

ORECFORM

ORECF

Record format for output file.

Possible values:

FIX	fixed record length
VAR	variable record length
UNDEF	undefined record length
FIXBLK	fix blocked
VARBLK	variable blocked
VARSPAN	variable spanned
FIXS	fix standard
Default:	VARBLK or value from the file header
Valid for:	decompression

Note: This value must only be specified if the record format differs from the original file.

ORECSIZE

ORECS

Record length of output file.

(True length without record length field.)

Possible values:

0 - 32760

Default:	32752 bytes or value from the file header
Valid for:	decompression

Note: This value must only be specified if the record length differs from the original file.

OSPACE1

Primary space allocation of the output file in Megabytes

Possible values:

1 - 4095

Default: 4

Valid for: decompression

Note: This value must only be specified if the FLAMFILE is non z/OS compressed or you need a special value. A z/OS FLAMFILE has stored the total space amount in the file header and uses this value for allocation on decompression..

OSPACE2

Secondary space allocation of the output file in Megabytes

Possible values:

1 – 4095

Default: 50

Valid for: decompression

OSTORCLAS

OSTORC

Storage class for allocation of the output file

Possible values:

name name of the SMS storage class

Default: none

Valid for: decompression

OUNIT

Unit for allocation of the output file

Possible values:

name name of the unit (e.g. SYSDA, 3390)

Default: none

Valid for: decompression

OVOLUME Volume for allocation of the output file

Possible values:

name	name of the volume (e.g. SYSWK1)
Default:	none
Valid for:	decompression

PADCHAR Padding character for an output record

Possible values:

X'..'	a hex value X'00' - X'FF'
C'.'	any character
Default:	X'40'
Valid for:	decompression

Note: This parameter is only used when an output record has to be filled up during decompression (e.g. the original variable record has to be converted to a fix output format).

PARDDN Symbolic file name for parameter file.

Possible values:

DD-NAME with max. 8 characters

Default:	FLAMPAR
Valid for:	compression, decompression

Note: This parameter allows to change DD-NAME in the DD command. If no symbolic file name is specified for the parameter file (PARDDN=(NONE)), FLAM will not try to read from the parameter file. If the parameter file is not existent or empty, no error is recorded.

PARFILE File name for parameter file.

PARF Possible values:

File name with max. 54 characters.

Default: no name

Valid for: compression, decompression

Note: Specifying the file name is an alternative to assigning the file by means of a DD statement. This file is needed only if additional parameters are to be read from a catalogued file.

PASSWORD

Password used for en-/decryption (same as CRYPTOK).

PASSW

Possible values:

Any input up to 64 characters (512 bit).

Default: no password

Valid for: compression, decompression

Note: Please take care of the different code tables or national character sets used on the different platforms.

E.g. using the password C'FLAM ' both on Windows systems (ASCII) and on MVS (EBCDIC) leads to a password error. You have to pass X'464C414D20' (this is 'FLAM ' in ASCII) on MVS instead.
We recommend to use the hex input for a heterogeneous environment.

RECDEL

Record delimiter for FLAMFILE.

RECD

Possible values:

String with max. 4 characters.

Default: no record delimiter

Valid for: compression, decompression

Note: Not evaluated by FLAM under MVS.

RECFM

Record format for FLAMFILE.

Possible values:

F	fixed record length
V	variable record length
U	undefined record length
FB	fix blocked
VB	variable blocked

VBS	variable spanned
FBS	fix standard
Default:	FB
Valid for:	compression, decompression

Note: The record format of the compressed file is independent from the record format of the original file. Fix blocked format is recommended.

RECFORM

Record format for compressed file.

RECF

Possible values:

FIX	fixed record length
VAR	variable record length
UNDEF	undefined record length
FIXBLK	fix blocked
VARBLK	variable blocked
VARSPAN	variable spanned
FIXS	fix standard
Default:	FIXBLK
Valid for:	compression, decompression

Note: The record format of the compressed file is independent from the record format of the original file. Fix blocked format is recommended.

SECUREINFO

Additional information stored in the FLAMFILE.

SEC

Increasing the security of data. Changing the FLAMFILE (in any way) leads to a decompression error.

Possible values:

YES	create these information (default on encryption) on compression
NO	do not store any additional data
IGNORE	ignore any security violations on decompression

MEMBER	only member-specific security informations are verified on decompression, not the entire FLAMFILE

Default: NO (without encryption)
YES (with encryption

Valid for: compression, decompression

Note: Concatenation of 'secure' FLAMFILEs lead to secure violations!

SECUREINFO=YES needs MODE=ADC or NDC.

SHOW

SH

Control of protocol

Possible values:

ALL	All messages and statistics - generate and display
NONE	Don't display messages.
ATTRIBUT	Display the parameters for compression or decompression but do not execute processing.
ERROR	Display error messages and end-of-program message only.
DIR	The names of all files (and their attributes) that are to be processed are listed.

Default: ALL

Valid for: compression, decompression

Note: The SHOW parameter is in effect after it is recognized.
The statistics inform about elapsed time and CPU time needed. Also the number of bytes and records for both input and output is computed. If relative files are decompressed the number of records after the removal of gaps is computed in addition. For conversion into fixed record format the resulting number of bytes is displayed. This parameter corresponds to the INFO parameter (see INFO).

SPACE1

Primary space allocation of the FLAMFILE in Megabytes

Possible values:

1 - 4095

Default: 2

Valid for: compression

SPACE2 Secondary space allocation of the FLAMFILE in Megabytes

Possible values:

1 – 4095

Default: 40

Valid for: compression

SPLITMODE Mode to split a FLAMFILE

SPLITM Possible values**:**

NONE	no split
SERIAL	serial split
PARALLEL	parallel split

Default: NONE

Valid for: compression

Note: Splitting of FLAMFILEs has been introduced in FLAM V4.0 and is not compatible to older versions.

The split information is stored in the FLAMFILE. So it is not necessary to use any parameter on decompression.

File- or DD-names must have numeric characters (ch. 3.1.5, or example in ch. 5.1.3).

SPLITNUMBER Number of fragments on parallel split

SPLITN Possible values:

2 - 4	number of simultaneously written files

Default: 4

Valid for: compression

Note: The information is stored in the FLAMFILE. So it is not necessary to use any parameter on decompression.

All fragments have to be catalogued and ready to read. It is not possible, to decompress one fragment alone.

Using this parameter needs SPLITMODE=PARALLEL.

SPLITSIZE Amount in MB of a fragment on serial split

SPLITS Possible values:

1 - 4095

Default: 100

Vaild for: compression

Note: The number of created files depends on the amount of compressed data. The information is stored in the FLAMFILE.

Using this parameter requires SPLITMODE=SERIAL.

STORCLAS Storage class for allocation of the FLAMFILE

STORC Possible values:

name name of the SMS storage class

Default: none

Valid for: compression

UNIT Unit for allocation of the FLAMFILE

Possible values:

name name of the unit (e.g. SYSDA, 3390)

Default: none

Valid for: compression

VOLUME
VOL

Volume for allocation of the output file

Possible values:

name	name of the volume (e.g. SYSWK1)
Default:	none
Valid for:	compression

TRANSLATE
TRA

Code conversion.

<CODE>

Possible values:

E/A	Converts EBCDIC to ASCII.
A/E	Converts ASCII to EBCDIC.
TRE2A00	Converts IMB273 to ISO 8859-1.
TRA2E00	Converts ISO 8859-1 to IBM273.
name	Name of a data module (1-8 characters) that contains a 256 byte table for code conversion.
Default:	no code conversion
Valid for:	compression, decompression

Note: With this function original data characters can be translated before compression or after decompressing before storage.

The specified table is loaded dynamically.

Code conversion may be necessary for file transfer between heterogeneous systems. It can be performed on any system, but it is recommended to do it on the target system, since FLAM already contains the necessary conversion tables for that system.

Please look for the translation tables used by FLAM in chapter A. The library FLAM.SRCLIB contains examples of table modules.

Example:

```
CODETAB   CSECT
TAB       DC      256AL1(*-TAB)
          ORG     TAB+X'0C'
          DC      X'F1'
```

```
ORG     TAB+C'A'
DC      C'B'
ORG
END
```

If TRA=CODETAB was specified, the original data is converted in the following way: From X'OC' to X'F1' and each letter A to B.

TRUNCATE

Truncate output record.

TRU

<OPTION=CUT/NOCUT>

Possible values:

YES	If a decompressed record is longer than specified for the output file, the record is truncated.
NO	No truncation. If records longer than specified occur, decompression is aborted.
Default:	NO
Valid for:	decompression

3.1.2 JCL for FLAM

This description is valid for batch processing. Appropriate commands must be used for dialogue processing (TSO). FLAM is started via the EXEC command:

```
//stepname  EXEC PGM=FLAM,PARM= ...
```

Parameter specification must be made according to the JCL conventions (max. 100 characters, inclusion in quotes in case of special characters, like '=', '(', etc.).

FLAM parameters specified via PARM= will overwrite FLAM parameters stored in the FLAM parameter file.

If the FLAM modules are not stored within a system library a FLAM load library must be specified:

```
//STEPLIB  DD   DSN=user.FLAM.LOAD,DISP=SHR
```

The input and output files can be assigned to FLAM via DD cards. The DD names used are predefined default names. Other DD names can be chosen via parameters.

The following file types are supported by FLAM:

- Physical sequential PS (also members of a PO library)
- PO libraries
- VSAM ESDS / KSDS / RRDS / LDS

The following record formats are supported:

```
V / VB / VS / VBS / F / FB / FS / FBS / U
```

Print files (A or M) are also supported.

The specification of a parameter file is possible but not mandatory (usually the PARM= instruction is sufficient):

```
//FLAMPAR  DD  DSN=parameter_file,DISP=OLD
```

It is also possible to define the parameter file directly imbedded in the JCL:

```
//FLAMPAR  DD  *
 parameter0,parameter1
 parameter2
/*
```

For compression an **input file** must be specified:

```
//FLAMIN   DD  DSN=uncompressed_input_file,
//             DISP=SHR
```

This file must already exist and must be catalogued. It is allowed to be "logically empty", which means that it does not contain a data record.

For the creation of a compressed file (**FLAMFILE**) it is sufficient to write:

```
//FLAMFILE  DD DSN=compressed_file,
//             DISP=(NEW,CATLG),
//             UNIT=....,
//             SPACE=....
```

With this card a sequential data set with a fixed record length of 512 and a system determined block length will be created. However, this may depend on the default parameters specified or on additional parameters supplied with the job.

```
//FLAMFILE  DD DSN=compressed_file,
//             DISP=(NEW,CATLG),
//             UNIT=...,SPACE=...,
//             DCB=(LRECL=1024,BLKSIZE=26624)
```

Using DCB attributes this assignment overwrites the MAXSIZE or BLKSIZE parameters for the FLAMFILE. The file is created according to the specifications in the DD statement.

One example for using flexible configurations with the FLAMFILE comes from the area JES / NJE:

```
//FLAMFILE  DD  SYSOUT=F,DEST=(node,userid),
//              DCB=(LRECL=80,BLKSIZE=3120),...
```

If the FLAMFILE is already catalogued (DISP=OLD) FLAM will use the configuration details from the catalogue entry (even, if it is a VSAM file).

For decompression an **output file** must be assigned.

```
//FLAMOUT  DD  DSN=decompressed_file,
//             DISP=(NEW,CATLG),
//             UNIT=...,SPACE=...
```

With this specification a decompressed file will be created that has identical characteristics regarding file format, record and block length with the original file.

If a **PO library** has been compressed and a new PO file is to be defined for decompression by JCL, the selection rule FLAMOUT=<*> (see chapter 3.1.4.3) must always be specified. Otherwise member names are not known.

```
//EXEC PGM=FLAM,PARM='D,FLAMOUT=<*>'
//FLAMOUT DD DSN=po_file,DISP=OLD
```

If the original file came from a foreign system (VM, VSE, UNIX, Windows, ...), FLAM will choose a format closest to the original file.

If the FLAMFILE does not contain information about the original file, e.g. caused by an 'HEADER=NO' parameter during compression, the output file is created as variable blocked (VB) and with a maximum record length of 32756 bytes and a block length of 32760 bytes.

Each DCB specification within a DD statement will overwrite the values automatically chosen by FLAM or specified as FLAM parameter.

If the output file is already catalogued the catalogue entry will specify the file characteristics:

```
//FLAMOUT  DD  DSN=output_file,DISP=OLD
```

All **messages** are written by default into a message file:

```
//FLPRINT  DD  DSN=list_file,DISP= ...
```

or

```
//FLPRINT  DD  SYSOUT=*
```

If no file is specified despite the fact that the parameter MSGDISP=MSGFILE is set, a message is written to the console via WTO, ROUTCDE=11 and the program is terminated with a condition code.

If messages shall be suppressed, SHOW=NONE must be specified in the PARM= entry within the EXEC card.

Each file can be assigned as:

```
//ddname  DD  DUMMY
```

This will suppress file output or inhibit file input.

Since this does not affect the general processing of FLAM, it can be used conveniently for testing purposes:

- evaluation of compression ratio
- general testing of control flow
- using own read and write routines from user exits (we recommend instead the USERIO interface, but in principle the possibility mentioned above can be used, too).

The DUMMY assignments are not suitable for benchmarks. Because all physical disc or tape I/O is suppressed, this leads to a much shorter elapsed time, especially in case of later tape assignments or VSAM I/Os.

Note: The record length in the DD statement must always include the length of the record length field in case of variable length records.

Opposite to that, FLAM requires the actual length of the data (without length field) in it's own parameters and displays also only the actual length of the data in the protocol. To make this difference clear the parameter name LRECL has been changed to MAXSIZE for the FLAMFILE parameter printout. IRECSIZE and ORECSIZE are equivalent notations for the input file and the output file.

The FLAM user interface (see chapter 9) for TSO users removes the task of writing JCL statements from the user (including creation of VSAM files). All specifications can be made interactively. Also batch job streams can be generated with this facility.

3.1.2.1 Dynamic file allocation

The specified files are allocated automatically by FLAM (dynamic allocation, SVC 99) by means of parameter input (FLAMIN=filename, FLAMFILE=filename, FLAMOUT=filename, FLAMOUT=<*>, ...), as long as a DD statement has not been specified.

If the files are already catalogued, FLAM will use the configuration details from the catalogue entry. For reading the file is assigned with 'DISP=SHR', for writing with 'DISP=OLD'. If the file is not already catalogued, it is created ('DISP=(NEW,CATLG)').

For decompression (FLAMOUT=...) data such as file organization, record length, block length, format and file size are taken from the FLAM file header, which means that the output file essentially corresponds to the original file. Parameter inputs, however, take priority over the stored data. When the parameter FLAMOUT=<*> is set, the file names stored in the file header are used (see also chapter 3.1.4).

The storage assignment contains the file size as primary specification, 1/4 of the file size being the secondary specification. In the case of PO libraries, the number of directory entries is also known. The file is thus stored in one extent on the disk.

If the file size is not known, because e.g. a FLAMFILE is assigned or the compressed data has not been generated under MVS (and therefore does not contain the file size), default values are used (TRK,(30,900),RLSE).

If the default values are inadequate, a DD statement specifying other values must be made or the file can be created before the job is run (e.g. with function 3.2 in ISPF).

All file types supported by FLAM (PS, PO, and VSAM) can also be created dynamically again. However, since important information required for creating new files is missing (such as UNIT and VOLUME), SMS must be used. The fact that this information is missing is on the one hand due to data protection reasons (data exchange via file transfer!); on the other hand, they would usually not be useable due to transferral of the compressed data to other computers. After all, compressed data from foreign operating systems (VMS, VSE, UNIX, etc.) cannot even contain these specifications.

Note: If parameters are specified and FLAM finds a DD statement via the DD-NAME, all the specifications in this statement take priority over the parameters specified or values stored in the file header.

3.1.3 Condition Codes

For job control FLAM returns the following condition codes:

- 0 Normal termination (no error)
- 4 Not all I/O files were processed during processing of group files
- 8 Less severe error occurred (e.g. parameter error)
- 12 Usually a data management error
- 16 Severe error during compression or decompressi on
- 80 Compression ratio was less than the specified limit (see CLIMIT parameter)
- 88 The file assigned was not a FLAMFILE

Only for condition code 0 and 80 compression was done properly. In all other cases no or a faulty compressed file was created. We recommend to rename this file in order to avoid using it for processing at a later time.

If a condition code greater 0 was returned an error message was already displayed by FLAM.

Condition code 16 could be caused by a FLAM system error.

The library FLAM.SRCLIB contains the call module FLAM. This can be customized as required, allowing other condition codes to be returned.

3.1.4 File names

In principle, FLAM is able to process all file names via JCL valid in MVS.

FLAM stores the name of the compressed file into the FLAMFILE, if required. On decompression file names are found that are not common to the z/OS system.

To avoid any conflicts with national character sets or naming conventions in other systems, all file names stored in ASCII character set are translated for message and selection in the following way:
all national characters are translated to 'X', a backslash '\' to slash '/', and blanks ' ' to underline '_'.

So it is easier to enter foreign file names that are unsupported in the z/OS environment. The file name itself remains unchanged in the FLAMFILE.

Entering '*DUMMY' as a file name causes FLAM to use this file as dummy like the JCL command //ddname DD DUMMY. I.e. reading an input file leads to EOF (end of file), writing to an output file has no effect.
So DD-statements are not longer necessary for DUMMY files.

3.1.4.1 File name list

By prefixing the '>' sign (greater than) to the file name or DD-NAME, the FLAM parameters FLAMIN and IDDN are able to specify a list of files for compression instead of one single file.

It is also possible to specify a file list for decompression by means of the FLAMFILE or FLAMDD parameter.

In this list, each file name must be contained in a separate record; leading or trailing space characters (X'40') are ignored. A comment can be inserted after the first space character following the file name.

Empty records or records with an asterisk '*' in the first column are regarded as comment lines.

All file names used in MVS are legal. Wildcard syntax is allowed.

Example: If the records in the file USER.DAT.LIST contain the following file names:

```
USER.DAT.PS
USER.VSAM.ESDS
USER.POLIB.*
USER.PO(MEMBER)
```

the specification

```
//... EXEC PGM=FLAM,PARM='C,FLAMIN=>USER.DAT.LIST'
```

leads to all the files specified being compressed into one FLAMFILE (group file).

The file that contains the list of file names can have any format supported by FLAM and be of any type.

For 'in-stream files', i.e. input files temporarily created by JES, it is advisable to assign the file names by means of DD names:

```
//... EXEC PGM=FLAM,PARM='C,IDDN=>DDNAME'
//DDNAME  DD  *
USER.DAT.PS
USER.VSAM.ESDS
USER.POLIB.*
USER.PO(MEMBER)
/*
```

This allows the file name list to be specified directly in the job.

3.1.4.2 Wildcard syntax

File names can be specified in FLAM by means of parameters written with wildcard syntax. Correspondingly, the entry

```
FLAMIN=USER.*.DATA.%BC
```

leads to the compression of all files with the 1st qualifier USER, any 2nd qualifier, DATA as the third part of their name, and a three-digit 4th qualifier which ends with BC and begins with any character.

The asterisk '*' stands for any (even an empty) character string.

The percent sign '%' stands for any character.

These special characters are also allowed within members of one or more PO libraries:

```
USER.POLIB(FL*)
```

specifies all members of the library USER.POLIB, that begin with FL;

```
USER.*D.LIB(A%%B)
```

specifies all members whose 4-digit names begin with A and end with B and whose libraries have the identification USER, LIB as their final qualifier, and D as the last character of the second to the last part of their name.

All input files are in this way stored in one compressed file (group file).

Correspondingly, the wildcard entry in the selection rule (see chapter 3.1.4.3) for decompression

```
FLAMOUT=<USER.DAT.*LIB>
```

leads to the decompression of only those items of compressed data whose original names conform to the syntax specified. (Comment: this procedure requires a FLAM file header (HEADER=YES) and the file information (FILEINFO=YES) for compression).

Example:

```
...C,FLAMFILE=USER.DAT.CMP,FLAMIN=USER.*B.*LIB,...
```

All files with names that conform to the FLAMIN specification are to be compressed into the file USER.DAT.CMP.

If the files

```
USER.DATA.ALIB
USER.DATAB.BLIB
USER.DATCB.CLIB
USER.DATCB.DLIB
```

are catalogued, the 1st file does not conform to the wildcard syntax and is ignored during compression.

If

```
´D,FLAMFILE=USER.DAT.CMP,FLAMOUT=<USER.DATCB.*LIB>´
```

is now specified for decompression, only the files USER.DATCB.CLIB and USER.DATCB.DLIB will be decompressed.

Correspondingly, it is also possible to select a number of FLAMFILEs for decompression:

```
..D,FLAMFILE=USER.CMP.*.VR8,...
```

means all FLAMFILEs with the identification USER, CMP as their 2nd qualifier, any character string as the 3rd part of their name and VR8 as their last qualifier are to be decompressed.

3.1.4.3 Selection rule for decompression

During decompression, the files can be created and catalogued by FLAM automatically. This requires the existence of a valid file name in the file header of the FLAMFILE (i.e. the parameters HEADER and FILEINFO must not be set to NO for compression).

In addition, a FLAMFILE can contain several files (i.e. it is a group file) and it is possible to decompress specific files from such a group FLAMFILE on a targeted basis by specifying a selection rule.

In order to distinguish it from a 'genuine' file name, a selection rule is written between pointed brackets '<>'.

```
FLAMOUT=<USER.FILE.ORG>
```

With the above rule set, the file USER.FILE.ORG is decompressed from the FLAMFILE. (Comment: if the pointed brackets were left out, the entire FLAMFILE would be decompressed and written into the file USER.FILE.ORG !) This name must be contained in a file header of the FLAMFILE.

If the FLAMFILE contains other compressed files, they are ignored due to the unique selection rule set.

If several files are to be decompressed from a group file, a wildcard syntax can be specified. The simplest entry possible is a lone asterisk:

```
FLAMOUT=<*>
```

This decompresses all of the files from the FLAMFILE and locates them on the disk under their original names.

A selection rule is implicitly regarded by FLAM as having an asterisk at the beginning and at its end, i.e.:

```
<DAT*ABC > corresponds to <*DAT*ABC*>
```

When analyzing the file names, FLAM synchronizes itself to the character string specified.

Example: The FLAMFILE contains the data USER.DAT1.PS and USER.DAT2.PO. The specification

```
...D,FLAMOUT=<DAT1>,...
```

decompresses only the file USER.DAT1.PS.

Since there is no asterisk specified in the name, decompression is termimated after the first hit.

Note: If, in addition to the selection rule, a file is assigned by means of JCL, the JCL specification takes priority. I.e., the above specification together with the DD statement

```
/FLAMOUT DD DSN=USER2.POLIB(MEMBER),DISP=...
```

leads to decompression of the USER.DAT1.PS file from the FLAMFILE and it is written into the PO library USER2.POLIB as the member MEMBER.

This, then, allows the changing of file names during decompression by means of JCL specifications.

3.1.4.4 Conversion rule

This simple method of selecting files for decompression by means of a selection rule can, however, not normally be used for compressed data that has been generated under a different operating system (heterogeneous exchange of compressed data). The file names do not normally conform to the rules of the MVS operating system and can therefore not be used without being modified.

For this purpose, it is possible to specify a conversion rule as the parameter for the output file. This character string describes how a new name is to be generated from a file name selected. At the same time, precisely those files that conform to the rule (selection rule) are selected.

A conversion rule is a selection rule that is extended by an equals sign '=' and a second character string. In order to distinguish it from a 'genuine' file name, it must be written between pointed brackets '<>'. The rule comprises a character string that may contain an asterisk '*' as a substitute character for any number of characters or a percent sign '%' as a substitute for precisely one character. An ignore character (apostrophe ') is also defined.

Each asterisk '*' or percent sign '%' in the selection rule must be assigned an asterisk or percent sign or an apostrophe in the conversion rule.

The asterisk means that the character string in the input file is to be transferred unchanged to the output file. Correspondingly, the '%' transfers precisely that character located in this position.

The apostrophe sees to it that the character string or character represented in the input file by an asterisk or percent sign respectively, is not transferred to the output file. The remaining characters from the input file are translated into the corresponding characters from the conversion rule. The length of the character string can be changed as desired.

```
<USER.*=USER2.*>
```

With this rule, all file names beginning with USER are converted to the new identification USER2. The rest of the name is retained.

```
<USER.DAT%B.*=USER.DEC.DAT%C.*>
```

With this rule, all files with the identification USER are given the prefix DEC before their old name. In the process, only those files are affected whose second part of their name begins with DAT, is followed by any character and ends with B. This B is converted to C in the name of the output file.

Above all, empty character strings are also allowed in the conversion rule, so that characters can be deleted. E.g.:

```
<USER*UP*=USER.CMP**>
```

Old name: `USER.FLAMUP00`
New name: `USER.CMPFLAM00`

The UP part of the name is not mentioned in the output name and is therefore left out.

A conversion rule is implicitly supplemented by FLAM, e.g.:

```
<ASM.=CMP.> corresponds to <*ASM.*='CMP.*>
```

This can be particularly useful when converting the file names from other systems.

Example:

The FLAMFILE created under DEC/VMS contains the following file names:

```
DUA1:[ABC]DE0051.;7
DUA1:[ABC]DE0052.;4
DUA1:[ABC]DE0080.;2
DUA1:[ABC]DE0152.;4
```

This corresponds to specification of the file versions of the user ABC on disk volume DUA1, with the version number stated after the semicolon.

In order to be able to create these files in MVS, the following conversion rule could be specified:

```
FLAMOUT=<DE*.;*=USER.DE*'>
```

This implicitly deletes the name prefix DUA1:[ABC], takes over the part of the name beginning with DE and supplements it with the identification, and deletes the rest of name:

```
USER.DE0051
USER.DE0052
USER.DE0080
USER.DE0152
```

The conversion rule is also important with respect to generating members of a PO library:

```
FLAMOUT=<USER.*.LIST=USER.POLIB(*)>
```

This rule means that the 2nd name qualifiers of the original file are used as member names of the PO library.

So far, the conversion rule has only been described for decompression from a (group) FLAMFILE.

It can, however, also be used for compression with simultaneous generation of several items of compressed data in different files. In this respect, the conversion rule relates to the FLAMFILEs whose names are generated from the file names of the input files (FLAMIN).

This is how all compressed data could be given a prefix:

```
C,FLAMIN=USER.*.LIST,FLAMFILE=
<USER.*.LIST=USER.CMP.*.LIST>
```

It is, however, also possible to store each item of compressed data as a member of a library by means of the following specification:

```
C,FLAMIN=USER.*.LIST,FLAMFILE=
<USER.*.LIST=USER.POLIB(*)>
```

The member name is, in this case, the 2nd qualifier of the input file.

Also valid here: if a library is specified for the FLAMFILE by the JCL, the name of the library as specified in the rule is ignored and only the members in the specified PO file are generated.

The reverse also applies: file names of the FLAMFILEs are used to generate file names for the decompressed files:

```
D,FLAMFILE=USER.CMP.*,FLAMOUT=
<USER.CMP.*=USER.*.LIST>
```

Note: If a group file has been created with HEADER=YES but FILEINFO=NO, no file name has been stored for this file.

The individual files can then be accessed for decompression via the internal file name: the name is in the range from FILE0001 (for the first file) to FILE9999 (for the 9999th file).

```
...D,FLAMOUT=<FILE0003=USER.DAT.THREE>,..
```

is for the third file in the group file; or

```
...D,FLAMOUT=<FILE*=USER.DAT*>,..
```

is for decompressing all files according to the conversion rule.

Comment: As a "final rescue option" for automatic generation of the decompressed data with "impossible" file names from foreign operating systems, the parameter FILEINFO=NO can be specified for the decompression process. With this parameter, the stored file names are ignored and the internal names FILE0001 to FILE9999 are generated. These must then be converted to valid file names by means of a conversion rule.

3.2 Subprogram interface FLAMUP

In the following we describe the interface using ASSEMBLER language. Therefore the following table shows, how the different data types must be defined in COBOL and FORTRAN.

With FLAMUP it is possible to compress a file completely or to decompress a compressed file. Similar to the FLAM utility the parameters must be passed to FLAMUP. FLAMUP uses the same parameters as the FLAM utility. All parameters defaults can be defined during generation.

Assembler	Cobol	Fortran	Meaning
F	PIC S9 (8) COMP SYNC	INTEGER*4	Aligned Full-word
H	PIC S9 (4) COMP SYNC	INTEGER*2	Aligned Half-word
CL*n*	PIC X (*n*) USAGE DISPLAY	CHARACTER**n*	*n* printable characters
XL*n*	PIC X (*n*)	CHARACTER**n*	*n* bytes

The arrows define the direction of data flow:

→	the field must be filled by the calling program
←	the field is filled by the called program
↔	the field is filled by the calling program as well as by the called program

Parameters:

1 ← **FILEID** **F** Identification

2 ← **RETCO** **F** Return code

= 0 No error

some usual codes, for all return codes see chapter 8

= 1 Records truncated
= 9 CLIMIT exceeded
= 10 File is not a FLAMFILE
= 11 FLAMFILE format error
= 12 Record length error
= 13 File length error
= 14 Checksum error
= 15 Original record is greater than 32764 bytes
= 16 Original record is greater than matrix - 4
= 20 Invalid OPENMODE

=	**21**	Invalid size of matrix buffer

=	**22**	Invalid compression mode
=	**23**	Invalid code in FLAMFILE
=	**24**	Invalid MAXRECORDS parameter
=	**25**	Invalid record length MAXSIZE
=	**29**	Password error
=	**30**	FLAMFILE is empty
=	**31**	FLAMFILE is not assigned
=	**32**	Invalid OPENMODE
=	**33**	Invalid file type
=	**34**	Invalid record format
=	**35**	Invalid record length
=	**36**	Invalid block length
=	**37**	Invalid key position (not 1)
=	**38**	Invalid key length
=	**39**	Invalid file name
=	**x'Exxxxxxx'**	FLAMFIO error for original file input
=	**x'Axxxxxxx'**	FLAMFIO error for original file output
=	**x'Fxxxxxxx'**	FLAMFIO error for compressed file
=	**x'Cxxxxxxx'**	FLAMFIO error for parameter file
=	**x'Dxxxxxxx'**	FLAMFIO error for message file
=	**x'xFxxxxxx'**	Error in data management (VSAM)
=	**40**	Module or table cannot be loaded
=	**41**	Module cannot be called
=	**42**	Module cannot be unloaded
=	**43-49**	Abort by user exit
=	**52**	Too many or invalid keys
=	**80**	Syntax error during parameter input
=	**81**	Unknown parameter (key word)
=	**82**	Unknown parameter value
=	**83**	Parameter value not decimal
=	**84**	Parameter value too long
=	**96**	No file name found or error when determining file names
=	**98**	Not all files were processed
=	**999**	Error during memory request

3 → **PARAM** **Cl*n*** Parameter area

4 → **PARLEN** **F** Length of parameter area

=	0	No parameter
>	0	Length (n) of parameter area

Note: Parameters must be written in the same way as in the utility.

Only upper case letters are allowed!

Example for the call of FLAMUP in COBOL:

```
  IDENTIFICATION DIVISION.
    PROGRAM-ID. EXAMPLE.
*
* EXAMPLE FOR THE CALL OF FLAMUP
*
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  77 FLAMID   PIC S9(8) COMP SYNC.
  77 RETCO    PIC S9(8) COMP SYNC.
  77 PARAM     PIC X(80).
     VALUE 'C,MODE=ADC'
  77 PARLEN   PIC S9(8) COMP SYNC VALUE 10.
*
 PROCEDURE DIVISION.
*
 CALL 'FLAMUP' USING FLAMID, RETCO, PARAM, PARLEN.
*
  STOP RUN.
```

Example for the call of FLAMUP in ASSEMBLER:

```
EXAMPLE  CSECT
         .
         .
         .
*
*   CALL FLAMUP
*
         LA       1,FLAMUPAR
         L        15,=V(FLAMUP)
         BALR     14,15
         .
         .
         .
*
*   PARAMETER FOR FLAMUP
*
FLAMUPAR    DC    A(FLAMID)
            DC    A(RETCO)
            DC    A(PARAM)
            DC    A(X'80000000'+PARLEN)
*
FLAMID      DS    F
RETCO       DS    F
PARAM       DC    C'C,MODE=ADC'
PARLEN      DC    F'10'
*
*
*   SAVEAREA
*
SAVEAREA    DS    18F
            END
```

Register usage for ASSEMBLER:

```
→ R1:      address of parameter list
→ R13:     points to save area (18 words)
→ R14:     contains return address
→ R15:     contains call address
```

Example for the call of FLAMUP in C ++:

```
// an example for calling flamup from C++
//
// set linkage convention
extern "OS" void FLAMUP(void **,long *,char
*,long *);
int main()
{
 void *flamid;
 long retco;
 long parlen=10;
 char param[10]="C,MODE=ADC";
 FLAMUP(&flamid,&retco,param,&parlen);
 return 0;
}
```

3.3 Record level interface FLAMREC

FLAMREC consists of a number of subroutines that can be called by any programming language such as COBOL, 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.

All parameter lists start with an identificator. This identifies the compressed file between FLMOPN and FLMCLS. 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 FLAMREC should be specified explicitly when linking.

Example for the call of FLMOPF in COBOL:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EXAMPLE.
*
*  EXAMPLE FOR THE CALL OF FLMOPF
*
ENVIRONMENT DIVISION
DATA DIVISION.
WORKING-STORAGE SECTION.
77  FLAMID        PIC S9(8) COMP SYNC.
77  RETCO         PIC S9(8) COMP SYNC.
77  VERSION       PIC S9(8) COMP SYNC.
77  FLAMCODE      PIC S9(8) COMP SYNC.
77  COMPMODE      PIC S9(8) COMP SYNC.
77  MAXBUFF       PIC S9(8) COMP SYNC.
77  HEADER        PICS9(8)  COMP SYNC.
77  MAXREC        PIC S9(8) COMP SYNC.
77  BLKMODE       PIC S9(8) COMP SYNC.
77  EXK20         PIC X(8)  VALUE SPACES.
77  EXD20         PIC X(8)  VALUE SPACES.
01  KEYDESC.
    05 KEYFLAGS   PIC S9(8) COMP SYNC.
    05 KEYPARTS   PIC S9(8) COMP SYNC.
    05 KEYELEM    OCCURS 8 TIMES.
       10 KEYPOS  PIC S9(8) COMP SYNC.
       10 KEYLEN  PIC S9(8) COMP SYNC.
       10 KEYTYPE PIC S9(8) COMP SYNC.
*
PROCEDURE DIVISION.
*
    CALL "FLMOPF" USING  FLAMID, RETCO, VERSION
    FLAMCODE, COMPMODE, MAXBUFF, HEADER,
    MAXREC,KEYDESC, BLKMODE, EXK20, EXD20.
    .
    .
    .
```

Example for the call of FLMOPF in ASSEMBLER:

```
EXAMPLE     CSECT
            .
            .
            .
*
*  DEFINE VALUES
*
            LA    0,3
            ST    0,COMPMODE  COMPMODE = ADC
            LA    0,1
            ST    0,HEADER    HEADER = YES
            ST    0,BLKMODE   BLKMODE = YES
            LA    0,4095
            ST    0,MAXREC    MAXRECORDS = 4095
            L     0,=F'65536'
            ST    0,MAXBUFF   MAXBUFFER = 65536
            LA    0,0
            ST    0,KEYPARTS  KEYPARTS = 0
            ST    0,KEYPOS1   KEYPOS1 = 0
            ST    0,KEYFLAGS  = NO DUPLICATE KEY
            ST    0,KEYTYPE1  = PRINTABLE
            ST    0,KEYLEN1   KEYLEN1 = 0
            MVI   EXK20,C' '  NO EXK20
            MVI   EXD20,C' '  NO EXD20
*
*  CONSTRUCT PARAMETER LIST FOR FLMOPF
*
            LA    15,VERSION
            ST    15,ARVERSIO
            LA    15,CODE
            ST    15,ARCODE
            LA    15,COMPMODE
            ST    15,ARCOMPMO
            LA    15,MAXBUFF
            ST    15,ARMAXBUF
            LA    15,HEADER
            ST    15,ARHEADER
```

```
            LA    15,MAXREC
            ST    15,ARMAXREC
            LA    15,KEYDESC
            ST    15,ARKYDESF
            LA    15,BLKMODE
            ST    15,ARBLKMOD
            LA    15,EXK20
            ST    15,AREXK20
            LA    15,EXD20
            ST    15,AREXD20
            OI    AREXD20,X'80' last parameter
*
*  CALL FLMOPF
*
            LA    1,RECPAR
            L     15,=V(FLMOPF)
            BALR  14,15
            .
            .
            .
*
*  PARAMETER LIST FOR FLMOPN
*
RECPAR   DS    0A
ARFLAMID DS    A   ADDRESS FLAMID
ARETCO   DS    A   ADDRESS RETCO
AREST    DS    0F
ARLAST   DS    A   ADDRESS LASTPAR
ARMODE   DS    A   ADDRESS MODE
ARLINK   DS    A   ADDRESS DD-NAME
ARSTATIS DS    A   ADDRESS STATIS
*
*  PARAMETERS FOR FLMOPD
*
         ORG   ARMODE
ARNLEN   DS    A   ADDRESS NAMELEN
ARNAME   DS    A   ADDRESS FILENAME
ARDSORG  DS    A   ADDRESS DSO&G
ARECFORM DS    A   ADDRESS REFORM
```

```
ARMAXSIZ DS    A   ADDRESS MAXSIZE
ARECDELI DS    A   ADDRESS RECDELIM
ARKYDESD DS    A   ADDRESS KEYDESC
ARBLKSIZ DS    A   ADDRESS BLKSIZE
ARCLOSDI DS    A   ADDRESS CLOSDISP
ARDEVICE DS    A   ADDRESS DEVICE
*
*  PARAMETERS FOR FLMOPF
*
         ORG   AREST
ARVERSIO DS    A   ADDRESS VERSION
ARCODE   DS    A   ADDRESS CODE
ARCOMPMO DS    A   ADDRESS COMPMODE
ARMAXBUF DS    A   ADDRESS MAXBUFFER
ARHEADER DS    A   ADDRESS HEADER
ARMAXREC DS    A   ADDRESS MAXREC
ARKYDESF DS    A   ADDRESS KEYDESC
ARBLKMOD DS    A   ADDRESS BLKMODE
AREXK20  DS    A   ADDRESS EXK20
AREXD20  DS    A   ADDRESS EXD20
*
*  PARAMETERS FOR FLMCLS
*
         ORG   AREST
ARCPUTIM DS    A   ADDRESS CPUTIME
ARECORDS DS    A   ADDRESS RECORDS
ARBYTES  DS    A   ADDRESS BYTES
ARBYTOFL DS    A   ADDRESS BYTEOFL
ARCMPREC DS    A   ADDRESS CMPRECS
ARCMPBYT DS    A   ADDRESS CMPBYTES
ARCBYOFL DS    A   ADDRESS CBYTEOFL
*
*  PARAMETERS FOR FLMGET, FLMLOC AND FLMPUT
*
         ORG   AREST
ARECLEN  DS    A   ADDRESS RECLEN
ARECPTR  DS    A   ADDRESS RECORD (RECPTR WITH LOCATE)
ARBUFLEN DS    A   ADDRESS BUFLEN
*
```

```
*  PARAMETERS FOR FLMPOS
*
         ORG   AREST
ARPOS    DS    A   ADDRESS POSITION
*
*  PARAMETERS FOR FLMGHD AND FLMPHD
*
         ORG  AREST
ARHNAML  DS    A    ADDRESS NAMLENE
ARHNAME  DS    A    ADDRESS FILENAME
ARHFCBT  DS    A    ADDRESS FILE FORMAT
ARHRECF  DS    A    ADDRESS RECORD FORMAT
ARHRECS  DS    A    ADDRESS RECORD LENGTH
ARHRECD  DS    A    ADDRESS RECDELIM
ARHKEYD  DS    A    ADDRESS KEYDESC
ARHBLKS  DS    A    ADDRESS BLOCK LENGTH
ARHPRCTR DS    A    ADDRESS PRINTER CONTROL CHARACTER
ARHSATTL DS    A    ADDRESS ATTRIBUTE LENGTH
ARHSATTR DS    A    ADDRESS SYSTEMSPEC. ATTRIBUTES
ARHSYST  DS    A    ADDRESS OPERATING SYSTEM
         ORG
*
*  PARAMETERS VALUES FOR FLAMREC
*
RETCO    DS    F    RETURN CODE
FLAMID   DS    F    FLAMFILE-ID
LASTPAR  DS    F    END OF PARAMETERS INPUT
OPENMODE DS    F    OPENMODE
POSITION DS    F    RELATIVE POSITION
ABSPOS   DS    F    ABSOLUTE POSITION
*
NAMELEN  DS    F    LENGTH OF FILE NAME FLAMFILE
FILENAME DS    CL54 FILE NAME OF FLAMFILE
DSORG    DS    F    DSORG
RECFORM  DS    F    RECFORM
MAXSIZE  DS    F    MAXSIZE
RECDELIM DS    XL4  RECDELIM
KEYSIZE  DS    F    LENGTH OF ALL PARTIAL KEYS
BLKSIZE  DS    F    BLKSIZE
```

```
CLOSDISP DS   F    CLOSDISP
DEVICE   DS   F    DEVICE
*
VERSION  DS   F    FLAM VERSION
CODE     DS   F    FLAMCODE
COMPMODE DS   F    COMPMODE
MAXBUFF  DS   F    MAXBUFFER
HEADER   DS   F    HEADER
MAXREC   DS   F    MAXRECORDS
BLKMODE  DS   F    BLKMODE
EXK20    DS   CL8  EXK20
EXD20    DS   CL8  EXD20
*
CPUTIME  DS   F    CPU TIME IN MILLI SECONDS
ELATIME  DS   F    ELAPSED TIME IN MILLI SECONDS
RECORDS  DS   F    NUMBER OF ORIGINAL RECORDS
BYTES    DS   F    NUMBER OF ORIGINAL BYTES
BYTEOFL  DS   F    OVERFLOW COUNTER FOR
*                  ORIGINAL BYTES
CMPRECS  DS   F    NUMBER OF COMPRESSED RECORDS
CMPBYTES DS   F    NUMBER OF COMPRESSED BYTES
CBYTEOFL DS   F    OVERFLOW COUNTER FOR
*                  COMPRESSED BYTES
FSATTRL  DS   F    LENGTH OF SYSTEM
*                  SPECIFIC ATTRIBUTES
FSYSATTR DS   0X   SYSTEM SPECIFIC ATTRIBUTES
*
*  KEY DESCRIPTION
*
KEYDESC  DS    0F
KEYFLAGS DS    F
KEYPARTS DS    F   NUMBER OF KEY PARTS
KEYPOS1  DS    F   FIRST BYTE OF FIRST PART
KEYLEN1  DS    F   LENGTH OF FIRST PART
KEYTYPE1 DS    F   DATA TYPE OF FIRST PART
KEYPOS2  DS    F
KEYLEN2  DS    F
KEYTYPE2 DS    F
KEYPOS3  DS    F
```

```
KEYLEN3  DS    F
KEYTYPE3 DS    F
KEYPOS4  DS    F
KEYLEN4  DS    F
KEYTYPE4 DS    F
KEYPOS5  DS    F
KEYLEN5  DS    F
KEYTYPE5 DS    F
KEYPOS6  DS    F
KEYLEN6  DS    F
KEYTYPE6 DS    F
KEYPOS7  DS    F
KEYLEN7  DS    F
KEYTYPE7 DS    F
KEYPOS8  DS    F    FIRST BYTE OF LAST PART
KEYLEN8  DS    F    LENGTH OF LAST PART
KEYTYPE8 DS    F    DATA TYPE OF LAST PART
*
RECLEN   DS    F
RECPTR   DS    A
*
*  SAVEAREA
*
SAVEAREA DS    18F
         .
         .
         .
         END
```

3.3.1 Function FLMOPN

The function FLMOPN must be called first. The compressed file is assigned to the program and the processing mode is defined.

Parameters:

1 ←	**FLAMID**	**F**	Identification. Must be specified in all following calls without any modifications.
2 ←	**RETCO**	**F**	Return code (see ch. 8)
	=	**0**	No error
	=	**-1**	Error during memory request
	=	**10**	File is not a FLAMFILE
	=	**11**	FLAMFILE format error
	=	**12**	Record length error
	=	**13**	File length error
	=	**14**	Checksum error
	=	**20**	Invalid OPENMODE
	=	**21**	Invalid size of matrix buffer
	=	**22**	Invalid compression mode
	=	**23**	Invalid code in FLAMFILE
	=	**24**	Invalid BLOCKMODE
	=	**25**	Invalid record length
	=	**30**	FLAMFILE is empty
	=	**37**	Invalid key position (unequal 1)
	=	**40**	Module or table cannot be loaded
	=	**41**	Module cannot be called
	=	**42**	Module cannot be unloaded
	=	**43-49**	Abort by user exit
	=	**52**	Invalid duplicate keys in the FLAMFILE
	=	**57**	Invalid partially compressed data length
	=	**x'F00000XX'**	FLAM error code from FLAMFIO for FLAMFILE
		x' 1F' = 31	FLAMFILE no assigned
		x' 20' = 32	Invalid OPENMODE
		x' 21' = 33	Invalid file type
		x' 22` = 34	Invalid record format
		x' 23' = 35	Invalid record length
		x' 24' = 36	Invalid block length
		x' 26' = 38	Invalid key length
		x' 27' = 39	Invalid file name
		x' 28' = 40	I/O-module not loaded (e.g. missing STEPLIB in JCL)
	=	**x'FFXXXXXX'**	DMS error code from FLAMFIO for FLAMFILE
3 →	**LASTPAR**	**F**	End of parameters for OPEN
	=	**0**	No further parameters
	=	**otherwise**	Additional function call with FLMOPD or FLMOPF will follow
4 →	**OPENMODE**	**F**	The OPEN mode controls the operation mode
	=	**0**	INPUT = read FLAMFILE (DECOMPRESSION)
	=	**1**	OUTPUT = write FLAMFILE (COMPRESSION)

	=	**2**	INOUT (with key and sequential read and update) **Note:** not allowed in MODE=ADC/NDC
5 →	**DDNAME**	**CL8**	Symbolic file name padded with blanks
6 →	**STATIS**	**F**	Switch statistics on or not
	=	**0**	No statistics
	=	**1**	Collect statistic data and transfer to user with FLMCLS

3.3.2 Function FLMOPD

The function FLMOPD describes special file attributes of the FLAMFILE. If FLMOPD is used, this function must be called as the second FLAM function after FLMOPN. Otherwise the default values described in the following are used. Generated parameters are not provided.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid identification, invalid call (e.g. LASTPAR=0 for FLMOPN)
	=	**otherwise**	Further return codes, see chapter 8
3 →	**LASTPAR**	**F**	End of parameters for OPEN
	=	**0**	No further parameters
	=	**otherwise**	Additional function call with FLMOPF follows
4 ↔	**NAMELEN**	**F**	→ Length of file name area ← Length of file name see note *)
5 ↔	**FILENAME**	**CL*n***	File name of FLAMFILE. The used file name is returned.
6 ↔	**DSORG**	**F**	File format of FLAMFILE
	=	**0; 8; 16**	PS; ESDS
	=	**1; 9; 17**	IS; KSDS
	=	**2; 10**	; RRDS
	=	**3; 11**	; LDS
7 ↔	**RECFORM**	**F**	Record format of FLAMFILE
	=	**0; 8; 16**	V; VB; VBS
	=	**1; 9; 17**	F; FB; FBS
	=	**2; 10; 18**	U
8 ↔	**MAXSIZE**	**F**	Maximum record length of FLAMFILE. Valid values: 80 - 32760. For CX7 a maximum record length of 4096

applies for the FLAMFILE (512 = DEFAULT)

9 ↔ RECDELIM XL*n* Record delimiter (Currently not supported.)

10 → KEYDESC STRUCT Key description for original records (the address of the structure must be passed)

↔ Key description

KEYFLAGS F Option
= 0 No duplicate key (DEFAULT)
= 1 Duplicate keys allowed
KEYPARTS F Number of key parts
= 0 - 8 (0 = no keys)
KEYPOS1 F Byte position of first key part
= 1-32759
KEYLEN1 F Length of first key part
= 1 - 255

KEYTYPE1 F Data type of first key part
= 0 Printable characters
= 1 Binary values

.
.
.

KEYPOS8 F Byte position of eighth key part
= 0 - 32759
KEYLEN8 F Length of eighth key part
= 1 - 255
KEYTYPE8 F Data type of eighth key part
= 0 Printable characters
= 1 Binary values

11 ↔ BLKSIZE F Block size
= 0 unblocked
= 80 - 32760

12 ↔ CLOSDISP F Mode of CLOSE processing
= 0 REWIND
= 1 UNLOAD
= 2 LEAVE

13 ↔ DEVICE F Device type
= 0; 8; 16 Disc or unknown
= 1; 9; 17 Tape
= 2; 10; 18 Floppy disc
= 3; 11; 19 Streamer
= 7; 15; 23 User

FLAM computes an optimum key length from the key description of the original file. In the case of binary compressed data, this key length is 1 byte longer than the sum of the original keys, with printable compressed data syntax (MODE=CX7), 2 bytes are added. The key position is al-

ways 1. If the KSDS-FLAMFILE is created by means of IDCAMS, the specifications mentioned above should be taken into account. A too short key length leads to a loss of performance during further processing.

***note:** Sometimes the actual used file has a different name than given (found by jcl DD-statement) and is longer than expected. If the buffer FILENAME (in length of NAMELEN) is too short to fit the full file name, it will be truncated.
It is recommended to define FILENAME in length of 54 bytes, to fill up the file name with blanks, and to set NAMELEN to 54. On return the actual length is stored, FILENAME is padded with blanks.

3.3.3 Function FLMOPF

The function FLMOPF defines the attributes of the compressed file. FLMOPF can be called as second function after FLMOPN or as third function after FLMOPD.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= 0		No error
	= -1		Invalid identification, invalid call (e.g. LASTPAR=0 for FLMOPN or FLMOPD)
	= 40		Load of user exit failed
	= 43-49		Abort by user exit
	= otherwise		Further return codes, see chapter 8
3 ↔	**VERSION**	**F**	FLAMFILE version
	= 100		Version 1 / 6020
	= 101		Version 1 / 6035
	= 200		Version 2
	= 300		Version 3
	= 400		Version 4
4 ←	**FLAMCODE**	**F**	Character code of FLAMFILE
	= 0		EBCDIC
	= 1		ASCII
5 ↔	**COMPMODE**	**F**	Compression mode
	= 0		CX8
	= 1		CX7
	= 2		VR8
	= 3		ADC
	= 5		NDC
6 ↔	**MAXBUFF**	**F**	Size of matrix buffer in BYTES. Any positive value is

allowed. The actual used value is returned.

7 ↔	**HEADER**	**F**	FILEHEADER to be created or available.
	= 0		No file header to be created or available.
	= 1		File header to be created or available.
8 →	**MAXREC**	**F**	Maximum record number in matrix
	= 1- 255		in mode CX7/CX8/VR8
	= 1- 4095		in mode ADC
9 ↔	**KEYDESC STRUCT**		Key description for original records (the address of the data structure must be passed).
	KEYFLAGS	**F**	Option
	= 0		No duplicate keys
	= 1		Duplicate keys allowed
	KEYPARTS	**F**	Number of key parts
	= 0 - 8		(0 = no key)
	KEYPOS1	**F**	Byte position of first part of key
	= 1 - 32759		
	KEYLEN1	**F**	Length of first part of key
	= 1 - 255		
	KEYTYPE1	**F**	Data type of first part of key
	= 0		Printable characters
	= 1		Binary values
	.		
	.		
	.		
	KEYPOS8	**F**	Byte position last part of key
	= 1 - 32759		
	KEYLEN8	**F**	Length of last part of key
	= 1 - 255		
	KEYTYPE8	**F**	Data type of last part of key
	= 0		Printable characters
	= 1		Binary values (DEFAULT)
10 →	**BLKMODE**	**F**	Blocked or unblocked output for sequential compressed files.
	= 0		Unblocked (one compressed record contains only data from one matrix)
	= 1		Blocked (one compressed record can contain data from many matrices)
11 →	**EXK20**	**CL8**	Blank or name of user exit for output of compressed file
12 →	**EXD20**	**CL8**	Blank or name of user exit of input of compressed file
←			If the exit *STREAM is activated automatically during decompression.

3.3.4 Function FLMCLS

With function FLMCLS the access to the record level interface is terminated.

When compression is closed, the last matrix is compressed now, the compressed data is written to the FLAMFILE. Additional information (byte-, record counter, MACs) are stored as an 'ending record', if required (SECUREINFO=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.

If specified with FLMOPN (STATIS=1), statistical data is transferred to the caller.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= **0**		No error
	= **-1**		Invalid identification (e.g. FLMCLS already done ?)
	= **43-49**		Abort by user exit
	= **x'FFXXXXXX'**		DMS- error code

The following parameters are only used if the statistics are switched on in FLMOPN.

3 ←	**CPUTIME**	**F**	CPU in milliseconds
4 ←	**RECORDS**	**F**	Number of original records
5 ←	**BYTES**	**F**	Number of original bytes
6 ←	**BYTEOFL**	**F**	Overflow counter for original bytes
7 ←	**CMPRECS**	**F**	Number of compressed records
8 ←	**CMPBYTES**	**F**	Number of compressed bytes
9 ←	**CMPBYOFL**	**F**	Overflow counter for compressed bytes

With extremely big compressed files (greater 4 gigabytes) one word byte counters are not sufficient. For this purpose the overflow counters are provided. They allow to extend the counter to a double word:

```
01 BYTEFIELD.
    05 BYTEOFL            PIC 9(8) COMP SYNC.
    05 BYTES              PIC 9(8) COMP SYNC.
01 BYTECNT REDEFINES BYTEFIELD PIC S9(18) COMP SYNC.
```

3.3.5 Function FLMDEL

With function FLMDEL it is possible to delete the last read original record from an index sequential FLAMFILE.

Parameters:

1 →	**FLAMID**	**F**		Identification
2 ←	**RETCO**	**F**		Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function invalid
	=	**5**		No active record found
	=	**43-49**		Abort by user exit
	=	**x'FFXXXXXX'**		DMS error code see FLMOPN

3.3.6 Function FLMEME

The function FLMEME finishes the current data as member of a Group-FLAMFILE (end member). During compression, the content of the matrix is compressed and written immediately, enlarged with some security information, if required (SECUREINFO=YES); during decompression, the next matrix is decompressed.

On AES encryption, a member MAC is created and written. The MAC is returned.

On AES decryption, the member MAC is returned.

Statistical values are returned.

To end only a matrix and not the entire member, function FLMFLU is provided.

Parameter:

1 →	**FLAMID**	**F**		Identification
2 ←	**RETCO**	**F**		Return code
	=	**0**		No error
	=	**-1**		Invalid identification
	=	**43-49**		Abort by user exit
	=	**x'FFXXXXXX'**		DMS error code
3 ←	**CPUTIME**	**F**		CPU in milliseconds
4 ←	**RECORDS**	**F**		Number of original records
5 ←	**BYTES**	**F**		Number of original bytes
6 ←	**BYTEOFL**	**F**		Overflow counter for original bytes
7 ←	**CMPRECS**	**F**		Number of compressed records
8 ←	**CMPBYTES**	**F**		Number of compressed bytes
9 ←	**CMPBYOFL**	**F**		Overflow counter for compressed bytes
10 ←	**MEMBRMAC**	**XL8**		Member-Mac

With extremely big compressed files (greater 4 gigabytes) one word byte counters are not sufficient. For this purpose the overflow counters are provided. They allow to extend the counter to a double word:

```
01 BYTEFIELD.
    05 BYTEOFL             PIC 9(8) COMP SYNC.
    05 BYTES               PIC 9(8) COMP SYNC.
01 BYTECNT REDEFINES BYTEFIELD PIC S9(18) COMP SYNC.
```

3.3.7 Function FLMFKY

The function FLMFKY (Find Key) can be used to search in an index sequential FLAMFILE for a record of the original file, whose key is equal to or greater than a specified key value. The specified value can be generic, i.e. not all of the positions of the key value have to be specified uniquely. The record found is the next record to be processed.

If FLMFKY does not find a record, the old position is retained.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid identification or function
	=	**5**	Key not found
	=	**otherwise**	See function FLMGET
3 →	**KEYLEN**	**F**	Key length This contains the number of significant bytes in the specified key value. It can be less than the key length. In this case, only the length passed here is taken into account in the logical relation specified in the argument checkmod.
4 →	**RECORD**		Record buffer with search key
5 →	**CHECKMOD**		Type of relation
	=	**0**	equal to
	=	**1**	greater than or equal to
	=	**2**	greater than

3.3.8 Function FLMFLU

The function FLMFLU finishes the current FLAM matrix. If specified with FLMOPN (STATIS=1), statistical data is transferred to the caller. During compression, the content of the matrix is compressed and written immediately; during decompression, the next matrix is decompressed.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code

=	**0**	No error
=	**-1**	Invalid identification
=	**43-49**	Abort by user exit
=	**x'FFXXXXXX'**	DMS error code

The following parameters are only used if the statistics are switched on.

3 ←	**CPUTIME**	**F**	CPU in milliseconds in foreign processes
4 ←	**RECORDS**	**F**	Number of original records
5 ←	**BYTES**	**F**	Number of original bytes
6 ←	**BYTEOFL**	**F**	Overflow counter for original bytes
7 ←	**CMPRECS**	**F**	Number of compressed records
8 ←	**CMPBYTES**	**F**	Number of compressed bytes
9 ←	**CMPBYOFL**	**F**	Overflow counter for compressed bytes

With extremely big compressed files (greater 4 gigabytes) one word byte counters are not sufficient. For this purpose the overflow counters are provided. They allow to extend the counter to a double word:

```
01 BYTEFIELD.
    05 BYTEOFL              PIC 9(8) COMP SYNC.
    05 BYTES                PIC 9(8) COMP SYNC.
01 BYTECNT REDEFINES BYTEFIELD PIC S9(18) COMP SYNC.
```

3.3.9 Function FLMFRN

The function FLMFRN (Find Record Number) positions the record pointer to a record with a specified number in an index sequential FLAMFILE. This number corresponds to the record number of the sequential or relative original file. The record is the next record to be processed. Specifying checkmod=1 or 2 allows gaps and empty records to be skipped.

If FLMFRN does not find a record, the old position is retained.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid identification or function
	=	**5**	Invalid position
	=	**otherwise**	See function FLMGET
3 ↔	**RECNO**	**F**	Record number
	=	**1**	File start. With checkmod=1,2 the true record number is returned
4 →	**CHECKMOD**		Type of relation
	=	**0**	Record with specified number
	=	**1**	Record with specified number, skip gaps and empty records
	=	**2**	Record with next number, skip gaps and empty records

3.3.10 Function FLMGET

With function FLMGET the next original record is read in sequential order. It is possible to position to a certain record in the compressed file using FLMGKY or FLMPOS and then to continue with sequential reading. Data is transferred from the record buffer to the calling program (MOVE mode).

Parameters:

1 →	**FLAMID**	**F**		Identification
2 ←	**RETCO**	**F**		Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**1**		Record was truncated because original record was larger than BUFLEN
	=	**2**		END-OF-FILE found
	=	**3**		Gap in relative file found
	=	**6**		New file starts, eventually the new file header can be read
	=	**7**		Password required. Pass it via FLMPWD
	=	**11**		FLAMFILE format error
	=	**12**		Record length error
	=	**13**		File length error
	=	**14**		Checksum error
	=	**29**		Illegal password
	=	**43-49**		Abort by user exit
	=	**52**		Too many or invalid duplicate keys
	=	**x'FFXXXXXX'**		Data management error code
	=	**otherwise**		See chapter 8.4
3 ←	**RECLEN**	**F**		Record length in bytes of the passed record
4 ←	**RECORD**	**XL*n***		Original record (data)
5 →	**BUFLEN**	**F**		Length of available record buffer in bytes

Note:

With return codes 2 , 6 and 7 no record is passed.

With return code 3 a record of length 0 is passed.

3.3.11 Function FLMGHD

The function FLMGHD (Get File Header) is only allowed during decompression. The file header describes the file format of the original records. It is possible to request the file header information with function FLMGHD at any time between FLAM-OPEN (FLMOPN, FLMOPD, FLMOPF) and FLAM-CLOSE (FLMCLS). If there are several file headers in the FLAMFILE (see FLMPHD), the last file header recognized by FLAM is transferred with FLMGHD. The first file header is usually available immediately after FLAM-OPEN (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:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= 0		No error
	= -1		Invalid identification or function
3 ↔	**NAMLEN**	**F**	Length of file name or of area
	= 0		File name not known
4 ↔	**FILENAME**	**CL*n***	File name of original file
5 ↔	**FCBTYPE**	**F**	File format
	= 0		sequential
	= 1		index sequential
	= 2		relative
	= 3		random access
	= 5		library
	= 6		physically
6 ←	**RECFORM**	**F**	Record format
	= 0; 8; 16 ...		VARIABEL (V) 8 = VARBLK 16 = SPNBLK
	= 1; 9; 17 ...		FIX (F) 9 = FIXBLK
	= 2; 10; 18 ...		UNDEFINED (U)
	= 3; 11; 19 ...		STREAM (S) 11 = text delimiter 19 = length fields
7 ←	**RECSIZE**	**F**	Record length
	= 0 - 32760		
	RECFORM = V:		Maximum record length or 0
	RECFORM = F:		Record length
	RECFORM = U:		Maximum record length or 0
	RECFORM = S:		Length of text delimiter or of length field
8 ←	**RECDELIM**	**XL*n***	Record delimiter
9 ←	**KEYDESC**	**STRUCT**	Key description
	KEYFLAGS	**F**	Options
	= 0		No duplicate keys

	=	**1**	Duplicate keys allowed
	KEYPARTS	**F**	Number of key parts
	=	**0 - 8**	0 = No keys available
	KEYPOS1	**F**	First byte of first key part
	=	**1 - 32759**	Value < = Record length
	KEYLEN1	**F**	Length of first key part
	=	**1 - 255**	
	KEYTYP1	**F**	Data type of first key part
	=	**0**	Printable characters
	=	**1**	Binary values
	.		
	.		
	KEYPOS8	**F**	First byte of eighth key part
	=	**1 - 32759**	Value < = record length
	KEYLEN8	**F**	Length of eighth key part
	=	**1 - 255**	
	KEYTYP8	**F**	Data type of eighth key part
	=	**0**	Printable characters
	=	**1**	Binary values
10 ←	**BLKSIZE**	**F**	Block length
	=	**0**	unblocked
	=	**1- 32760**	
11 ←	**PRCTRL**	**F**	Printer control characters
	=	**0**	none
	=	**1**	ASA control characters
	=	**2**	Machine specific control characters
12 ←	**SYSTEM**	**XL2**	Operating system where FLAMFILE was created.
	=	**x'0000'**	unknown
	=	**x'0080'**	MS-DOS
	=	**x'0101'**	IBM OS-MVS MVS/XA MVS/ESA
	=	**x'0102'**	IBM DOS/VSE VSE/SP
	=	**x'0103'**	IBM VM/SP VM/XA
	=	**x'0104'**	IBM DPPX/8100
	=	**x'0105'**	IBM DPPX/370
	=	**x'02XX'**	UNISYS
	=	**x'0301'**	DEC VMS
	=	**x'0302'**	DEC ULTRIX
	=	**x'0401'**	SIEMENS BS2000
	=	**x'0402'**	SIEMENS SINIX
	=	**x'0403'**	SIEMENS SYSTEM V
	=	**x'0501'**	NIXDORF 886X
	=	**x'0502'**	NIXDORF TARGON
	=	**x'06XX'**	WANG
	=	**x'07XX'**	PHILLIPS
	=	**x'08XX'**	OLIVETTI
	=	**x'11XX'**	INTEL 80286
	=	**x'12XX'**	INTEL 80386
	=	**x'13XX'**	INTEL 80486

3.3.12 Function FLMGKY

The user can obtain an original record from an index sequential FLAMFILE via a key by using the function FLMGKY.

The search key must be placed at the key position within the record area.

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**1**		Record was truncated because original record exceeds BUFLEN
	=	**5**		Key not found
	=	**otherwise**		See function FLMGET or chapter 8.4
3 ←	**RECLEN**		**F**	Length of passed record in bytes
4 ←	**RECORD**		**XL*n***	Original record (Data with key)
5 →	**BUFLEN**		**F**	Length of available record buffer in bytes

3.3.13 Function FLMGRN

The function FLMGRN (Get Record Number) reads the original record of a sequential or relative file from an index sequential FLAMFILE as specified by the record number.

If FLMGRN does not find a valid record, the new position moved to is the next record or the end of the file.

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**1**		Record was truncated because original record was larger than BUFLEN
	=	**2**		END-OF-FILE found
	=	**3**		Gap in relative file found
	=	**5**		Invalid record number (0 or negative)
	=	**6**		New file starts, eventually the new file header can be read
	=	**otherwise**		See function FLMGET or chapter 8.4
3 ←	**RECLEN**		**F**	Record length in bytes of the passed record
4 ←	**RECORD**		**XL*n***	Original record (data)
5 →	**BUFLEN**		**F**	Length of available record buffer in bytes
6 →	**RECNO**		**F**	Record number
	=	**1**		File start

With return codes 2, 6 and 7 no record is passed.

With return code 3 a record of length =0 is passed.

3.3.14 Function FLMGTR

With function FLMGTR (Get reverse) the previous original record is read in sequential order. It is possible to position to a certain record in the compressed file using FLMGKY or FLMPOS and then to read backward sequentially. Data is transferred from the record buffer to the calling program (MOVE mode).

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**1**		Record was truncated because original record was larger than BUFLEN
	=	**2**		END-OF-FILE found
	=	**3**		Gap in relative file found
	=	**6**		New file starts, eventually the new file header can be read
	=	**otherwise**		See function FLMGET or chapter 8.4
3 ←	**RECLEN**		**F**	Record length in bytes of the passed record
4 ←	**RECORD**		**XL*n***	Original record (data)
5 →	**BUFLEN**		**F**	Length of available record buffer in bytes

With return codes 2 and 6 no record is passed.

With return code 3 a record of length 0 is passed.

3.3.15 Function FLMGUH

The function FLMGUH (Get User Header) reads the user data from the file header of the FLAMFILE.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= 0		No error
	= -1		Invalid identification or function
3 ↔	**UATTRLEN**	**F**	Length of user data in bytes or length of area
	= 0		No data
	= 1-3500		With 8-bit compressed data (CX8, VR8)
	= 1-1750		With 7-bit compressed data (CX7)
4 ←	**UATTR**	**XL*n***	User data

The user data is reproduced exactly as it is written, i.e. converting the code of a file transfer has no effect here.

3.3.16 Function FLMIKY

The function FLMIKY allows to insert records with a key to an index sequential FLAMFILE (VSAM-KSDS).

Parameter:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**5**		Key exists already
	=	**15**		Original record is larger than 32763 bytes
	=	**16**		Original record is larger than matrix - 4
	=	**43-49**		Abort by user exit
	=	**52**		Too many or invalid duplicate keys
	=	**x'FFXXXXXX'**		DMS error code
3 →	**RECLEN**		**F**	Record length (data length) in bytes without record length field
4 →	**RECORD**		**XL*n***	Original record (data with key)

3.3.17 Function FLMLCR

The function FLMLCR is equivalent to FLMGTR (Get reverse). The data will not be transmitted but only a pointer on the record is provided (locate Mode).

Parameter:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**2**		Beginning of file found
	=	**otherwise**		see FLMGET or chapter 8.4
3 ←	**RECLEN**		**F**	Record length in bytes of the passed record
4 ←	**RECPTR**		**A**	Record address (data address)

Note:

With return codes 2 and 6 no record address is passed.

With return code 3 the length 0 is passed.

3.3.18 Function FLMLOC

The function FLMLOC is equivalent to FLMGET. But data is not transferred. Instead a pointer to the record is set (LOCATE mode).

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**2**		END-OF-FILE found
	=	**3**		gap in relative file found
	=	**6**		New file begins, eventually a new file header can be read.
	=	**otherwise**		See function FLMGET or chapter 8.4
3 ←	**RECLEN**		**F**	record length in bytes of record passed
4 ←	**RECPTR**		**A**	record address (data address)

Note:

With return codes 2 and 6 no record address is passed.

With return code 3 the length 0 is passed.

3.3.19 Function FLMPHD

The function FLMPHD (Put File Header) is only allowed during compression. The file header describes the file format for the original records following. If multiple files are compressed into one FLAMFILE, each original file may have its own file header created with FLMPHD. FLAM returns the header information during decompression if required (FLMGHD). The function FLMPHD is only allowed if HEADER=1 was specified with FLMOPF.

Using SECUREINFO=YES, function FLMPHD is mandatory!

The FLAM utility uses these data during decompression to build the output file (FLAMOUT).

Note: The parameter in FLMPHD control also 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:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= 0		No error
	= -1		Invalid identification or function
3 →	**NAMLEN**	**F**	Length of file name
	= 0		Don't use file name
4 →	**FILENAME**	**CL*n***	File name of original file
5 →	**DSORG**	**F**	File format
	= 0; 8; 16 ...		sequential
	= 1; 9; 17 ...		index sequential
	= 2; 10; 18 ...		relative
	= 3; 11; 19 ...		random access
	= 5; 13; 21 ...		library
	= 6; 14; 22 ...		physical
6 →	**RECFORM**	**F**	Record format
	= 0; 8; 16 ...		VARIABEL (V) 8 = VARBLK 16 = SPNBLK
	= 1; 9; 17 ...		= FIX (F) 9 = FIXBLK 17=FBS
	= 2; 10; 18 ...		= UNDEFINED (U)
	= 3; 11; 19 ...		= STREAM (S) 11 = text delimiter 19 = length fields
7 →	**RECSIZE**	**F**	Record length
	= 0 - 32760		
	RECFORM = V:		Maximum record length or 0
	RECFORM = F:		Record length

	RECFORM = U:		Maximum record length or 0
	RECFORM = S:		Length of text delimiter or of text length field
8 →	**RECDELIM**	**XL*n***	Record delimiter
9 →	**KEYDESC**	**STRUCT**	Key description
	KEYFLAGS	**F**	Options
	= **0**		No duplicate keys
	= **1**		Duplicate keys allowed
	KEYPARTS	**F**	Number of key parts
	= **0 - 8**		0 = No key available
	KEYPOS1	**F**	First byte of first key part
	= **1 - 32759**		Value < = record length
	KEYLEN1	**F**	Length of first key part
	= **1 - 255**		
	KEYTYP1	**F**	Data type of first key part
	= **0**		Printable characters
	= **1**		Binary values
	.		
	.		
	KEYPOS8	**F**	First byte of last key part
	= **1 - 32759**		Value < = record length
	KEYLEN8	**F**	Length of last key part
	= **1 - 255**		
	KEYTYP8	**F**	Data type of last key part
	= **0**		Printable characters
	= **1**		Binary values
10 →	**BLKSIZE**	**F**	Block length
	= **0**		unblocked
	= **1 - 32760**		
11 →	**PRCTRL**	**F**	Printer control characters
	= **0**		none
	= **1**		ASA control characters
	= **2**		Machine specific control characters
12 →	**SYSTEM**	**XL2**	Operating system
	= **x'0000'**		unknown
	= **x'0101'**		IBM MVS/z/OS
	= **x'0102'**		IBM VSE
	= **x'0103'**		IBM VM
13 →	**LASTPAR**	**F**	End of parameters for file header
	= **0**		No more parameters.
	otherwise		A user header is to be transferred with FLMPUH.

3.3.20 Function FLMPKY

The function FLMPKY allows to insert records into an index sequential FLAMFILE or to update records within such a file via a key.

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**5**		Key not allowed
	=	**15**		Original record greater than 32763 bytes
	=	**16**		Original record greater than matrix - 4
	=	**43**		Abort by user exit
	=	**52**		Too many or invalid duplicate keys
	=	**x'FFXXXXXX'**		DMS error code
3 →	**RECLEN**		**F**	Record length (data length) in bytes without record length field (exclusive length)
4 →	**RECORD**		**XL*n***	Original record (Data with key)

3.3.21 Function FLMPOS

FLMPOS allows to position the record pointer within compressed files.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid identification or function
	=	**5**	Invalid position
	=	**40 - 78**	see function FLMGET
	=	**x'FFXXXXXX'**	DMS error code
3 →	**POSITION**	**F**	Position
	= - MAXINT		File start (-2147483648 or X'80000000' or -99999999)
	= + MAXINT		File end (+2147483647 or X'7FFFFFFF' or + 99999999)
	= - N		N records backwards
	= + N		N records forwards
	= **- 9999 9998**		Back to the start of the current file or to the start of the previous file in a group file
	= **+ 9999 9998**		Beginning of the next file in a group file.

With OPEN=INPUT and INOUT or OUTIN the pointer can positioned anywhere, irrespective of whether the original file is index sequential or sequential.

With OPEN = OUTPUT it is possible to create gaps in relative files by advancing the write pointer by N records.

3.3.22 Function FLMPUH

The function FLMPUH (Put User Header) writes user data into the file header of the FLAMFILE.

Parameter:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
3 →	**UATTRLEN**		**F**	Length of file name or of area
	=	**1-3500**		With 8-bit compressed data
	=	**1-1750**		With 7-bit compressed data (Mode=CX7)
4 →	**USERATTR**		**XL*n***	User data as binary data string.

In CX7, this data is converted in such a way that the integrity of the FLAMFILE is not corrupted.

The user data itself remains in its original code during reading, even during code conversions using heterogeneous data exchange.

3.3.23 Function FLMPUT

With function FLMPUT one original record is transferred for compression.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= **0**		No error
	= **-1**		Invalid identification of function
	= **15**		Original record is larger than 32763 bytes
	= **16**		Original record is larger than matrix - 4
	= **43-49**		Abort by user exit
	= **x'FFXXXXXX'**		DMS error code (see FLMOPN)
3 →	**RECLEN**	**F**	Record length (Data length) in bytes without record length field (exclusive length)
4 →	**RECORD**	**XL*n***	Original record (Data)

3.3.24 Function FLMPWD

With function FLMPWD a password is given in. This function can only called up once.

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 in the FLAMFILE and can be obtained from function FLMQRY.

Parameters:

1 →	**FLAMID**	**F**	Identification
2 ←	**RETCO**	**F**	Return code
	= 0		No error
	= -1		Password function invalid, e.g. for MODE=CX8, VR8, CX7, renewed call up
3 →	**PWDLEN**	**F**	Password length in bytes (max. 64)
4 →	**PASSWORD**	**XL*n***	Password

3.3.25 Function FLMQRY

With function 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 other function calls the field RETCO 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.

Parameters:

1 → FLAMID		**F**	Identification
2 ← RETCO,INFCO		**2F**	Return code, Info code
	= 0,0		no error, Info code=0
			else: the parameter in error is returned in INFCO.
	= 91,param		unknown parameter
3 →	**PARAM1**	**F**	first parameter
4 →	**VALUE1**	**F**	first parameter value
.			
.			
.			
n →	**PARAMn**	**F**	last parameter
n+1→	**VALUEn**	**F**	last parameter value

Note: Multiple parameter can be set in one call. It is necessary to mark the end of the parameter list! Most compilers do it automatically, but in Assembler the last parameter address has to be flagged: A(X'80000000' +VALUEn).

Following parameters are available:

Description	Parameter	Value
Splitmode	**1**	**0 / 1 / 2 none / serial / parallel**
Splitnumber	**2**	**2 - 4**

Cryptomode	**2001**	**0 / 1 / 2** **none / FLAM / AES**
Secureinfo	**2002**	**0 / 1 / 2 / 3** **no / yes / ignore / member**

3.3.26 Function FLMSET

Function FLMSET sets parameter that open functions do not support.

It is called before FLMOPD and/or FLMOPF. A call at the wrong moment will be rejected with return code 90.

Note: In opposite to the other function calls the field RETCO 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.

Parameters:

1 → FLAMID		**F**	Identification
2 ← RETCO,INFCO		**2F**	Return code, Info code
	= 0,0		no error, Infocode=0
			else: the parameter in error is returned in INFCO.
	= 90,param		not allowed (e.g. SPLITMO after FLMOPD)
	= 91,param		unknown parameter
	= 92,param		unknown parameter value
3 → PARAM1		**F**	first parameter
4 → VALUE1		**F**	first parameter value
.			
.			
.			
n → PARAMn		**F**	last parameter
n+1→ VALUEn		**F**	last parameter value

Note: Multiple parameter can be obtained in one call. It is necessary to mark the end of the parameter list! Most compilers do it automatically, but in Assembler the last parameter address has to be flagged: A(X'80000000' +VALUEn).

Following parameters are to be set before FLMOPD:

Description	Parameter	Value
Splitmode	**1**	**0** / **1** / **2** none / serial / parallel
Splitnumber	**2**	**2 – 4**
Splitsize	**3**	**1 - 4095** in megabytes

Following parameters are to be set before FLMOPF:

Description	Parameter	Value
Cryptomode	**2001**	**0** / **1** / **2** none / FLAM / AES
Secureinfo	**2002**	**0** / **1** / **2** / **3** no / yes / ignore / member

3.3.27 Function FLMUPD

The function FLMUPD updates the original record last read from a VSAM-KSDS FLAMFILE.

Parameters:

1 →	**FLAMID**		**F**	Identification
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid identification or function
	=	**5**		No current record
	=	**15**		Original record is greater than 32764 bytes
	=	**16**		Original record is greater than matrix - 4
	=	**43**		Abort by user exit
	=	**x'FFXXXXXX'**		DMS error code
3 →	**RECLEN**		**F**	Record length (data length) in bytes without record length field
4 →	**RECORD**		**XL*n***	Original record (data)

3.4 User I/O interface

The user I/O interface can be used for the FLAM utility, for the subprogram FLAMUP and for the record level interface FLAMREC.

Under FLAM and FLAMUP it is possible to process the input file (FLAMIN), the output file (FLAMOUT) and the compressed file (FLAMFILE) with the user I/O interface. The user I/O interface is activated by specifying the parameters IDEVICE=USER, ODEVICE=USER and DEVICE=USER.

At the record level interface FLAMREC the user I/O interface is activated with the parameter DEVICE in function FLMOPD only for the compressed file (FLAMFILE).

The required functions are then provided by the user. The functions USROPN and USRCLS are mandatory. From the other functions only that functions must be provided, that are actually needed by the application.

The product FLAM contains an example source program for the user I/O interface written in COBOL and ASSEMBLER. In this example all functions are provided as dummies.

USROPN Open of file or of interface.

USRCLS Close of file or of interface.

USRGET Read one record and pass.

USRPUT Accept one record and write.

USRGKY Read one record with key and pass.

USRPOS Position in file.

USRPKY Accept one record and write with key.

USRDEL Delete the last read record.

3.4.1 Function USROPN

Open the interface for the file defined in the DD-name.

Parameters:

1 ↔	**WORKAREA**	**256F**	The working area is initialized with x00. This area is assigned exclusively to the file opened. It can be used as a memory or scratchpad between different function calls.
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid function
	=	**30**	Input file is empty
	=	**31**	Input file does not exist
	=	**32**	Invalid OPENMODE
	=	**33**	Invalid file type
	=	**34**	Invalid record format
	=	**35**	Invalid record length
	=	**36**	Invalid block length
	=	**37**	Invalid key position
	=	**38**	Invalid key length
	=	**39**	Invalid file name
	=	**x'0FXXXXXX'**	Other error codes
3 →	**OPENMODE**	**F**	The OPEN mode controls the operation mode.
	=	**0**	INPUT (read sequential) (File must exist)
	=	**1**	OUTPUT (write sequential) (New file is created or old file is overwritten)
	=	**2**	INOUT (with key, also read and write sequentially) (File must exist)
	=	**3**	OUTIN (with key, also read and write sequentially) (New file is created or old file is overwritten)
4 →	**DDNAME**	**CL8**	Symbolic file name
5 ↔	**DDSORG**	**F**	File format
	=	**0; 8; 16 ...**	sequential
	=	**1; 9; 17 ...**	index sequential
	=	**2; 10; 18 ...**	relative
	=	**3; 11; 19 ...**	random access
	=	**5; 13; 21 ...**	library
	=	**6; 14; 22 ...**	physical
6 ↔	**RECFORM**	**F**	Record format
	=	**0; 8; 16 ...**	VARIABEL (V) 8 = VARBLK 16 = SPNBLK
	=	**1; 9; 17 ...**	FIX (F) 9 = FIXBLK 17=FBS
	=	**2; 10; 18 ...**	UNDEFINED (U)
	=	**3; 11; 19 ...**	STREAM (S) 11 = text delimiter 19 = record length field

7 ↔	**RECSIZE**	**F**	Record length
	=	**0 - 32760**	
	RECFORM = V:		Maximum record length or 0
	RECFORM = F:		Record length
	RECFORM = U:		Maximum record length or 0
	RECFORM = S:		Length of text delimiter or of record length field
8 ↔	**BLKSIZE**	**F**	Block length
	=	**0**	unblocked
	=	**1 - 32760**	
9 ↔	**KEYDESC**	**STRUCT**	Key description
	KEYFLAGS	**F**	Options
	=	**0**	No duplicate keys
	=	**1**	Duplicate keys allowed
	KEYPARTS	**F**	Number of key parts
	=	**0 - 8**	0 = No key available
	KEYPOS1	**F**	First byte of first key part
	=	**1 - 32759**	Value record length
	KEYLEN1	**F**	Length of first key part
	=	**1 - 255**	
	KEYTYP1	**F**	Data type of first key part
	=	**0**	Printable characters
	=	**1**	Binary values
	.		
	.		
	.		
	KEYPOS8	**F**	First byte of last key part
	=	**1 - 32759**	Value record length
	KEYLEN8	**F**	Length of last key part
	=	**1 - 255**	
	KEYTYP8	**F**	Data type of last key part
	=	**0**	Printable characters
	=	**1**	Binary value
10 ↔	**DEVICE**	**F**	Device type
	=	**7; 15; 23 ...**	User devices
11 ↔	**RECDELIM**	**XL*n***	Record delimiter
12 ↔	**PADCHAR**	**XL1**	Padding character
13 ↔	**PRCTRL**	**F**	Printer control characters
	=	**0**	none
	=	**1**	ASA control characters
	=	**2**	Machine specific control characters
14 →	**CLOSDISP**	**F**	Method of CLOSE processing
	=	**0**	REWIND
	=	**1**	UNLOAD
	=	**2**	LEAVE

15 →	**ACCESS**		**F**	Access method
	=	**0**		logical (by record)
16 ↔	**NAMELEN**		**F**	Length of file name or of file name area
17 ↔	**FILENAME**		**CL*n***	File name

Note: In the current version only one key is supported.

3.4.2 Function USRCLS

The interface is closed for a given file.

Parameters:

1 ↔	**WORKAREA**		**256F**	Working storage area
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Invalid function
	=	**x'0FXXXXXX'**		Other error codes

3.4.3 Function USRGET

Head record sequentially and pass.

Parameters:

1 ↔	**WORKAREA**		**256F**	Working storage area
2 ←	**RETCO**		**F**	Return code
	=	**0**		No error
	=	**-1**		Function invalid
	=	**1**		Record was truncated
	=	**2**		END-OF-FILE found
	=	**3**		Gap in relative file found
	=	**x'0FXXXXXX'**		Other error codes
3 ←	**RECLEN**		**F**	Length of passed record in bytes
4 ←	**RECORD**		**XL*n***	Original record (Data)
5 →	**BUFLEN**		**F**	Length of available record buffer in bytes

3.4.4 Function USRPUT

Accept record and write sequentially.

Parameters:

1 ↔	**WORKAREA**	**256F**	Working storage area
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid function
	=	**1**	Record was truncated
	=	**4**	Record was filled with padding characters (PADCHAR)
	=	**x'0FXXXXXX'**	Other error codes
3 →	**RECLEN**	**F**	Length of passed record in bytes
4 →	**RECORD**	**XL*n***	Original record (Data)

3.4.5 Function USRGKY

Head record with specified key and pass. The key value is filled into the record at the key position defined in KEYDESC.

Parameters:

1 ↔	**WORKAREA**	**256F**	Working storage area
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid function
	=	**1**	Record was truncated
	=	**2**	END-OF-FILE found
	=	**5**	Key not found
	=	**x'0FXXXXXX'**	Other error codes
3 ←	**RECLEN**	**F**	Record length in bytes
4 ↔	**RECORD**	**XL*n***	Record with key / Record
5 →	**BUFLEN**	**F**	Length of available record buffer in bytes

3.4.6 Function USRPOS

Position record pointer in file.

Parameters:

1 ↔	**WORKAREA**	**256F**	Working storage area
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid function
	=	**5**	Invalid position
	=	**x'0FXXXXXX'**	Other error codes
3 →	**POSITION**	**F**	Relative position
	=	**0**	No positioning
	=	**- MAXINT**	File start (-2147483648 or x'80000000')
	=	**+ MAXINT**	File end (+2147483647 or x'7FFFFFFF')
	=	**- n**	n records backwards
	=	**+ n**	n records forwards

Note: This function allows to create gaps within a relative file by positioning the record pointer forward.

3.4.7 Function USRPKY

Write record with specified key.

Parameters:

1 ↔	**WORKAREA**	**256F**	Working storage area
2 ←	**RETCO**	**F**	Return code
	=	**0**	No error
	=	**-1**	Invalid function
	=	**1**	Record was truncated
	=	**4**	Record was filled with padding characters (PADCHAR).
	=	**5**	Key is invalid
	=	**x'0FXXXXXX'**	Other error code
3 →	**RECLEN**	**F**	Length of passed record in bytes
4 →	**RECORD**	**XL*n***	Original record (Data)

Note: Usually the record is inserted. Only if the key of the last record read is identical with the key passed in the USRPKY function, the record is updated (REWRITE). Otherwise, for identical keys, a new record with the same key is inserted if duplicate keys are allowed.

3.4.8 Function USRDEL

Delete the last read record.

Parameters:

1 ↔	**WORKAREA**	**256F**		Working storage area
2 ←	**RETCO**	**F**		Return code
	=	**0**		No error
	=	**-1**		Invalid function
	=	**5**		No active record found
	=	**x'0FXXXXXX'**		Other error codes

3.5 User exits

Addressing mode when calling user exits:

User exits can be written for any addressing mode (AMODE=ANY, AMODE=31, AMODE=24, no mode specified).

The addressing mode only has to be taken into account if FLAM is loaded with AMODE=31 and the user exit is for some reason only able to run with AMODE=24. Only in this case the addressing mode must be switched over in the user exit itself. It must be noted that the save area, the return address, the parameter list and the parameters can be addressed only in AMODE=31. The addressing mode used by FLAM is stored in the most significant bit of R14 and can be looked up there.

In all other cases, the addressing mode is already set correctly and it is switched over again by FLAM after the return if this is necessary. It is irrelevant whether the return is executed with a BR14 or a BSM 0,14.

3.5.1 Input original data EXK10

In this user exit the original records (which shall be compressed) are passed to a user module immediately after they are read from the input file. This user exit can be used with the FLAM utility or with the subprogram FLAMUP. In this user exit records can be accepted, modified, inserted and deleted.

The exit is activated via the parameter EXK10=<name>. The user exit module must be contained in the library that has been assigned with the STEPLIB command.

Name: free choice (max. 8 characters)

Register usage:

→ R1: Address of parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameter list:

1 →	**FUCO**	**F**	Function code
	= 0		First call for file (after OPEN)
	= 4		Record read and passed
	= 8		Last call for file (before CLOSE)
2 ←	**RETCO**	**F**	Return code
	= 0		Accept record / no error
	= 4		Do not accept record
	= 8		Insert record
	= 12		Enforce end of compression
	= 16		Error in user exit; abnormal termination
3 ↔	**RECPTR**	**A**	Record pointer
4 ↔	**RECLEN**	**F**	Record length (maximum 32760)
5 ↔	**EXWORK**	**256F**	During the first call the working storage area contains the symbolic file name of the original file within the first 8 characters. The rest of the area is padded with x00. This area can be used by the user exit module for any purpose. With each call this working storage area is made available again with the old content.

Note: If a record shall be extended or inserted, the necessary working storage area must be provided by the user exit module.

Return code 12 is only necessary if the compression shall be finished before the end of the input file is reached.

For function code 0 and 8 no record is passed to the module. For function code 8 it is allowed to insert a record with return code 8.

For return code 8 the record provided by the user exit module is processed. Then the user exit module is called again for the old record from the input file.

Table of valid function codes and return codes:

function code:		0	4	8
return code:	0	x	x	x
	4		x	
	8		x	x
	12		x	
	16	x	x	x

3.5.2 Output compressed data EXK20

In this user exit the compressed records are passed to a user module immediately before they are written to the compressed file. This user exit can be used with the FLAM utility or with the subprogram FLAMUP. In this user exit records can be accepted, modified, inserted and deleted.

The exit is activated via the parameter EXK20=<name>. The user exit module must be contained in the library that has been assigned with the STEPLIB command.

Name: free choice (max. 8 characters)

Register usage:

→ R1: Address of parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameter list:

1 →	**FUCO**		**F**	Functions code
	=	**0**		First call for file (after OPEN)
	=	**4**		Record passed
	=	**8**		Last call for file (before CLOSE)
2 ←	**RETCO**		**F**	Return code
	=	**0**		Accept record / no error
	=	**4**		Do not accept record
	=	**8**		Insert record
	=	**12**		Enforce end of compression
	=	**16**		Error in exit; abnormal termination
3 ↔	**RECPTR**		**A**	Record pointer
4 ↔	**RECLEN**		**F**	Record length (maximum 32760)
5 ↔	**EXWORK**		**256F**	During the first call the working storage area contains the symbolic file name of the original file within the first 8 characters. The rest of the area is padded with x'00'. This area can be used by the user exit module for any purpose. With each call this working storage area is made available again with the old content.

Note: If a record shall be extended or inserted, the necessary working storage area must be provided by the user exit module.

Return code 12 is only necessary if the compression shall be finished before the end of the input file is reached.

For function code 0 and 8 no record is passed to the module. For function code 8 it is allowed to insert a record with return code 8.

For return code 8 the record provided by the user exit module is written. Then the user exit module is called again for the old compressed record.

Table of valid function codes and return codes:

function code:		0	4	8
return code:	0	x	x	x
	4		x	
	8		x	x
	12		x	
	16	x	x	x

3.5.3 Output original data EXD10

In this user exit the decompressed records are passed to a user module immediately before they are written to the output file. This user exit can be used with the FLAM utility or with the subprogram FLAMUP. In this user exit records can be accepted, modified, inserted and deleted.

The exit is activated via the parameter EXD10=<name>. The user exit module must be contained in the library that has been assigned with the STEPLIB command.

Name: free choice (max. 8 characters)

Register usage:

→ R1: Address of parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameter list:

1 →	**FUCO**		**F**	Functions code
	=	**0**		First call for file (after OPEN)
	=	**4**		Record passed
	=	**8**		Last call for file (before CLOSE)
2 ←	**RETCO**		**F**	Return code
	=	**0**		Accept record / no error
	=	**4**		Do not accept record
	=	**8**		Insert record
	=	**12**		Enforce end of decompression
	=	**16**		Error in exit; abnormal termination
3 ↔	**RECPTR**		**A**	Record pointer
4 ↔	**RECLEN**		**F**	Record length (maximum 32760)
5 ↔	**EXWORK**		**256F**	During the first call the working storage area contains the symbolic file name of the original file within the first 8 characters. The rest of the area is padded with x'00'. This area can be used by the user exit module for any purpose. With each call this working storage area is made available again with the old content.

Note: If a record shall be extended or inserted, the necessary working storage area must be provided by the user exit module.

Return code 12 is only necessary if decompression shall be finished before the end of the compressed file is reached.

For function code 0 and 8 no record is passed to the module. For function code 8 it is allowed to insert a record with return code 8.

For return code 8 the record provided by the user exit module is written. Then the user exit module is called again for the old record.

A change of the record length is accepted if the output file is defined with RECFORM=V.

Table of valid function codes and return codes:

function code:		0	4	8
return code:	0	x	x	x
	4		x	
	8		x	x
	12		x	
	16	x	x	x

3.5.4 Input compressed data EXD20

In this user exit the compressed records are passed to a user module immediately after they are read from the compressed file. This user exit can be used with the FLAM utility or with the subprogram FLAMUP and in the record level interface FLAMREC. In this user exit records can be accepted, modified and deleted.

The exit is activated via the parameter EXD20=<name>. The user exit module must be contained in the library that has been assigned with the STEPLIB command.

Name: free choice (max. 8 characters)

Register usage:

→ R1: Address of parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameter list:

1 →	**FUCO**		**F**	Functions code
	=	**0**		First call for file (after OPEN)
	=	**4**		Record passed
	=	**8**		Last call for file (before CLOSE)
2 ←	**RETCO**		**F**	Return code
	=	**0**		Accept record / no error
	=	**4**		Do not accept record
	=	**8**		Insert record
	=	**12**		Enforce end of decompression
	=	**16**		Error in exit; abnormal termination
3 ↔	**RECPTR**		**A**	Record pointer
4 ↔	**RECLEN**		**F**	Record length (maximum 32760)
5 ↔	**EXWORK**		**256F**	During the first call the working storage area contains the symbolic file name of the original file within the first 8 characters. The rest of the area is padded with x'00'. This area can be used by the user exit module for any purpose. With each call this working storage area is made available again with the old content.

Note: If a record shall be extended or inserted, the necessary working storage area must be provided by the user exit module.

Return code 12 is only necessary if decompression shall be finished before the end of the compressed file is reached.

Because of the necessary synchronisation with the construction of the matrix this return code is not always possible.

For function code 0 and 8 no record is passed to the module.

Table of valid function codes and return codes:

Funktionscode:		0	4	8
Returncode:	0	x	x	x
	4		x	
	8		x	x
	12		(x)	
	16	x	x	x

3.5.5 Key management KMEXIT

This user exit is an interface to a special (e.g. user written) key management system.

On encryption, parameters (KMPARM=…) are passed to the module. It returns a key for encryption of the FLAMFILE and a string up to 512 byte. These data are stored in the FLAMFILE as an user header (see parameter COMMENT or function FLMPUH).

On decryption, parameters (KMPARM=…) and the data stored in the user header are passed to the exit. The module returns the same key as on encryption.

It is up to the module, how to create a key and what kind of information are to be stored into the user header of the FLAMFILE. These data will help the module to find the correct key on decryption.

The exit is activated via the parameter KMEXIT=<name>.

The user exit module must be contained in the library that has been assigned with the STEPLIB command.

Name: free choice (max. 8 characters)

Register usage:

→ R1: Address of parameter list
→ R13: Points to save area (18 words)
→ R14: Contains return address
→ R15: Contains call address

Parameter list:

1 → FUCO F Functions code
= 0 Decryption
= 1 Encryption

2 ← RETCO F Return code
= 0 No error
= else Error(s) detected

3 → PARMLEN F Length of parameter (up to 256 byte)

4 → PARAM XLn Parameter

5 ↔ DATALEN F Length of data
Decryption:
→ Length of data
Encryption:
→ Buffer length of field DATA (512)
← Length of returned data (max. 512)

6 ↔ DATA **XLn** Data (in length of DATALEN)

7 ↔ CKYLEN **F** Length of key for en-/decryption
→ Buffer length of field CRYPTOKEY (64)
← Length of returned key (max. 64)

8 ← CRYPTOKEY XLn Returned key (in length of CKYLEN)

9 ↔ MSGLEN **F** Message length
→ Length of message buffer (field MESSAGE) (128)
← Length of returned Message length (max. 128)

10 ← MESSAGE **CLn** Returned message (in length of MSGLEN)

If a message is returned (MSGLEN > 0), it is sent to the protocol (FLM0445 …).

The returned key is not sent to the protocol.

The data DATA are stored 'as is' in the user header of the FLAMFILE. If a special security is required it has to be done by the exit.

Usage of this exit overrules the parameter COMMENT and CRYPTOKEY, if any.

The exit is called only once if encryption of many files into a Group-FLAMFILE (C,FLAMIN=user.*) is required. It is called only at the beginning of the first file.

The exit is called many times for decryption of many FLAMFILEs (D,FLAMFILE=user.*.aes). It is called after opening each FLAMFILE.

In the DD-statement concatenated FLAMFILEs are treated as one FLAMFILE!

Note: look for an example in FLAM.SRCLIB (KMX-SAMPL).

3.6 Bi-/serial compression BIFLAMK

BIFLAMK is used for compression of data record by record. The compressed data is always returned in the same call.

BIFLAMK is reentrant. For operation a working storage area is needed that must be provided by the calling program. The content of the area before the call is ignored by BIFLAMK. All calls to BIFLAMK are totally independent from each other. All areas can have any alignment. The working storage area for the input record and for the compressed record must not overlap. A compression in place is not possible.

Name: BIFLAMK

Parameters:

→ R1:	Address of parameter list
→ R13:	Points to save area (18 words)
→ R14:	Contains return adress
→ R15:	Contains call adress

Parameter list:

1 →	**FUCO**		**F**	Function code
	=	**0**		Serial compression without sample
	=	**8**		Biserial compression with sample, serial post compression of the remainder and static sample
	=	**9**		Sample record for biserial compression with serial post compression
	=	**10**		Biserial compression with sample, serial post compression of the remainder and dynamic sample
	=	**11**		Sample record for biserial compression with serial post compression
	=	**12**		Biserial compression with sample, serial post compression and encryption of the remainder and static sample
	=	**13**		Sample record for biserial compression with encryption
	=	**14**		Biserial compression with sample, serial post compression and encryption of the remainder and dynamic sample
	=	**15**		Sample record for biserial compression with encryption
2 ←	**RETCO**		**F**	Return code
	=	**0**		Function executed
	=	**2**		Invalid function code
	=	**3**		Length error - working storage too small - return area too small - record bigger than 32767 bytes

3 →	**WORK**	**XL*n***	Working storage area. The working storage area must be at least 512 bytes in size. For biserial compression the working storage area must be at least 512 bytes + length of return area.
4 →	**WRKLEN**	**F**	Length of working storage area in bytes
5 →	**BUFLEN**	**F**	Length of return area or maximum length of compressed record; this size must be at least 8 bytes + 1.1 * length of original record.
6 ←	**RECIN**	**XL*n***	Original record
7 →	**RECLEN**	**F**	Record length in bytes
8 ←	**COMPREC**	**XL*n***	Compressed record (length of this area = BUFLEN)
9 ←	**COMPLEN**	**F**	Length of compressed record in bytes
			The next two parameters are only used for biserial compression:
10 →	**SAMPREC**	**XL*n***	Sample
11 →	**SAMPLEN**	**F**	Sample length in bytes

3.7 Bi-/serial decompression BIFLAMD

BIFLAMD is used for record by record decompression of compressed data that was created with BIFLAMK.

BIFLAMD is reentrant. For operation a working storage area is needed that must be provided by the calling program. The content of the area before the call is ignored by BIFLAMD. All calls to BIFLAMD are totally independent from each other. All areas can have any alignment. The working storage area for the compressed record, for the sample record, and for the output record must not overlap. A decompression in place is not possible.

Name: BIFLAMD

Parameters:

→ R1:	Address of parameter list
→ R13:	Points to save area (18 words)
→ R14:	Contains return adress
→ R15:	Contains call adress

Parameter list:

1 →	**FUCO**	**F**	Function code
	= 0		Serial decompression without sample
	= 8		Biserial decompression with sample
2 ←	**RETCO**	**F**	Return code
	= 0		Function executed
	= 1		Sample record for biserial decompression returned; no original record was written (only during biserial decompression)
	= 2		Invalid function code or record is compressed serially (function code = 8) or record is compressed biserially (function code = 0).
	= 3		Length error - working storage area too small - compressed record smaller than 3 bytes - return area too small.
	= 4		Checksum error in compressed record
	= 5		Checksum error in sample record (only for dynamic samples)
	= 6		Checksum error in original record
	= 7		Other errors in compressed record
	= 8		Sample record is shorter as during compression (only for biserial compression)
	= 9		Compressed record too short
3 →	**WORK**	**XL*n***	Working storage area. The working storage area must be at least 512 bytes in size. For biserial compression the working storage area must be at least 512 bytes + 1.125 * the sum of length of the return areas.

4 →	**WRKLEN**	**F**	Length of working storage area in bytes
5 →	**BUFLEN**	**F**	Length of return areas; maximum length of the original record or the sample record in bytes
6 ↔	**RECOUT**	**XL*n***	Original record (length of area = BUFLEN)
7 ←	**RECLEN**	**F**	Record length in bytes
8 →	**COMPREC**	**XL*n***	Compressed record
9 →	**COMPLEN**	**F**	Length of compressed record in bytes
			The next two parameters are only needed for biserial decompression:
10 ↔	**SAMPREC**	**XL*n***	Sample record (length of area = BUFLEN)
11 ↔	**SAMPLEN**	**F**	Length of sample record in bytes
12 ←	**RETCO**	**F**	Return code

3.8 Utilities

Some utilities are released that improves support form FLAM and FLAMFILEs.

3.8.1 FLAMCKV

FLAMCKV analyses a cataloged VSAM-KSDS FLAMFILE. It displays the procentual distribution of record lengths and the number of records needed for one FLAM matrix (in FLPRINT, RECFM=VB,LRECL=124).

This is very important for direct access to a VSAM-KSDS FLAMFILE.

Please remember:
FLAM needs a complete matrix (i.e. a block of self-contained compressed records) for decompression.

Even for direct access to a single record this complete matrix is requirerd. So performance is best when this matrix is stored in one VSAM record.

If the VSAM record is too small to fit a complete matrix, FLAM has to read or write mutiple records. This decreases perfomance.

These requirements are not so important for a small amount of data. But if thousands or millions of records are stored it becomes more and more relevant.

Example:

```
//CKV  EXEC PGM=FLAMCKV
//STEPLIB   DD DSN=FLAM.LOAD,DISP=SHR
//FLPRINT   DD SYSOUT=*
//FLAMFILE  DD DSN=USER.XMLDAT1.ADC,DISP=SHR
```

FLPRINT protocol:

```
* FLAMCKV, a program of FLAM utilities * copyright (c) 2009 by limes datentech-
nik gmbh

Utility to check a VSAM-KSDS FLAMFILE for proper settings

Data Set Name : USER.XMLDAT1.ADC

RECSIZE :   4,096  CINV :  16,384  RKP :       0  KEYLEN :   34
High used relative byte address (HURBA):               737,280

Number of Records :             164
Number of Bytes   :         172,216

Min. RECSIZE :     968    Max. RECSIZE :   1,186

Number of VSAM-records needed for one FLAM-matrix:
 1 :            164
 2 :              0
 3 :              0
 4 :              0
 5 :              0
 6 :              0
 7 :              0
 8 :              0
 9 :              0
10 :              0
 > :              0

Record length distribution:

RECSIZE   No. Records   in Percent
-------+--------------+----------
 10 %               0        0
 20 %               0        0
 30 %             164      100
 40 %               0        0
 50 %               0        0
 60 %               0        0
 70 %               0        0
 80 %               0        0
 90 %               0        0
100 %               0        0
```

The above expample 164 VSAM-KSDS records are stored in the FLAMFILE. Only one record is to be read for one complete matrix., this is best.

If you see many records >10 (i.e. more than 10 VSAM records are to be read for one matrix) it is recommended to reorganize the KSDS file: decompress and compress it again with a longer RECSIZE (and probably CISIZE) for the new FLAMFILE. You can use the FLAM subsystem (if available on your machine) for FLAMIN file, so no original data are stored even temporarily to disk.

In this example all records have lengths about 30% of the cataloged RECSIZE. The file could be cataloged with a shorter RECSIZE parameter.

Having a high variation in the record length distribution you probably have very inhomogeneous data (different record lengths, record types, …), so one matrix differs highly from the other in length.

Having max. record length and one matrix per record is best. But increasing the cataloged RECSIZE is required when many records are needed for one matrix.

3.8.2 FLAMCTAB

FLAMCTAB reads a file (DD-name TABLE) and creates a translation table module (look for parameter TRANSLATE) from the input data in an output LOAD library (DD-Name FLAMLIB).

So it is no longer necessary to use an Assembler source with assembling and binding a loadable module.

A translation table consists of 256 byte of data. These data have to be stored in the input file. This file may have any data organization or format, as required by your organization.

Records with an asterisk '*' in the first column are regarded as comment lines.

Input data exceeding 256 bytes are truncated to 256 bytes, a warning will be logged and the program ends with cond code 4.

Shorter input leads to an unexpected EOF (end of file).

The file may be edited with ISPF or any other editor or program. But it is recommended to use the table editor of the FLAM (Windows) distribution. You can create any table interactively. So it is automatically error-checked. Transfer this file in binary mode to host and use it as input for FLAMCTAB.

An 8 byte module name is used as a parameter in the EXEC statement .

Returncodes are the same as in FLAM.

The LOADlib FLAMLIB requires module FLAMTR11 from the FLAM distribution.

Example:

```
//TAB  EXEC PGM=FLAMCTAB,PARM=TRAEDOS
//STEPLIB   DD DSN=FLAM.LOAD,DISP=SHR
//FLPRINT   DD SYSOUT=*
//FLAMLIB   DD DSN=FLAM.LOAD,DISP=SHR
//TABLE     DD DSN=USER.TABLE.DAT,DISP=SHR
```

and the protocol FLPRINT:

```
FLAMCTAB, a program of FLAM utilities      Copyright (C) 2009 by limes daten-
technik gmbh   10:17:29   8/31/2009

Creates a translation table module from an 256 byte input file, loadable by
FLAM

TABLE file: USER.TABLE.TAB

To create : Member TRAEDOS in LOAD library FLAMLIB

DONE SUCCESSFULLY.
```

There, translation table module TRAEDOS is created. The input data are stored in the file USER.TABLE.DAT, the library FLAM.LOAD is the STEPLIB library as well as FLAMLIB. The protocol is printed directly to the JES log.

„DONE SUCCESSFULLY“ means that the table module was created and stored. The input data were exactly 256 bytes (no warnings, no error messages).

Now FLAM is able to use this module by entering the parameter TRANSLATE=TRAEDOS when FLAMLIB is concatenated to STEPLIB in the Job.

Take care: if FLAMLIB is integrated in the LINKLIST chain of the system, you have to refresh the member index using the console command LLA REFRESH before first using of the new module.

3.8.3 FLAMDIR

FLAMDIR reads a FLAMFILE (DD-name FLAMFILE) and logs the directory of the compressed/encrypted files. The output (DD-name FLPRINT) looks like an ISPF panel 3.4 or option ‚I' in FLAM start panel or FLTOC clist.

Using the FLAM-parameter „D,SHOW=DIR" you still get a full protocol of the directory of the FLAMFILE.

But using FLAMDIR the protocol is a conveniently laid out short summary of the directory of a Group FLAMFILE.

Example:

```
//DIR  EXEC PGM=FLAMDIR
//STEPLIB   DD DSN=FLAM.LOAD,DISP=SHR
//FLPRINT   DD SYSOUT=*
//FLAMFILE  DD DSN=USER.ARCHIV.ADC,DISP=SHR
```

and protocol FLPRINT:

```
* FLAMDIR, a program of FLAM utilities * copyright (c) 1999-2009 by limes dat-
entechnik gmbh

*
* Table of Contents of FLAMFILE USER.ARCHIV.ADC
*
* Original File Name        System  ORG  RECFM  RECSI BLKSI Space
*-------------------------------------------------------------------
*
  FLAMV43A.ADATA(BIFLAMD)    zOS    SEQ   VB     8184 27998 91150 KB
  FLAMV43A.ADATA(BIFLAMK)    zOS    SEQ   VB     8184 27998 91150 KB
  FLAMV42A.CLIST($FABOUT)    zOS    SEQ   FB       80 27920   250 KB
  FLAMV43A.CLIST(CUST)       zOS    SEQ   FB       80 27920   250 KB
  FLAMV43A.CLIST(FLAMLIBS)   zOS    SEQ   FB       80 27920   250 KB
  FLAMV43A.LISTEN.ADC        zOS    SEQ   FB      512 23040  5050 KB
  FLAMV42A.LOAD(BIFLAMD)     zOS    SEQ   U         0  6144  3450 KB
  FLAMV42A.LOAD(BIFLAMK)     zOS    SEQ   U         0  6144  3450 KB
.
.
```

In each row the filename, the creating system, its organization, data format, recordsize, blocksize, and the amount of space are printed.

If the FLAMFILE was created on a system other than z/OS the space column is empty.

FLAM (MVS)

User Manual

Chapter 4:

Method of Operation

Content

4. Method of Operation

We discussed in the previous chapters where compression is useful, which functions are offered by FLAM and the use of these functions in different contexts.

In this chapter we discuss the principles of operation from an internal view.

We have to make a distinction between FLAM as a utility for processing whole files (which may be called as an independent main program or as a subprogram) and an interface for processing on record level (the application exchanges single records with FLAM).

Utility

FLAM as a utility can be started on job control level via a job control command. Parameters define the mode of operation. Depending on the operating system, parameters may be supplied as a part of the command or may be entered via the screen.

In addition can be parameters read from a parameter file. This file can be assigned to the process using job control commands or command parameters.

Subprogram

FLAM as a subprogram offers the same functionality as used as a main program. The difference is, that this time FLAM is called from an application or a driver program. It is possible to specify parameters with the call.

Record Level Interface

Using the record level interface a user program may control compression and decompression. FLAM maintains the compressed file beyond that interface. It is possible that an application program processes more than one compressed file at the same time. To the application program the FLAM record level interface behaves in the same way as an operating system interface for file access. The difference is, that the file is stored compressed and that the interface is the same on all operating systems.

User I/O

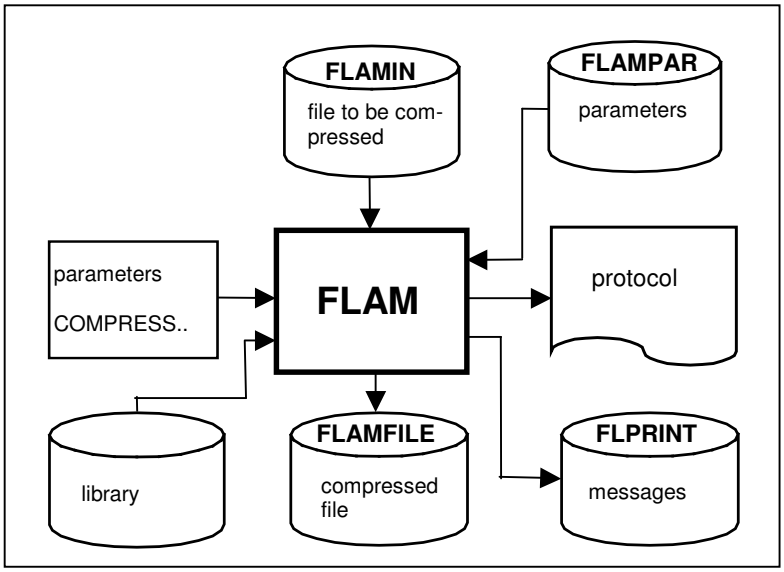
The user interface for I/O allows FLAM to use user defined file access methods instead of the operating system defined ones. This is possible under the utility for the original file and under the FLAM record level interface for both original file and compressed file FLAM.

User Exits

User exits allow pre- and post processing of records before compression and after decompression under FLAM as utility as well as pre- and post processing of compressed records under both FLAM as utility and FLAM record level interface. E.g., this user exits can be used for encryption purposes or for selective processing of original data.

4.1 Processing of file with the utility

4.1.1 Compression



Data flow during compression

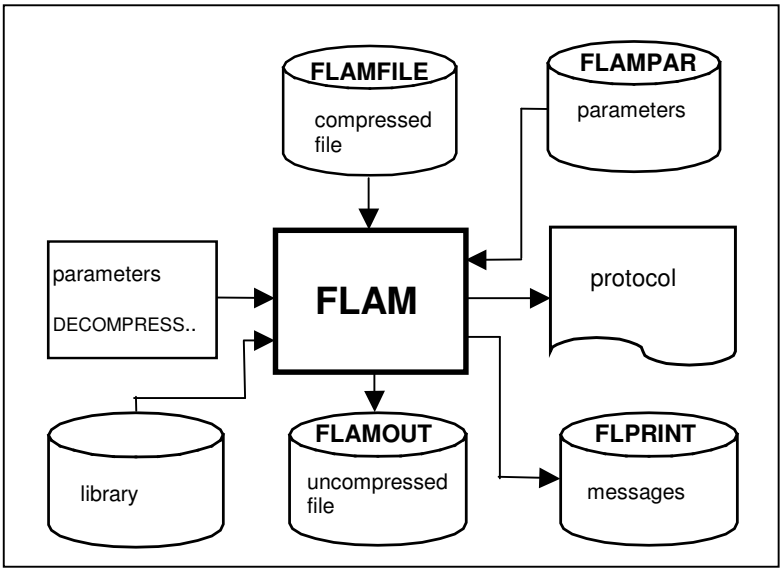
FLAM reads the uncompressed records from the original file, compresses them and writes them into the compressed file.

To do so FLAM needs specifications about compression mode and file format of both original file and compressed file.

The resulting compressed file can be decompressed with the FLAM utility, with the FLAM subprogram or with the FLAM record level interface.

Optionally a protocol is printed.

4.1.2 Decompression



Data flow during decompression

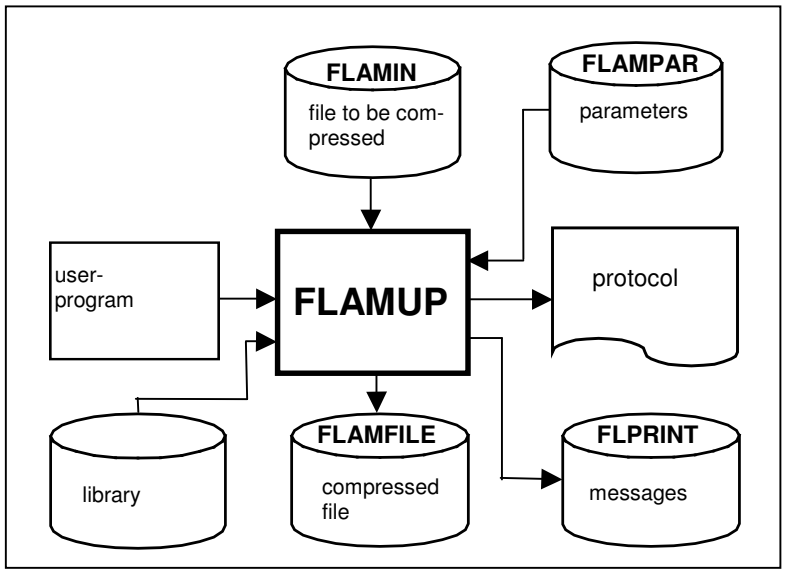
FLAM reads the compressed records from the compressed file, decompresses them and writes them into the target file.

If the file attributes of the original file are unknown (no file header available) the user has to specify the file format of the target file. By default FLAM will create a target file with variable record length.

On fact, for decompression both compressed file and target file must be assigned to FLAM.
Optionally a protocol can be printed.

4.2 File processing with the FLAM subprogram

4.2.1 Compression



Data flow during compression

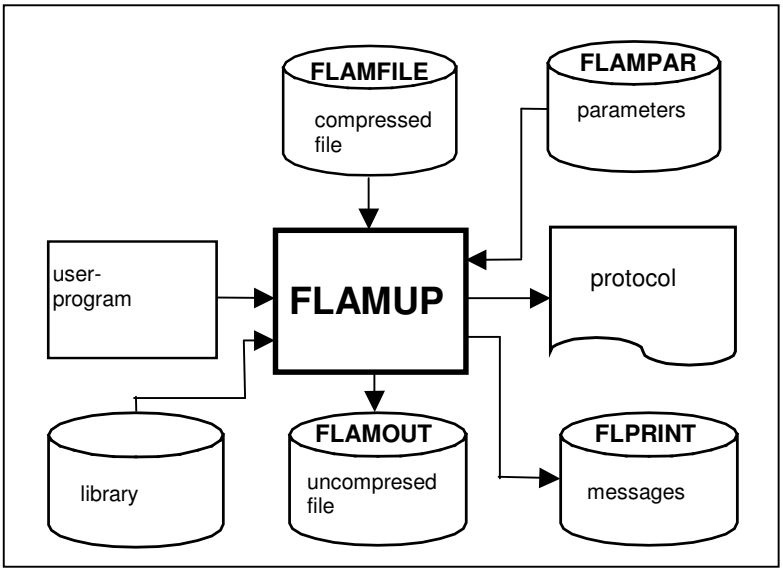
FLAMUP reads - similar to FLAM - the uncompressed records from the original file, compresses them and writes them into the compressed file.

Both compressed file and target file must be assigned to FLAMUP.

File parameters may be defined with the FLAMUP call or can be provided via a parameter file.

Optionally a protocol can be printed.

4.2.2 Decompression



Data flow during decompression

FLAMUP reads - similar to FLAM - the compressed records from the compressed file, decompresses them and writes them into the target file. The target file must be allocated in the same format as the original file or as specified by the user.

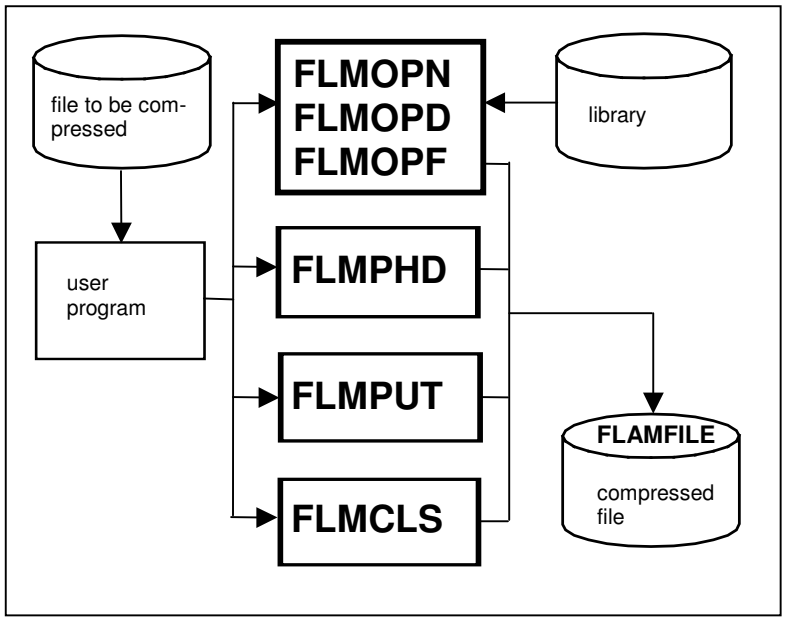
FLAMUP needs for decompression specifications about the target file and the compressed file - similar as with decompression with the FLAM utility.

Parameters may be defined with the FLAMUP call or can be provided via a parameter file.

Optionally a protocol can be printed.

4.3 Processing of records

4.3.1 Compression



Data flow during compression

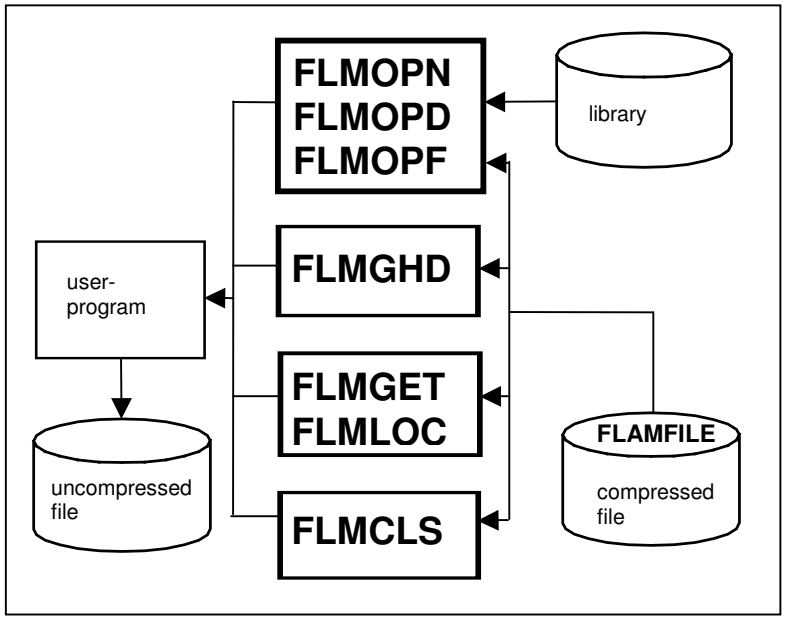
The user application passes the records via the record level interface directly to FLAM. FLAM collects these records until the maximal number of records within a block (MAXRECORDS) is hit or the provided buffer (MAXUFFER) is filled up. Then the data is compressed and written to the compressed file. The block structure is invisible to the user program. The user program only interacts on record level with FLAM, FLAM creates blocks and initializes compression autonomously.

The record level interface is controlled from the user program via different functions (FLMOPN, ..., FLMCLS).

Sequence of function calls:

1. **FLMOPN** The record level interface is opened for output. This function call may be followed by FLMOPD and FLMOPF if parameter specification is required.
2. **FLMPHD** Sending file header information (optional).
3. **FLMPUT** Sending a record. This function call must be repeated until all records are passed to FLAM.
4. **FLMCLS** Closing the interface and optionally receiving of statistics. The printout of a protocol and the definition of parameters in a parameter file is not provided with the record level interface.

4.3.2 Decompression



Data flow during decompression

The user program receives the decompressed records via the record level interface directly from FLAM. Records can be retrieved sequentially or randomly using keys. FLAM reads the compressed file block by block and decompresses the block autonomously. For the user program this process is invisible. The end of the compressed file or the end of an original file within a concatenated compressed file is signalled to the user program via a return code.

The record level interface is controlled from the user program via different functions (FLMOPN, ..., FLMCLS).

Sequence of function calls:

1. FLMOPN The record level interface is opened for input. This function call may be followed by FLMOPD and FLMOPF if parameter specification is required.

2. FLMGHD Receiving file header information (optional). This function can be applied repeatedly if a concatenated compressed file contains more than one file header.

3. FLMGET Receiving of the decompressed original record. This function can be executed repeatedly, until all record have been received from FLAM or until FLAM was closed with the function FLMCLS.

4. FLMCLS Closing the interface and optionally receiving of statistics. he printout of a protocol and the definition of parameters in a parameter file is not provided with the record level interface.

4.4 User I/O

The user can replace the FLAM file accesses methods with own file access methods by using the User I/O Interface. These access methods are used by the FLAM utility for accessing the original file, to the compressed file and for the target file.

In the record level interface, however, only access routines for the compressed file exist.

The use of user defined access routines can be specified for each file separately via the parameters DEVICE=USER or IDEVCE, ODEVICE. However, the user provided I/O routines must be linked to the FLAM utility or to the FLAM record level interface before.

The following routines must be provided by the user: Open and Close (USROPN, USRCLS), sequential Read and Write (USRPUT, USRGET), optionally random access Read and Write (USRPKY, USRGKY), Delete and Positioning (USRDEL, USRPOS).

How it works:

1. USROPN: For each allocated file this function is called once and only once. A working area of 1024 byte is passed to the routine. This working area acts as a file specific memory and is passed from function to function until USRCLS is called.

Files are identified via symbolic file names. The access mode is specified in parameter OPENMODE: INPUT, OUTPUT, INOUT, OUTIN. File format and file attributes are specified in the parameter RECFORM, RECSIZE, BLKSIZE, etc.. These settings may be adapted to special requirements.

Via predefined and user defined return codes the successful execution as well as special status information and errors can be reported to the higher layers.

2. USRCLS: This function is called to close the file. The 1024 byte working area reserved for this file is deallocated after returning control to FLAM.

3. USRGET: This function is called to retrieve the next record. The maximal amount of characters that may be returned are specified in parameter BUFLEN. If it is necessary to truncate the record this must be signalled via a return code. Also the end of file condition must be signalled via a return code. For each record returned the record length must be returned as well (also for records with fixed record length!).

4. USRPUT: This function is called to write a record. If it is not possible to write the record in full length the return code record truncated must be reported to the higher layers. Another possibility is to fill the record with the padding character (PADCHAR) as specified in USROPN and to return the corresponding return code.

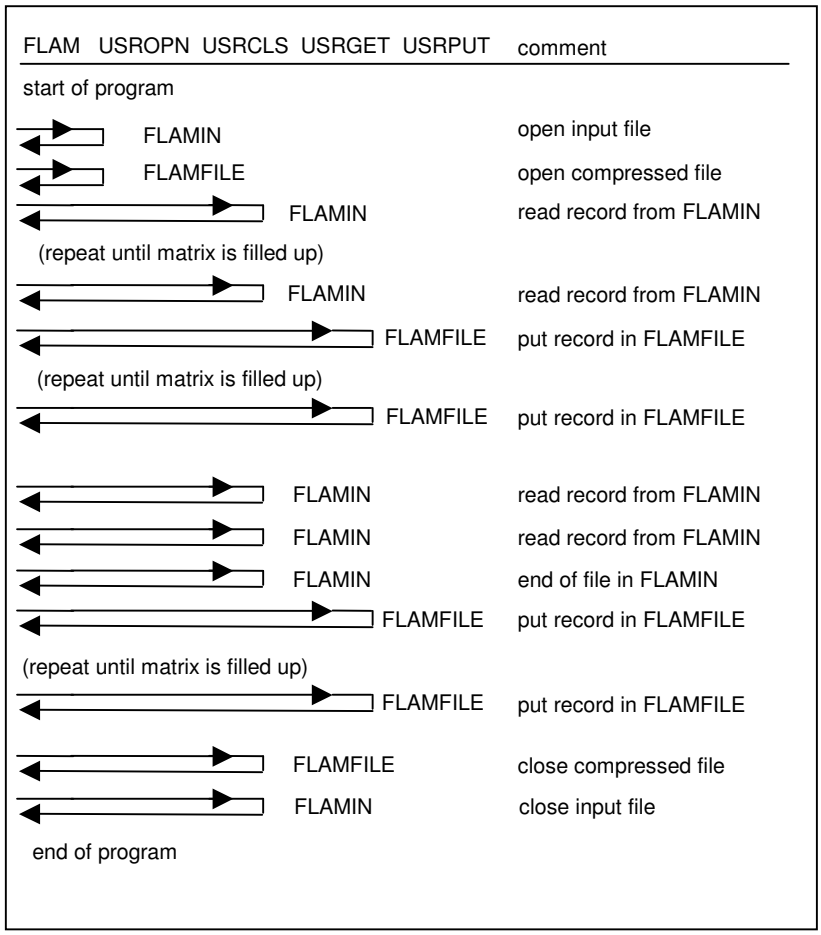
5. USRPOS: This function is called to move the current read or write pointer. Relative positioning (forward and backward) from the current position as well as absolute positioning from file start or file end is possible.

6. USRGKY: With this function a record with a specified key is read. The corresponding key is contained in the record area at that position and in that length as specified with the parameter (KEYDESC) during USROPN. Reading via key also sets the current read pointer for following USRGET function calls. If no record is found this must be signalled via a return code. Afterwards it is possible to read the record with the nearest higher key with function USRGET.

7. USRPKY: With this function a record with a specified key is updated or inserted. If the record has the same key as the record last read, the old record is replaced by the new record. Otherwise the record is inserted. If this is not possible (duplicate keys may be forbidden) this must be signalled via an appropriate return code. The writing of records with a key also updates the position of the current write pointer.

8. USRDEL: With this function the last read record is deleted.

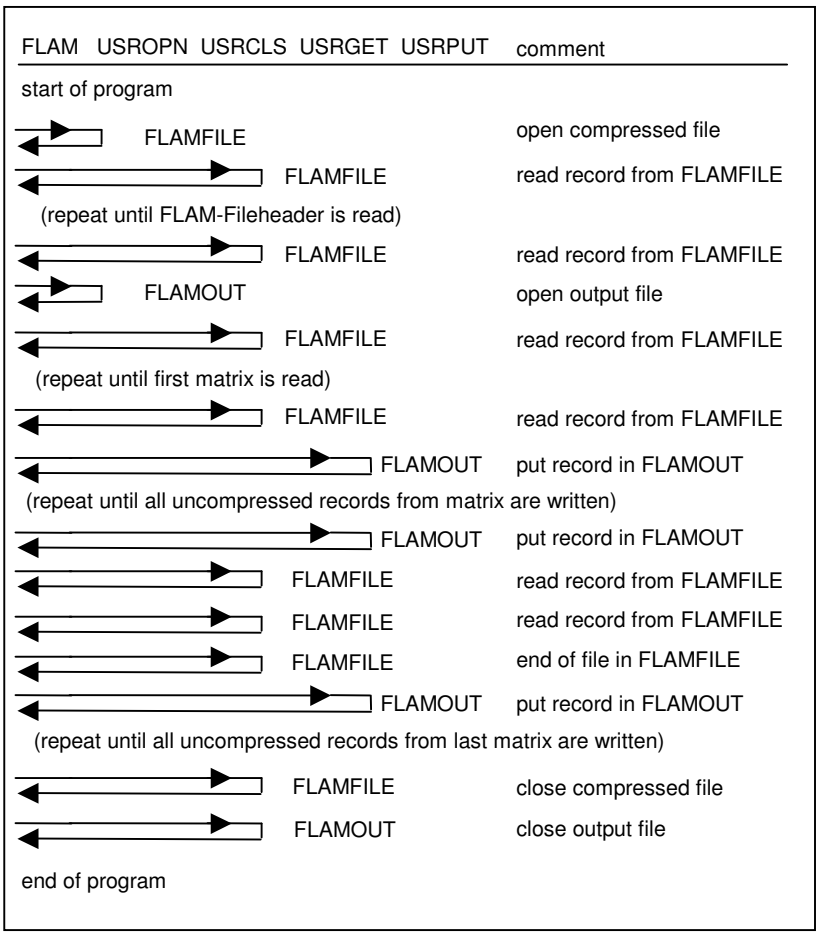
Schematic presentation of compression with User I/O:



Parameter for FLAM or FLAMUP:

COMPRESS, IDEVICE = USER, DEVICE = USER

Schematic presentation of decompression with User I/O:



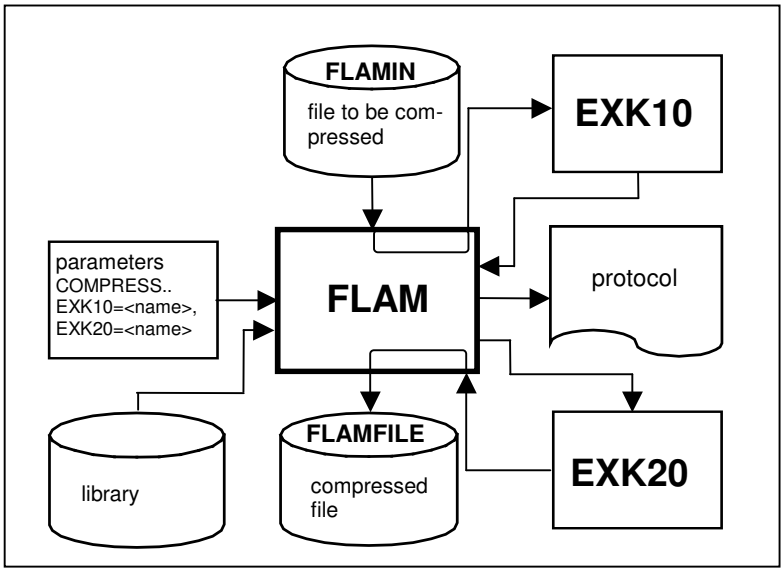
Parameter for FLAM or FLAMUP:

DECOMPRESS, ODEVICE = USER, DEVICE = USER

4.5 User exits

4.5.1 Utility

4.5.1.1 Compression with user exits EXK10, EXK20



Data flow of compression with user exits

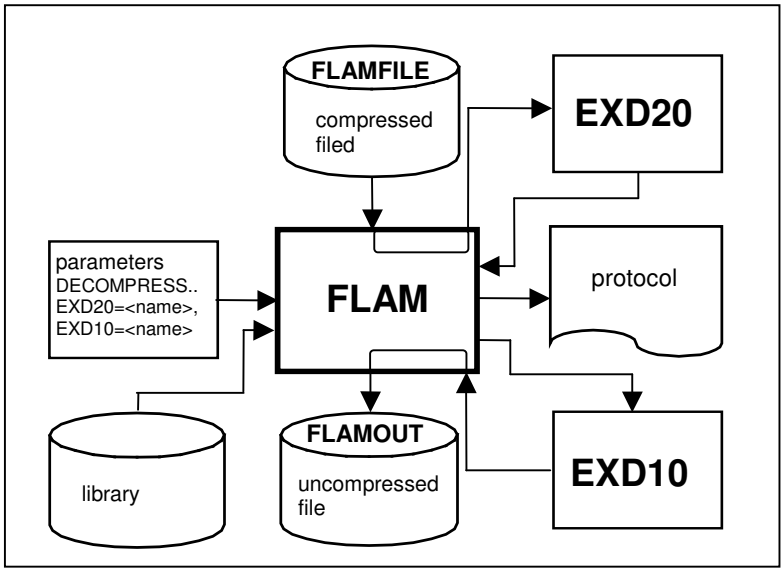
During compression it is possible to call additional routines for pre processing of original records and for post processing of compressed records.

E.g., pre processing may perform a selection of specific records or fields.

Post processing may introduce additional encryption of the compressed record.

Instead of using the more complicated record level interface record orientated processing can be done often with user exit EXK10.

4.5.1.2 Decompression with user exits EXD10, EXD20



Data flow during decompression with user exits

During decompression it is possible to call additional routines for pre processing of compressed records and for post processing of original records.

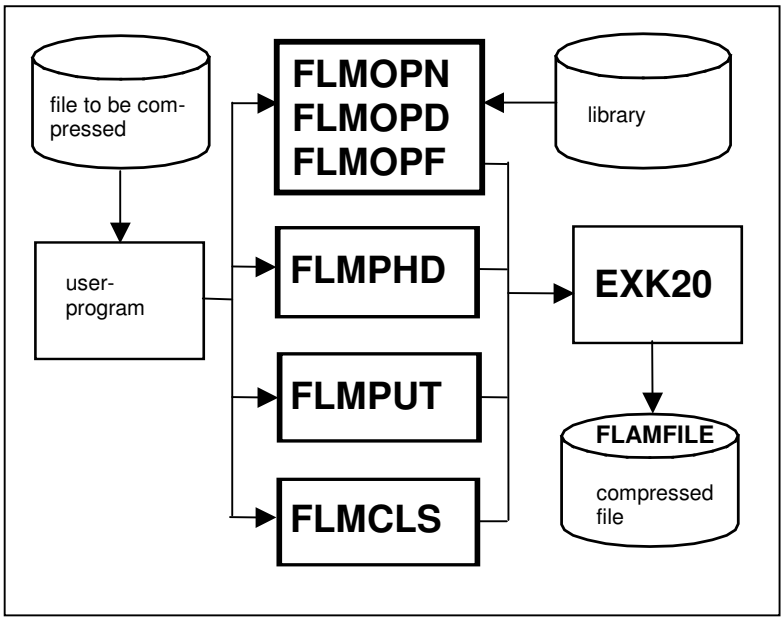
E.g., pre processing may decrypt encrypted records.

Post processing may perform a selection of specific records or fields.

Instead of using the more complicated record level interface record orientated processing can be done often with user exit EXD10.

4.5.2 Record level interface

4.5.2.1 Compression with user exit EXK20

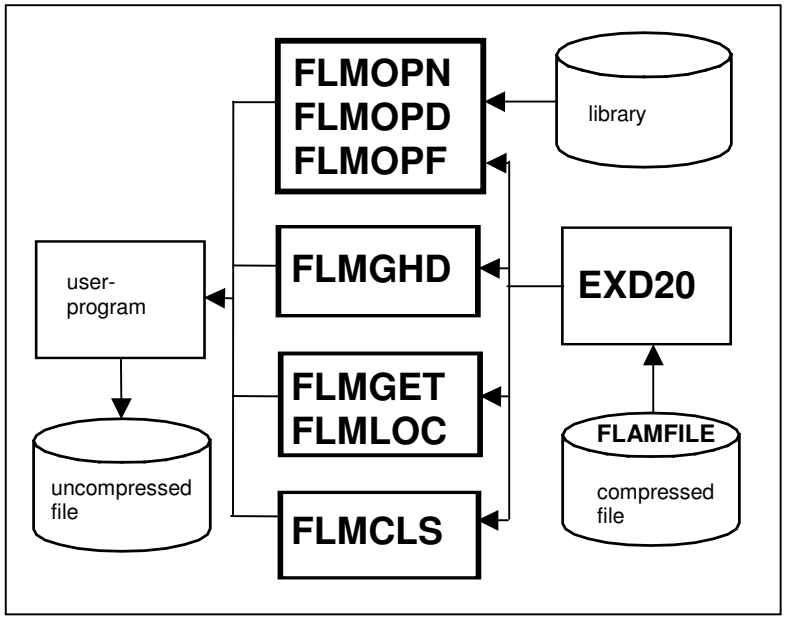


Data flow during compression with user exit

The user exit for compressed records can also be used under the record level interface.

The actual interception or original records by the interface is not affected.

4.5.2.2 Decompression with user exit EXD20

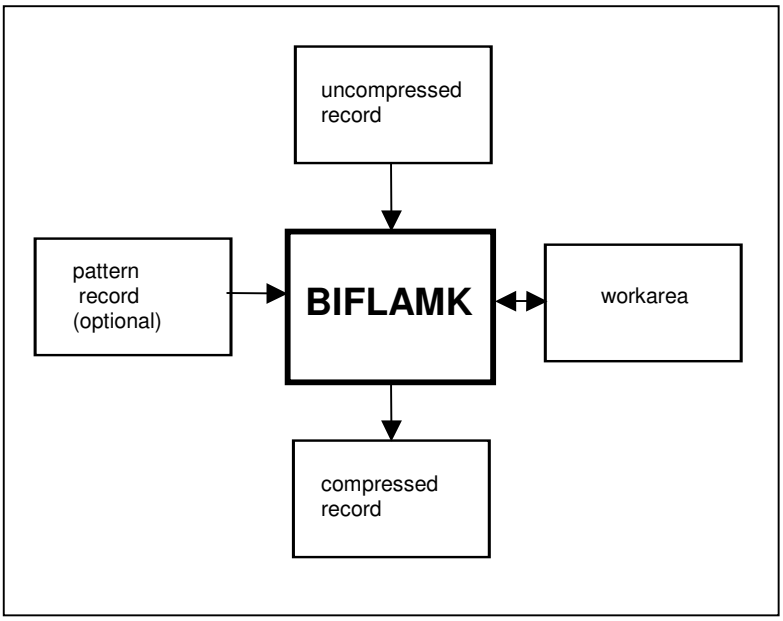


Data flow during decompression with user exit

The user exit for compressed records can also be used under the record level interface.

The actual interception or original records by the interface is not affected.

4.6 Bi-/serial compression



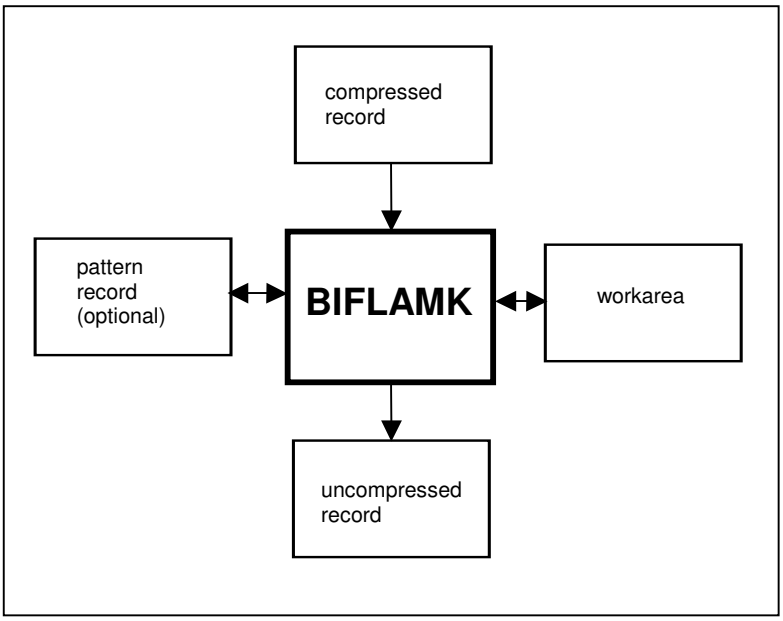
Data flow during compression with BIFLAMK

BIFLAMK processes original records and sample records on a record by record basis.

When only serial compression is used (function code 0) BIFLAMK only processes original records and compiles them into compressed records.

If bi-serial compression with sample (function codes 8,10,12,14) is used an original record is processed in combination with a sample record to create a compressed record. For storage of the sample records (function codes 9,11,13,15) only the sample record is processed to create a compressed record.

4.7 Bi-/serial decompression



Data flow during decompression with BIFLAMD

BIFLAMD processes compressed records record by record if necessary in combination with a sample record - and creates an original record or a sample record.

During serial decompression (function code 0) the compressed record only is processed and will create always an original record. Sample records are not necessary.

During bi-serial compression (function code 8) always one compressed record is processed at a time. Depending on compression the sample record is read in addition and the original record is created. If a sample record was compressed during compression, this sample record is reconstructed during decompression. This situation is signalled with a return code of 1.

4.8 The FLAMFILE

4.8.1 General description

Independently from the Frankenstein-Limes compression method FLAM establishes a concept for file conversion that satisfies compatibility requirements as much as possible. The compressed file created by FLAM is a logical image of the records of the original file. This is the basis for any conversion with FLAM.

The compressed file, the FLAMFILE, is stored by default as a sequential file in accordance with the above-mentioned principle (for random access, index sequential storage is also possible).

The problems which occur with uncompressed files when the requirements are comparable must therefore not be simply ignored due to the fact that FLAM is being used. Some of these problems are made easier to solve by the FLAM concept. Others remain despite FLAM and must therefore be solved along application-specific or organizational lines as before, the only difference being that the original file can be substituted by a FLAMFILE.

FLAM does not solve the problem of incompatible record and field structures in heterogeneous environments. Sometimes users are totally unaware about these problems. FLAM offers user exits that allow to embed special conversion routines. As FLAM is designed as an open system, it will be possible to offer standard solutions for special problem areas in the future.

FLAM requires, that the original records are passed to FLAM record by record. The chosen compression method implies that FLAM works asynchronously. N original records may result in k compressed records where n is unequal k. This can be a problem in special cases.

The FLAMFILE is always created with a fixed record length that may be specified by the user. This results usually in compressed records of equal length. This is necessary because some DP systems only support files with equal record length. This is also true for some data transmission products.

The smallest record length is 80 byte. This allows to process the FLAMFILE in the punched card format (RJE file transfer!). The upper limits are defined by the environment: where the file is stored and which products are used to transfer the file. The upper limit defined by FLAM is 32764 byte.

However, the user may define the format of the record as "variable length" or "fixed length". For a fixed length record that is not filled up, padding is applied.

It is also possible to define different block sizes to optimize the I/O operations, file transfer and storage requirements. This flexibility makes it mostly possible to find a suitable solution for all participating software and hardware environments as well as for special applications.

The FLAMFILE is principally a binary file where all 256 bit combinations are allowed. With this code the FLAMFILE can only be transmitted in transparent mode (MODE=CX8 and VR8).

If transmission on a 7-bit line is performed, file transfer products expand such binary files in a way that ASCII compatibility is guaranteed. Some products convert each half byte into one byte, other products expand 3 bytes into 4 bytes.

If the original file contains only printable characters, FLAM can provide a different code format of the compressed file (MODE=CX7). In this case characters from the original file are not combined with FLAM-descriptors but simply copied into the compressed file. This mode is mostly more efficient than MODE=CX8 with following expansion 3 to 4.

The FLAM-descriptors itself consist under MODE=CX7 only from printable characters that are unambiguous in the international code systems ASCII and EBCDIC. These are all small and big Latin letters, the ten digits and the blank. Any kind of control character, special characters, umlauts are excluded.

The advantage of this method is that the resulting FLAMFILE can be converted freely between ASCII and EBCDIC and vice versa at any time between compression and decompression. If the conversion is not handled by the data transmission system or on the data transmission path, the user may use the FLAM user exits and can apply the appropriate code conversions as part of the compression/decompression process.

FLAM always works in MODE=CX7 in respect to the code system of its host. If source and target are of the same code system (ASCII or EBCDIC) no conversion is necessary. If source and target are of different code systems, FLAM requires that the FLAMFILE is converted to the code system of the target host system first to decompression.

If the FLAMFILE in MODE=CX7 shall be transmitted via a 7-bit line as well as via an 8-bit line, it will be necessary to analyse the actual situation to maintain full compatibility. It must be considered, that FLAM offers integrated code conversion functions not on all platforms. However, this problem can be solved with MODE=CX7.

Because the FLAMFILE has records of equal length, the last record is filled with a padding character if fixed record format is used. In MODE=CX7 this padding character is blank, otherwise binary zero. If a variable record format is used, no padding is done for the last record - the last record is shortened instead.

Each record in the FLAMFILE has (internal) overhead: the FLAM syntax, which organizes a frame for the compressed data to fulfil the different requirements. The overhead is the same for each record: 4 byte in 7-bit format and 6 byte in 8-bit format. The user should consider this when he defines the record length for the FLAMFILE. The shorter the record length, the bigger the overhead. In addition the FLAMFILE contains the following syntactical elements: an optional file header for each original file, a compulsory block header for each matrix, etc.

Usually the FLAMFILE starts with a file header. This file header consists of a system independent on a system dependent part. The file header contains various information about the original file. During decompression FLAM will use this information - if not provided otherwise - to create the decompressed target file.

It is possible to concatenate multiple compressed files. In this case the FLAMFILE contains multiple file header. The FLAM utility ignores these file headers during decompression, and will use only the first one. However, the others appear in the protocol.

With this feature FLAM is prepared for the insertion of identical file headers into archive files, a feature that may help to identify a file even in the case of hardware faults.

If the FLAM record level interface is used, the different files can be separated during decompression.

An empty file is converted into a FLAMFILE that consists only out of a file header. This implies that an empty file no longer must be treated as a special case. The usual problems with empty files in the job command language and during file transfer belong to the past.

With a parameter it can be specified if and which file header shall be created during compression.

The decompression function can be used to only print out the file header without doing the actual decompression. This allows a quick information about the origin of the compressed file.

One block header is created per matrix. This block header contains all the necessary information for proper decompression even without a file header. However, in this case the user has to specify the target format via parameter, JCL or catalogue entry if another format than sequential and variable record length shall be created.

The block header contains also all information that is needed by the FLAM nucleus for decompression, e.g. MODE, version, matrix size, etc. This information guarantees the upward compatibility of FLAM.

The individual records of the FLAMFILE contain their length redundantly. If the FLAMFILE has variable record length an additional length field of 2- or 4-bytes length is part of each record. In MODE=CX7 on PC standard record separator characters of length 1 or 2 are used. Therefore the record length is not a physically unique size within a heterogeneous environment.

Each FLAMFILE record created in 8-bit code is protected against manipulations via a 16-bit checksum. In addition, so called block pointers allow synchronisation (and restart on block level), if data cannot be decompressed properly because of manipulation or hardware faults.

A FLAMFILE created in 7-bit code does not contain checksums, because this would inhibit code conversion from ASCII to EBCDIC and vice versa. Instead the number of bytes per record is checked. This function detects for example if a non 1:1 code conversion was applied. This may happen if printer control characters or tabulator characters are not converted 1:1. However, this does not comply with the preposition that the file must only contain printable characters.

It has a clear advantage to use the 8-bit format if it is not really necessary to work in 7-bit format.

Compression is faster, the compression ratio is better, the compressed file is better protected in respect to data security and data integrity and transmission of these files is faster in transparent mode. Also more possibilities for data encryption exist.

The reason is, that a 7-bit FLAMFILE can only be encrypted by randomising the character string. Other encryptions do not comply with the purpose for which these files were created (see above).

A *FLAMFILE* in 8-bit format, however, can be encrypted with all kinds of additional encryption methods to create a FLAMFILE not compatible with the market version.

For such conversions, the original state of the FLAMFILE as created by FLAM must be restored before the FLAMFILE is decompressed. In MODE=CX7 the FLAMFILE in addition must be converted into the code system of the host where the decompression is performed.

In the case that 1:1 code conversions shall be performed before compression or after decompression, FLAM offers the possibility to convert from EBCDIC to ASCII and vice versa as well from one EBCDIC dialect to another EBCDIC dialect. These conversions are implemented via code tables that can be replaced by the user. It is possible to use user defined code tables as well for encryption purposes. For all conversion problems not mentioned so far the user has the possibility to use the user exits for manipulation of uncompressed data. This is independent from the MODE parameter. The required conversions can be performed in connection with record processing.

Independently from the user exits, the record level interface provides access on record level before compression and after decompression. Using this possibility the user may process original data that otherwise could not be handled by FLAM. This interface also allows the integration of FLAM with user applications and software packages.

Also in the case when a FLAMFILE was created without a file header (HEADER=NO) FLAM can decompress this FLAMFILE.

Principally a restoration of a damaged FLAMFILE is possible but requires the consultation of an expert from the manufacturer. Such damages are caused exclusively by hardware or material faults or by unauthorised manipulation.

4.8.2 Group file

The ability to store several compressed files in one FLAMFILE has been realized in the form of the group file.

If several files are read during compression (see chapter 3.1.4), FLAM generates for each input file a file header (parameter HEADER=YES, default) in the FLAMFILE. A number of FLAMFILEs are written physically, in sequential order one after the other. (If the parameter HEADER=NO is set, no details about the respective file are stored in the group file. When decompressed, this file is then no longer recognized as a FLAMFILE containing a number of compressed files and it can only be decompressed altogether.)

The file type and format of a group file can be adapted to suit any requirements, exactly as in FLAMFILEs.

The parameter SHOW=DIR can be used to display the details of all of the compressed files in this group file, without the group file being decompressed.

FLAM is able to decompress each individual file of this group file by specifying a selection rule (see chapter 3.1.4.3). The decompressed file can be specified by means of a command or FLAM creates the file dynamically and catalogues it.

Libraries are compressed into a group file by FLAM on a member-by-member basis, i.e. it would be possible to decompress each member into a separate file using an appropriate conversion rule. Accordingly, a number of individual files can be used to generate library members.

This group file allows libraries that have been generated under a range of different operating systems to be exchanged between heterogeneous operating systems in a compatible way.

If neither a selection rule nor a conversion rule is specified, the compressed files are decompressed into a specified file as in earlier FLAM versions; i.e. all of the originally different files are now positioned one after the other, decompressed. Conversion is executed in accordance with the file attributes of the output.

Note: If FILEINFO=NO was set when the group file was being created, no file name was stored for the compressed data in question. This also means that there would be no file name available for creating the files.

Despite this, the compressed data can still be accessed and appropriate conversion rules compiled by means of the internal file names FILE0001 (for the 1st file) through FILE9999 (for the 9999th file).

4.9 Heterogeneous data exchange

Compressed files may be ported to a target system via file transfer or via tapes, cartridges, etc. It is not necessary that source system and target system are of the same type. However, necessary requirement is, that a file transfer feature exists or that a compatible tape or cartridge format exists.

If these requirements are fulfilled, data exchange is possible on both systems if FLAM exists and is installed.

All FLAM versions are upward compatible. That means, that systems with both FLAM older and newer version can compress/decompress in the old manner.

Since version 2.x the different formats of the compressed data are compatible on all systems where FLAM exists.

For data exchange between heterogeneous and homogeneous systems only logical data formats should be compressed with FLAM. Physical data formats cannot be reproduced identically on a physically different system.

Different methods for compression exist. With ADC, VR8 and CX8 data is compressed in 8-bit mode, with CX7 in 7-bit mode. For data exchange between mainframes any mode can be used therefore.

It is also necessary to check if the file transfer works transparently for 8-bit binary data. If yes, an 8-bit method (which can be decompressed on the target system) should be chosen for compression.

If file transfer is not transparent the CX7 mode must be selected. The file is only allowed to contain printable characters that can be converted 1:1 during the file transfer. Also in this case it should be checked if the selected compression mode is available on the target system.

For file transfer also transfer mode, record length and record format (variable or fixed) must be considered. It is possible that on the target system it is necessary to insert or delete length fields before decompression. Some file transfer products allow only certain record length and certain record formats.

One parameter that must be provided with the same value on both systems is the buffer size (MAXBUFFER) for the compression of one data block. This parameter has on mainframes the maximum value of 2.5 MB. FLAM uses on mainframes two alternating buffers, so double the memory is needed.

File attributes of the original file are meaningless for data exchange. The reason is that the file transmitted in sequential form is the FLAMFILE.

Within the target system the decompressed data can be stored in a file with any valid file organization. This may be

an organization that allows for sequential, index sequential or random access.

Important is, that the data must comply to this organization (e.g., a record key must be sorted in ascending order for index sequential organization).

Files can be compressed immediately during or after processing and can remain stored in compressed form until they are transmitted. Or they can be stored in uncompressed form and are compressed directly before transmission.

4.10 Code conversion

During compression and decompression any 1:1 code conversions can be applied to the original data.

For the conversion from EBCDIC to ASCII a standard table is provided. It is also possible to load a user defined table by specifying its name (TRANSLATE).

Generally spoken it is better to apply code conversion during decompression because the compression algorithm treats some frequent characters (like zero or blank) in a special way. Code conversion could reduce the compression effect. Also, because of the smaller character set of the ASCII code, it may happen that characters not contained in the ASCII code set are lost during code conversion, which cannot be reconstructed for decompression.

A delicate problem during the exchange of compressed data are index sequential files. Caused by the conversion the binary or alphanumeric keys may be out of order because of different collating sequences in the code systems. No problem, however, exists for keys that consist out of printable letters or for printable numeric keys.

A special conversion before or after processing with FLAM is necessary for index sequential files that contain binary and alphanumeric keys.

4.11 Transformation of file formats

Target files may be created during decompression with a file organization and record format differing from the original file.

This is especially true for compressed files received from another operating system.

If no other specifications are made by the user, all files that were compressed under the same operating system are reconstructed using the information in the file header.

In addition it is possible to convert the compressed file into any file format supported by FLAM on the target system.

However, dependencies between file organization and record format may exist:

If the file is transformed to fixed record length, the original records can be longer or shorter than the new record length.

Longer original records are truncated if parameter TRUNCATE=YES is specified.

Shorter original records are padded up to the new fix record length with blanks (PADCHAR).

If an index sequential file is transformed into a sequential file, the keys are removed if parameter KEYDISP=DEL is specified.

If a sequential file is transformed into an index sequential file, the original records must contain a field with key properties (unique and sorted in ascending order). Otherwise a printable key or arbitrary length can be inserted at the key position (KEYDISP=NEW).

Records of length 0 or gaps within relative files are removed if the file is converted into index sequential format.

If relative files are converted into sequential files, gaps are converted into records with length 0.

For a conversion into fixed format gaps are removed.

If files are converted into relative format, records of length 0 are represented as gaps, except if records of length 0 can be represented in the relative file format.

LDS files are managed on the disk by VSAM in units of 4096 bytes. If there is an "internal" format it is known only to the user and it is not used by VSAM.

In this regard, FLAM offers support not only for compression, but also for decompression.

Through the use of the parameters IRECSIZE, IBLKSIZE, and IDSORG=LDS for input and ORECSIZE, OBLKSIZE and ODSORG=LDS for output, it is possible to specify "internal" fixed record lengths with appropriate blocking. There is no requirement for the block size to be an exact multiple of the record length, any remainder will be ignored.

These specifications enable every file to be converted into an LDS format; that is, a considerably greater degree of efficiency can be achieved during compression.

FLAM (MVS)

User Manual

Chapter 5:

Application examples

Content

5. Application examples

In the following you find some examples to explain several FLAM functions. All examples are contained on the installation tape in form of command procedures or source code.

All examples were tested. However, it is possible that certain examples cannot be executed in a particular environment or that it is necessary to make some adaptations.

For the COBOL programs we have tried to stay as much independent from the operating system and the compiler as possible. The examples were therefore tested under BS2000 and DPPX as well as under MVS. During porting from MVS to BS2000 and DPPX some modifications must be made.

5.1 JCL

5.1.1 Compression

```
1 //USERCP JOB 7021000F,'LIMES-06172/5919-0',CLASS=A,
  //         MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=USER
  //*********************************************************
  //*          JOB FOR FLAM COMPRESSION
  //*********************************************************
2 //COMP     EXEC PGM=FLAM,PARM='C,SHOW (ALL)'
3 //STEPLIB  DD   DSN=USER.FLAM.LOAD,DISP=SHR
4 //FLAMFILE DD   DSN=USER.DAT.CMP,DISP=OLD
5 //FLAMIN   DD   DSN=USER.DAT.FB,DISP=SHR
6 //FLPRINT  DD   SYSOUT=*
7 //FLAMPAR  DD   *
    MAXB=4096     COMPRESSION BUFFER 4 KB
    MODE=VR8      COMPRESSION MODE
  /*
```

(1) Job card.

(2) The FLAM program is called for compression. All information shall be recorded in a protocol.

(3) Assignment of the library that contains all FLAM modules.

(4) Assignment of the FLAMFILE. In this example the FLAMFILE is already catalogued. So it is not necessary to make further specifications.

(5) Assignment of the input file for compression.

(6) Assignment of the protocol file. In this example we print the protocol directly via JES.

(7) Assignment of a parameter file. In this example the additional parameters are specified directly within the job stream. All specified parameters will overwrite the default parameters.

The resulting protocol is:

```
 1   FLM0448 COPYRIGHT (C) 1989-2005 BY LIMES DATENTECHNIK TEST 2006182
 2   FLM0428 RECEIVED: C,INFO(YES)
 3   FLM0410 DATA SET NAME : JES2.JOB01901.I0000101
     FLM0428 RECEIVED:  MAXB=4096
     FLM0428 RECEIVED:  MODE=VR8
 4   FLM0400 FLAM COMPRESSION VERSION 4.1A00 ACTIVE
 5   FLM0410 DATA SET NAME : USER.DAT.FB
     FLM0415 USED PARAMETER: ACCESS  : LOG
     FLM0415 USED PARAMETER: IDSORG  : SEQUENT
     FLM0415 USED PARAMETER: IRECFORM: FIXBLK
     FLM0415 USED PARAMETER: IRECSIZE:       80
     FLM0415 USED PARAMETER: IBLKSIZE:     3120
 6   FLM0410 DATA SET NAME : USER.DAT.CMP
     FLM0415 USED PARAMETER: MODE    : VR8
     FLM0415 USED PARAMETER: MAXBUFF :     4096
     FLM0415 USED PARAMETER: MAXREC  :      255
     FLM0415 USED PARAMETER: MAXSIZE :      512
     FLM0415 USED PARAMETER: DSORG   : SEQUENT
     FLM0415 USED PARAMETER: RECFORM : FIXBLK
     FLM0415 USED PARAMETER: BLKSIZE :     6144
 7   FLM0408 CPU - TIME:      0.0445
     FLM0409 RUN - TIME:      0.3382
 8   FLM0406 INPUT  RECORDS/BYTES:        155 /      12400
     FLM0407 OUTPUT RECORDS/BYTES:         10 /       5120
 9   FLM0416 COMPRESSION REDUCTION IN PERCENT:   58.71
10   FLM0440 FLAM COMPRESSION NORMAL END
```

(1) The copyright message also contains the licence number.
Here: Test licence with expiration date 182nd day in 2006.

(2) FLAM records the PARM= specification.

(3) The name of the parameter file is displayed. As parameters were given directly in the job stream the file name generated by JES is recorded. The file name is followed by the FLAM parameters contained in this file.

(4) The current FLAM version is recorded.

(5) The input file is recorded. Both the file name and the file attributes are displayed.

(6) The FLAMFILE is recorded. Both the file name and the file attributes are displayed. Also the compression parameters used are displayed.

(7) The CPU time used and the elapsed time is recorded.

(8) The number of records and bytes for both input and output is recorded.

(9) The compression ratio is recorded in percent.

(10) The compression was terminated normally.

5.1.2 Decompression

```
1 //USERDC JOB 7021000F,'LIMES-06172/5919-0',CLASS=A,
  //         MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=USER
  //*********************************************************
  //** JOB FOR DECOMPRESSION WITH FLAM
  //*********************************************************
2 //DECOMP   EXEC PGM=FLAM,PARM=DECO
3 //STEPLIB  DD   DSN=USER.FLAM.LOAD,DISP=SHR
4 //FLPRINT  DD   SYSOUT=*
5 //FLAMFILE DD   DSN=USER.DAT.CMP,DISP=SHR
6 //FLAMOUT  DD   DSN=USER.DAT.DEC,
  //              DISP=(NEW,CATLG,DELETE),
  //              SPACE=(TRK,(1,1)),
  //              UNIT=SYSDA
```

(1) Job card.

(2) FLAM call for decompression.

(3) Assignment of FLAM module library.

(4) Assignment of protocol file.

(5) Assignment of FLAMFILE.

(6) Assignment of output file. The output file is not catalogued (DISP=NEW). Record and block length are generated according to the original file (no DCB specification in JCL).

The corresponding protocol:

```
1      FLM0448 COPYRIGHT (C) 1989-2005 BY LIMES DATENTECHNIK TEST 2006182
2      FLM0428 RECEIVED: DECO
3      FLM0450 FLAMD VERSION 4.1A00 ACTIVE
4      FLM0460 DATA SET NAME: USER.DAT.CMP
       FLM0465 USED PARAMETER: MODE    : VR8
       FLM0465 USED PARAMETER: VERSION :      200
       FLM0465 USED PARAMETER: MAXBUFF :     4096
       FLM0465 USED PARAMETER: CODE    : EBCDIC
       FLM0465 USED PARAMETER: DSORG   : SEQUENT
       FLM0465 USED PARAMETER: RECFORM : FIXBLK
5      FLM0482 OLD ODSN    : USER.DAT.FB
       FLM0482 OLD ODSORG  : SEQUENT
       FLM0482 OLD ORECFORM: FIXBLK
       FLM0482 OLD ORECSIZE:       80
       FLM0482 OLD OBLKSIZE:     3120
6      FLM0469 COMPRESSED FILE FLAM-ID: 0101
7      FLM0460 DATA SET NAME : USER.DAT.DEC
       FLM0465 USED PARAMETER: ACCESS  : LOG
8      FLM0456 INPUT   RECORDS/BYTES:         10 /       5120
       FLM0457 OUTPUT  RECORDS/BYTES:        155 /      12400
9      FLM0458 CPU - TIME:      0.0456
       FLM0459 RUN - TIME:      0.1688
10     FLM0490 FLAM DECOMPRESSION NORMAL END
```

(1) The copyright message contains the licence number.
Here: test licence with expiration date at 182nd day in 2006.

(2) FLAM records the PARM= specification.

(3) The current FLAM version is recorded.

(4) The file name for the FLAMFILE is recorded. In the following the information contained in the FLAMFILE header is recorded: size of compression buffer, compression mode, coding of FLAM control characters and the file attributes of the FLAMFILE.

(5) Here all information about the original file is recorded as contained in the FLAMFILE header.

(6) This message informs that the FLAMFILE was created by an MVS system.

(7) The file name of the output file is recorded. Because no other file attributes are displayed, the file is created with the file attributes of the original file.

(8) The number of records and bytes for both input and output are recorded (always true data without record length fields).

(9) The CPU time used and the elapsed time are displayed.

(10) Decompression was terminated without error.

5.1.3 A more complex example

You'll see a more complex example for compression, showing the possibilities of FLAM.

All LIST data sets shall be compressed in ADC mode, encrypted with AES. The FLAMFILEs shall be limited at 1MB. We expect more than 9 files..

Cause the limitation of the PARM-entry (100 bytes) we need a file for all parameters.

```
1 //USERCP JOB 12345678,'LIMES-06172/59190',CLASS=A,
  //         MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=USER
  //*****************************************************
  //*          JOB FOR FLAM COMPRESSION
  //*****************************************************
2 //COMP     EXEC PGM=FLAM
3 //STEPLIB  DD   DSN=USER.FLAM.LOAD,DISP=SHR
4 //FLPRINT  DD   SYSOUT=*
5 //FLAMPAR  DD   *
      COMPRESS                  Start Compression
      MODE=ADC                  Mode 'Advanced Data Compression'
      FLAMIN=USER.*.LIST        Compress all LIST-Files
      FLAMFILE=USER.CMPLIST.ADC01   Name of FLAMFILE, 99 files possible
      SPLITMODE=SERIAL          Split FLAMFILE serially
      SPLITSIZE=1               at size of 1 MB
      CRYPTOMODE=AES            Use AES cipher mode for encryption
      SHOW=NO                   Protocol inactivated
      CRYPTOKEY=C'THIS IS A KEY FOR ENCRYPTION'
      SHOW=ALL                  Protocol activated
  /*
```

(1) Job card

(2) Call FLAM without any parameters

(3) Assignment of FLAM module library.

(4) Assignment of protocol.

(5) Assignment of the parameter file. The parameters are entered directly in the job stream. Cause the number characters '01' in the FLAMFILE name FLAM is able to create up to 99 files. The key-phrase starts with C', there are blanks in the parameter. The combination SHOW before and after the CRYPTOKEY parameter suppresses the protocol of the key!

The corresponding protocol:

```
1      FLM0448 COPYRIGHT (C) 1989-2003 BY LIMES DATENTECHNIK TEST 2003182
2      FLM0410 DATA SET NAME : USER.USERCP.JOB04480.D0000101.?  -PARFILE-
3      FLM0428 RECEIVED:  COMPRESS
       FLM0428 RECEIVED:  MODE=ADC
       FLM0428 RECEIVED:  FLAMIN=USER.*.LIST
       FLM0428 RECEIVED:  FLAMFILE=USER.CMPLIST.ADC01
       FLM0428 RECEIVED:  SPLITMODE=SERIAL
       FLM0428 RECEIVED:  SPLITSIZE=1
       FLM0428 RECEIVED:  CRYPTOMODE=AES
4      FLM0428 RECEIVED:  SHOW=ALL
5      FLM0400 FLAM COMPRESSION VERSION 4.0B00 ACTIVE
6      FLM0410 DATA SET NAME : USER.BIFLAMD.LIST  -FLAMIN-
       FLM0415 USED PARAMETER: IDSORG  : SEQUENT
       FLM0415 USED PARAMETER: IRECFORM: FIX
       FLM0415 USED PARAMETER: IRECSIZE:      133
       FLM0415 USED PARAMETER: IBLKSIZE:      133
       FLM0415 USED PARAMETER: IPRCNTRL: ASA
7      FLM0414 FLAMFILE SPLIT ACTIVE
8      FLM0415 USED PARAMETER: CRYPTO  : ACTIVE
9      FLM0410 DATA SET NAME : USER.CMPLIST.ADC01  -FLAMFILE-
10     FLM0415 USED PARAMETER: SPLITMOD: SERIAL
       FLM0415 USED PARAMETER: SPLITSIZ:        1
       FLM0415 USED PARAMETER: MODE    : ADC
       FLM0415 USED PARAMETER: CRYPTOMO: AES
       FLM0415 USED PARAMETER: MAXBUFF :    65536
       FLM0415 USED PARAMETER: MAXREC  :     4095
       FLM0415 USED PARAMETER: MAXSIZE :      512
       FLM0415 USED PARAMETER: DSORG   : SEQUENT
       FLM0415 USED PARAMETER: RECFORM : FIXBLK
       FLM0415 USED PARAMETER: BLKSIZE :    23040
11     FLM0435 MEMBER MAC: 23F747DB6705788A
12     FLM0406 INPUT   RECORDS/BYTES:          1,771 /             235,543
       FLM0407 OUTPUT  RECORDS/BYTES:             49 /              25,088
13     FLM0410 DATA SET NAME : USER.ASM01AC.LIST  -FLAMIN-
       FLM0415 USED PARAMETER: IDSORG  : SEQUENT
       FLM0415 USED PARAMETER: IRECFORM: FIX
       FLM0415 USED PARAMETER: IRECSIZE:      133
       FLM0415 USED PARAMETER: IBLKSIZE:      133
       FLM0415 USED PARAMETER: IPRCNTRL: ASA
11     FLM0435 MEMBER MAC: 5C94AB1028E5947A
       FLM0406 INPUT   RECORDS/BYTES:          2,136 /             284,088
       FLM0407 OUTPUT  RECORDS/BYTES:             59 /              30,208
       FLM0410 DATA SET NAME : USER.ASM02AC.LIST  -FLAMIN-
       FLM0415 USED PARAMETER: IDSORG  : SEQUENT
       FLM0415 USED PARAMETER: IRECFORM: FIX
       .
       .
       FLM0415 USED PARAMETER: IBLKSIZE:      133
       FLM0415 USED PARAMETER: IPRCNTRL: ASA
14     FLM0468 SPLIT   RECORDS/BYTES:          2,048 /           1,048,576
15     FLM0410 DATA SET NAME : USER.CMPLIST.ADC02  -F775020 -
       .
       .
       FLM0468 SPLIT   RECORDS/BYTES:          2,048 /           1,048,576
15     FLM0410 DATA SET NAME : USER.CMPLIST.ADC05  -F775020 -
```

```
      .
      .
      FLM0407 OUTPUT  RECORDS/BYTES:             10 /                5,120
16    FLM0468 SPLIT   RECORDS/BYTES:            451 /              230,912
17    FLM0410 DATA SET NAME : USER.CMPLIST.ADC01  -FLAMFILE-
18    FLM0435 FLAMFILE MAC: 50E22D8B48E0726B
19    FLM0406 INPUT   RECORDS/BYTES:        324,192 /           43,117,536
      FLM0407 OUTPUT  RECORDS/BYTES:          8,633 /            4,420,096
20    FLM0416 COMPRESSION REDUCTION IN PERCENT:    89.75
21    FLM0408 CPU - TIME:      16.6414
      FLM0409 RUN - TIME:      60.0083
22    FLM0440 FLAM COMPRESSION NORMAL END
```

(1) The copyright message contains the licence number.
Here:Test licence with expiration date at 182nd day in 2003.

(2) Recording the name of the parameter file. It is a JES generated file.

(3) Recording all parameter of this parameter file.

(4) The combination ‚SHOW=NONE ... SHOW=ALL' suppresses the recording of the encryption key.

(5) The current FLAM version is recorded.

(6) The first input file is recorded with its file attributes.

(7) The split mode has been activated.

(8) The cryptographic algorithm is used.

(9) Recording of the first FLAMFILE .

(10) And the corresponding file attributes.

(11) The member of the Group FLAMFILE is secured with the recorded Message Authorization Code (MAC). It is a unique value.

(12) The first input file is closed. Record- and byte counter are recorded as well as record- and byte counter of the compressed data.

(13) The second file is opened, recording its data.

(14) The first fragment of the FLAMFILE reached its limit. Recording the record- and byte counter for this file.

(15) The file name of the 2nd FLAMFILE fragment is shown and the corresponding (generated) DD-name.

(16) The last fragment has less records and bytes.

(17) Repeating the first FLAMFILE

(18) The entire Group FLAMFILE is secured with the recorded Message Authorization Code (MAC). It is a unique value for this FLAMFILE.

(19) The whole number of records and bytes for both input and output are recorded (always true data without record length fields).

(20) The compression ratio is recorded in percent.

(21) The CPU time used and the elapsed time are displayed.

(22) Compression was terminated without error.

To decompress all data with internal allocation by FLAM and change all output names from LIST to DEC only following information are necessary:

```
//DECO     EXEC PGM=FLAM
//STEPLIB  DD   DSN=USER.FLAM.LOAD,DISP=SHR
//FLPRINT  DD   SYSOUT=*
//FLAMPAR  DD   *
 DECOMPRESS               Start Decompression
 FLAMOUT=<*.LIST=*.DEC>   All files have DEC as last qualifier
 FLAMFILE=USER.CMPLIST.ADC01   Name of first FLAMFILE
 SHOW=NO                  Protocol inactivated
 CRYPTOKEY=C'THIS IS A KEY FOR ENCRYPTION'
 SHOW=ALL                 Protocol activated
/*
```

5.2 How to use the record level interface

5.2.1 Compression

The sequential file INDAT with fixed record length is read using a COBOL program. Each record is passed to the record level interface. FLAM generates the compressed FLAMFILE that will be read in the next example.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SAMPLE1C.
 AUTHOR.      LIMES DATENTECHNIK GMBH.
*
* SAMPLE1C READS A SEQUENTIAL DATA SET.
*          EVERY RECORD IS GIVEN TO FLAM FOR COMPRESSION.
*          FLAM MANAGES THE FLAMFILE ITSELF.
*
*          IN THIS EXAMPLE, THE FLAMFILE CAN BE
*             - ANY DATA SET   IN  MVS, BS2000
*             - VSAM               DOS/VSE
*
*          A SEQUENTIAL FILE IS READ.
*          EACH RECORD IS PASSED TO FLAM FOR
*          COMPRESSION.
*          FLAM HANDLES ITSELF THE COMPRESSED FILE.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 SPECIAL-NAMES.
     SYSOUT IS  OUT-PUT.
*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT  INDAT ASSIGN TO SYS010-S-DATAIN
             ACCESS MODE IS SEQUENTIAL
             ORGANIZATION IS SEQUENTIAL.
*
 DATA DIVISION.
*
 FILE SECTION.
 FD  INDAT   RECORD CONTAINS 80 CHARACTERS
             RECORDING MODE IS F.
*
 01  INDAT-RECORD.
     02  FILLER    PIC X(80).
*
 WORKING-STORAGE SECTION.
*
 77  OPERATION       PIC X(6).
*
```

```
 01  FLAM-PARAMETER.
*
*  USED FOR EVERY FLAM CALL
*
     02  FILE-ID   PIC S9(8)  COMP SYNC.
     02  RETCO     PIC S9(8)  COMP SYNC.
        88 FLAMOK  VALUE  0.
*
     02  RETCO-X   REDEFINES RETCO.
       03 RETCO-1   PIC X.
        88 NODMS-ERROR   VALUE  LOW-VALUE.
       03 RETCO-2-4 PIC XXX.
*
*  USED FOR FLAM OPEN
*
     02  LASTPAR   PIC S9(8)  COMP SYNC VALUE 0.
     02  OPENMODE  PIC S9(8)  COMP SYNC VALUE 1.
     02  DDNAME    PIC X(8)   VALUE "FLAMFILE".
     02  STATIS    PIC S9(8)  COMP SYNC VALUE 0.
*
*  USED FOR FLAM PUT
*
     02  DATLEN    PIC S9(8)  COMP SYNC VALUE +80.
     02  DATABYTES PIC X(80).
/
 PROCEDURE DIVISION.
 MAIN SECTION.
*
 OPEN-INPUT-DATA.
*
*  OPEN DATA SET TO READ RECORDS
*
     OPEN INPUT INDAT.
*
 OPEN-FLAM.
*
*    OPEN FLAM FOR OUTPUT (COMPRESSION)
*
     CALL  "FLMOPN" USING FILE-ID, RETCO,
                          LASTPAR, OPENMODE, DDNAME, STATIS.
     IF  NOT FLAMOK
        THEN  MOVE "OPEN" TO OPERATION
              PERFORM FLAM-ERROR
              GO TO CLOSE-DATA.
 READ-RECORD.
*
*  READ A RECORD FROM INPUT DATA SET
*
    READ INDAT INTO DATABYTES  AT END
                                GO TO FINISH-COMPRESSION.
*
 WRITE-RECORD.
*
*  WRITE THE RECORD WITH FLAM COMPRESSION
*
     CALL "FLMPUT" USING FILE-ID, RETCO,
                         DATLEN, DATABYTES.
*
```

```
     IF  FLAMOK
       THEN  GO TO READ-RECORD
       ELSE  MOVE "PUT" TO  OPERATION
             PERFORM FLAM-ERROR.
*
 FINISH-COMPRESSION.
*
*  CLOSE FLAM
*
     CALL "FLMCLS" USING FILE-ID, RETCO.
     IF  NOT FLAMOK
       THEN MOVE "CLOSE" TO OPERATION
            PERFORM FLAM-ERROR.
 CLOSE-DATA.
     CLOSE  INDAT.
 MAIN-END.
     STOP RUN.
*
 FLAM-ERROR SECTION.
 FLAM-ERROR-1.
     IF   NODMS-ERROR
       THEN   DISPLAY "FLAM-ERROR." UPON OUT-PUT
       ELSE   MOVE LOW-VALUE TO RETCO-1
              DISPLAY "DMS-ERROR FOR FLAMFILE." UPON OUT-PUT.
     DISPLAY "OPERATION " OPERATION "RETURNCODE= " RETCO
             UPON OUT-PUT.
 FLAM-ERROR-99.
      EXIT.
```

5.2.2 Decompression

FLAM reads here the compressed file from the last example. The decompressed records are passed via the record level interface to the COBOL program and are finally written into file OUTDAT.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SAMPLE1D.
 AUTHOR.      LIMES DATENTECHNIK GMBH.
*
*  SAMPLE1D READS WITH FLAM COMPRESSED RECORDS AND WRITES
*           THE RECEIVED DECOMPRESSED DATA IN A SEQUENTIAL
*           DATA SET.
*
*           IN THIS EXAMPLE, THE FLAMFILE CAN BE
*              - ANY DATA SET     IN  MVS, BS2000
*              - VSAM             IN DOS/VSE
*
*           HERE, FLAM IS USED FOR READ ACCESSES TO
*           COMPRESSED DATA.
*           RECORDS RECEIVED ARE WRITTEN TO A
*           SEQUENTIAL FILE.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
*
 SPECIAL-NAMES.
     SYSOUT IS  OUT-PUT.
*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT      OUTDAT
     ASSIGN TO SYS010-S-DATOUT
     ACCESS MODE IS SEQUENTIAL.
*
*       DATA DIVISION.
*
 FILE SECTION.
 FD  OUTDAT  RECORD CONTAINS 80 CHARACTERS
             RECORDING MODE F.
 01  OUTDAT-RECORD.
     02 FILLER   PIC X(80).
*
 WORKING-STORAGE SECTION.
*
 77  OPERATION   PIC  X(6).
*
 01  FLAM-PARAMETER.
*
*  USED FOR ALL FLAM CALLS
*
     02 FILE-ID  PIC S9(8)  COMP SYNC.
     02 RETCO    PIC S9(8)  COMP SYNC.
        88 FLAMOK            VALUE  0.
        88 FILEID-ERR        VALUE -1.
        88 MEMORY-ERR        VALUE -1.
        88 REC-TRUNCATED     VALUE  1.
```

```
         88 END-OF-FILE       VALUE  2.
         88 REC-NOT-FOUND     VALUE  5.
         88 NEW-HEADER        VALUE  6.
*
         88 NO-FLAMFILE       VALUE 10.
         88 FORMAT-ERR        VALUE 11.
         88 RECLEN-ERR        VALUE 12.
         88 FILELEN-ERR       VALUE 13.
         88 CHECKSUM-ERR      VALUE 14.
         88 MAXB-INVALID      VALUE 21.
         88 COMPMODE-INVALID  VALUE 22.
         88 COMPSYNTAX-ERR    VALUE 23.
         88 BLKSIZE-INVALID   VALUE 24.
         88 RECSIZE-INVALID   VALUE 25.
         88 FLAMCODE-INVALID  VALUE 26.
         88 FILE-EMPTY        VALUE 30.
         88 NO-DATA-SET       VALUE 31.
*
      02  RETCO-X   REDEFINES RETCO.
        03 RETCO-1   PIC X
          88 FLAM-ERROR-RC VALUE  LOW-VALUE.
        03 RETCO-2-4 PIC XXX.
*
*  USED FOR FLAM OPEN
*
      02  LASTPAR   PIC S9(8)  COMP SYNC VALUE 0.
      02  OPENMODE  PIC S9(8)  COMP SYNC VALUE 0.
      02  DDNAME    PIC X(8)   VALUE "FLAMFILE".
      02  STATIS    PIC S9(8)  COMP SYNC VALUE 0.
*
*  USED FOR FLAM GET
*
      02  DATLEN    PIC S9(8).
      02  MAXLEN    PIC S9(8)  COMP SYNC VALUE +80.
/
 PROCEDURE DIVISION.
*
 MAIN SECTION.
*
 OPEN-OUTPUT-DATA.
*
*  OPEN DATA SET TO WRITE RECORDS
*
      OPEN OUTPUT OUTDAT.
*
 OPEN-FLAM.
*
*  OPEN FLAM FOR INPUT (DECOMPRESSION)
*
      CALL  "FLMOPN" USING FILE-ID, RETCO,
                           LASTPAR, OPENMODE, DDNAME, STATIS.
      IF  NOT FLAMOK
         THEN  MOVE "OPEN" TO OPERATION
               PERFORM FLAM-ERROR
               GO TO CLOSE-DATA.
 READ-RECORD.
*
*  READ A RECORD WITH FLAM IN OUTPUT AREA
```

```
*
     CALL "FLMGET" USING FILE-ID, RETCO,
                         DATLEN, OUTDAT-RECORD, MAXLEN.
*
     IF  FLAMOK
       THEN  NEXT SENTENCE
       ELSE  IF  END-OF-FILE
                 THEN GO TO CLOSE-FLAM
                 ELSE MOVE "GET" TO OPERATION
                      PERFORM FLAM-ERROR
                      GO TO CLOSE-FLAM.
*
 WRITE-RECORD.
*
*  WRITE THE DECOMPRESSED RECORD
*
     WRITE OUTDAT-RECORD.
*
     GO TO READ-RECORD.
*
 CLOSE-FLAM.
*
*  CLOSE TO FLAM
*
     CALL "FLMCLS" USING FILE-ID, RETCO.
     IF  NOT FLAMOK
       THEN MOVE "CLOSE" TO OPERATION
            PERFORM FLAM-ERROR.
 CLOSE-DATA.
*
*  CLOSE OUTPUT DATA
*
     CLOSE  OUTDAT.
 MAIN-END.
     STOP RUN.
*
 FLAM-ERROR SECTION.
 FLAM-ERROR-1.
     IF   FLAM-ERROR-RC
       THEN   DISPLAY "FLAM-ERROR." UPON OUT-PUT
       ELSE   MOVE LOW-VALUE TO RETCO-1
              DISPLAY "DMS-ERROR FOR FLAMFILE." UPON OUT-PUT.
     DISPLAY "OPERATION " OPERATION "RETURNCODE= " RETCO
             UPON OUT-PUT.
 FLAM-ERROR-99.
     EXIT.
```

5.2.3 Random access to an index sequential FLAMFILE

This example requires an index sequential FLAMFILE created from an index sequential original file with 80 bytes record length and a key of 8 byte length at position 73. The keys are printable numeric from 1 to n. We assume n to be greater than 40. The compressed file can be created using the FLAM utility.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SAMPLE3D.
 AUTHOR.      LIMES DATENTECHNIK GMBH.
*
*  SAMPLE3D IS AN EXAMPLE FOR AN INFORMATION RETRIEVAL PROGRAM,
*  BASED ON A VSAM-KSDS-FLAMFILE, USING THE FLAM-CALL-INTERFACE
*
*  A DIRECT READ WITH KEY IS DONE.
*  IF RECORD FOUND, THE NEXT RECORDS ARE READ SEQUENTIAL AND
*  DISPLAYED, UNTIL A NEW SET OF KEYS START.
*
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
*
 SPECIAL-NAMES.
*
     SYSOUT IS OUT-PUT.
*
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
*
 77  NEXT-KEY        PIC  9(8).
*
 77  CONDITION-FLAG  PIC  X.
     88 SET-END      VALUE "X".
*
 77  SET-END-FLAG    PIC  X     VALUE "X".
*
 01  FLAM-FILEID     PIC  9(8)  COMP.
*
 01  FLAM-RETCO      PIC S9(8)  COMP.
     88 FLAMOK            VALUE  0.
     88 FILEID-ERR        VALUE -1.
     88 MEMORY-ERR        VALUE -1.
     88 REC-TRUNCATED     VALUE  1.
     88 END-OF-FILE       VALUE  2.
     88 REC-NOT-FOUND     VALUE  5.
     88 NEW-HEADER        VALUE  6.
*
     88 NO-FLAMFILE       VALUE 10.
     88 FORMAT-ERR        VALUE 11.
     88 RECLEN-ERR        VALUE 12.
     88 FILELEN-ERR       VALUE 13.
     88 CHECKSUM-ERR      VALUE 14.
     88 MAXB-INVALID      VALUE 21.
```

```
           88 COMPMODE-INVALID  VALUE 22.
           88 COMPSYNTAX-ERR    VALUE 23.
           88 BLKSIZE-INVALID   VALUE 24.
           88 RECSIZE-INVALID   VALUE 25.
           88 FLAMCODE-INVALID  VALUE 26.
           88 FILE-EMPTY        VALUE 30.
      *
       01  RETCO-X REDEFINES FLAM-RETCO.
           03 RETCO-1   PIC X.
              88 NODMS-ERROR    VALUE  LOW-VALUE.
           03 RETCO-2   PIC X.
           03 RETCO-3-4.
            05 RETCO-3  PIC X.
            05 RETCO-4  PIC X.
      ************************************************
      *
       01  FLMOPN-AREA.
           02  LASTPAR   PIC S9(8)  COMP SYNC VALUE 0.
           02  OPENMODE  PIC S9(8)  COMP SYNC VALUE 0.
           02  DDNAME    PIC X(8)   VALUE "FLAMFILE".
           02  STATIS    PIC S9(8)  COMP SYNC VALUE 0.
      *
       01  FLMGET-FLMGKY-AREA.
           02 DATALEN      PIC S9(8)  COMP SYNC.
           02 DATA-AREA.
              04 PURE-DATA PIC X(72).
              04 KEY-DATA  PIC 9(8).
           02 BUFFLEN      PIC S9(8)  COMP SYNC VALUE +80.
      *
       01  SEARCH-KEYS.
           02 S-KEY-1      PIC  9(8)  VALUE  10.
           02 S-KEY-2      PIC  9(8)  VALUE  30.
           02 S-KEY-3      PIC  9(8)  VALUE  0.
       01  STOP-KEYS.
           02 STOP-KEY-1   PIC  9(8)  VALUE  20.
           02 STOP-KEY-2   PIC  9(8)  VALUE  40.
           02 STOP-KEY-3   PIC  9(8)  VALUE  9.
      /
       PROCEDURE DIVISION.
      *
       MAIN SECTION.
       MAIN-OPEN-FILE.
      *
      *    OPEN FLAMFILE
      *
      *    THE FLAMFILE WAS BUILD BY THE FLAM UTILITY, SO IT HAS
      *    A FILE HEADER CONTAINING VALUES ABOUT THE ORIGINAL DATA SET.
      *    THEN WE NEED ONLY THE FLMOPN CALL.
      *
           CALL "FLMOPN" USING  FLAM-FILEID,
                                FLAM-RETCO,
                                LASTPAR,
                                OPENMODE,
                                DDNAME,
                                STATIS.
           IF  NOT FLAMOK
              THEN  DISPLAY "OPEN-ERROR." UPON OUT-PUT
                    PERFORM FLAM-ERROR
```

```
              GO TO MAIN-END.
 MAIN-SEARCH-1.
*
*  SEARCH FOR SPECIAL RECORD WITH KEY NO. 1
*
     MOVE  S-KEY-1        TO    KEY-DATA.
     PERFORM  GET-KEY.
*
*  IF RECORD FOUND, READ THE NEXT RECORDS
*
     IF  FLAMOK
        THEN MOVE  STOP-KEY-1  TO  NEXT-KEY
             MOVE  SPACE       TO  CONDITION-FLAG
             PERFORM GET-SEQ UNTIL SET-END.
 MAIN-SEARCH-2.
*
*  SEARCH FOR SPECIAL RECORD WITH KEY NO. 2
*
     MOVE  S-KEY-2        TO    KEY-DATA.
     PERFORM  GET-KEY.
*
*  IF RECORD FOUND, READ THE NEXT RECORDS
*
     IF  FLAMOK
        THEN MOVE  STOP-KEY-2  TO  NEXT-KEY
             MOVE  SPACE       TO  CONDITION-FLAG
             PERFORM GET-SEQ UNTIL SET-END.
 MAIN-SEARCH-3.
*
*  SEARCH FOR SPECIAL RECORD WITH KEY NO. 3
*  (KEY DOES NOT EXIST IN DATA SET).
*
     MOVE  S-KEY-3        TO    KEY-DATA.
     PERFORM  GET-KEY.
*
*  IF RECORD NOT FOUND, FLAM POSITIONS TO THE NEXT HIGHER KEY
*  IN THE DATA SET:
*
     IF  REC-NOT-FOUND
        THEN MOVE  STOP-KEY-3  TO  NEXT-KEY
             MOVE  SPACE       TO  CONDITION-FLAG
             PERFORM GET-SEQ UNTIL SET-END.
 MAIN-CLOSE-FILE.
*
*  CLOSE FLAMFILE
*           CALL "FLMCLS" USING  FLAM-FILEID,
                                 FLAM-RETCO.
 MAIN-END.
     STOP RUN.
/
 FLAM-ERROR SECTION.
*
*  FLAM RETURN CODE IS NOT ZERO.
*  DOCUMENT THE ERROR-SITUATION.
*
 FLAM-ERROR-1.
     IF   END-OF-FILE
       THEN   GO TO FLAM-ERROR-99.
```

```
     IF   NODMS-ERROR
       THEN   DISPLAY "FLAM-ERROR." UPON OUT-PUT
       ELSE   MOVE LOW-VALUE TO RETCO-1
*             THIS BYTE CONTAINS A SIGN FOR DATA SET-ERROR,
*             WE DON"T NEED TO DISPLAY IT
              DISPLAY "DMS-ERROR FOR FLAMFILE." UPON OUT-PUT.
 FLAM-ERROR-2.
     DISPLAY "RETURNCODE= " FLAM-RETCO UPON OUT-PUT.
 FLAM-ERROR-99.
     EXIT.
/
 GET-KEY SECTION.
*
*  GET A RECORD WITH SPECIFIED KEY
*
 GET-KEY-1.
     CALL "FLMGKY" USING   FLAM-FILEID,
                           FLAM-RETCO,
                           DATALEN,
                           DATA-AREA,
                           BUFFLEN.
 GET-KEY-2.
     IF  FLAMOK
        THEN  NEXT SENTENCE
        ELSE  IF   REC-NOT-FOUND
                 THEN  DISPLAY "KEY NOT FOUND:    "  KEY-DATA
                               UPON OUT-PUT
                       GO TO GET-KEY-99
                 ELSE  PERFORM FLAM-ERROR
                       GO TO GET-KEY-99.
 GET-KEY-3.
     DISPLAY "KEY FOUND: " KEY-DATA UPON OUT-PUT.
     DISPLAY "DATA: "               UPON OUT-PUT.
     DISPLAY DATA-AREA              UPON OUT-PUT.
 GET-KEY-99.
     EXIT.
/
 GET-SEQ SECTION.
*
*  GET RECORDS IN SEQUENTIAL ORDER
*
 GET-SEQ-1.
     CALL "FLMGET" USING   FLAM-FILEID,
                           FLAM-RETCO,
                           DATALEN,
                           DATA-AREA,
                           BUFFLEN.
 GET-SEQ-2.
*
*  CHECK RETURNCODE
*
     IF   FLAMOK
       THEN
*
*      IF RECORD CONTAINS TO THE SET, DISPLAY THE DATA,
*      ELSE SET THE SET-END CONDITION.
*
         IF  KEY-DATA    NEXT-KEY
```

```
           THEN  DISPLAY  DATA-AREA  UPON OUT-PUT
           ELSE  MOVE SET-END-FLAG TO CONDITION-FLAG
       ELSE
*
*      SET THE SET-END CONDITION,
*      ON ERROR, DISPLAY THE FLAM RETURN CODE.
*
           MOVE  SET-END-FLAG  TO    CONDITION-FLAG
           IF   NOT END-OF-FILE
             THEN  PERFORM FLAM-ERROR.
 GET-SEQ-99.
     EXIT.
```

5.2.4 Example for the entire record level interface FLAMREC

In this program, you are able to call all functions of the record interface FLAMREC with all parameters and in any sequence. This example thus contains all the file definitions and all the subprogram calls that can be used for the record level interface. It can be used as an example not only for development, but also for examining any compressed file.

You'll find the code in the example library FLAM.SRCLIB.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. RECTEST.
*****************************************************************
*  NAME:     RECTEST                                            *
*                                                               *
*  FUNKTION: FLAMREC-SCHNITTSTELLE BENUTZEN.                    *
*            MIT DIESEM TESTPROGRAMM KOENNEN ALLE FUNKTIONEN    *
*            DER FLAM SATZSCHNITTSTELLE FLAMREC MIT ALLEN PARA- *
*            METERWERTEN IN BELIEBIGER REIHENFOLGE AUFGERUFEN   *
*            WERDEN.                                            *
*                                                               *
*  FUNCTION: USE THE INTERFACE FLAMREC.                         *
*            IN THIS EXAMPLE YOU ARE ABLE TO CALL ALL FUNCTIONS *
*            OF THE RECORD INTERFACE FLAMREC WITH ALL PARAMETERS*
*            AND IN ANY SEQUENCE.                               *
*                                                               *
*                                                               *
*                                                               *
*  IT IS AN EXAMPLE FOR CALLING FLAM FROM A COBOL ROUTINE.      *
*  EVERY ENTRY AND ITS PARAMETER ARE DESCRIBED.                 *
*                                                               *
*  TO START RECTEST IN TSO, USE AFTER COMPILATION AND LINKING:  *
*                                                               *
*    ALLOC DSN(*) DD(SYSIN)                                     *
*    ALLOC DSN(*) DD(SYSOUT)                                    *
*    CALL 'USER.FLAM.LOAD(RECTEST)'                             *
*                                                               *
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
*
     SYSIN    IS TERMIN
     SYSOUT   IS TERMOUT.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
*    PARAMETER FOR FLMOPN
*
 77  FLAMID                    PIC S9(8) COMP SYNC.
 01  RETCO                     PIC S9(8) COMP SYNC.
     88  OK                        VALUE 0.
     88  INVALID                   VALUE -1.
```

```
 01  RETCO-RED REDEFINES RETCO.
     05  RETCO-INDICATOR        PIC X(1).
         88  DVS-ERROR              VALUE HIGH-VALUE.
     05  FILLER                 PIC X(1).
     05  RETCO-FLAM             PIC S9(4) COMP SYNC.
         88  CUT                    VALUE 1.
         88  EOF                    VALUE 2.
         88  GAP                    VALUE 3.
         88  INVKEY                 VALUE 5.
 77  LASTPAR                    PIC S9(8) COMP SYNC
                                    VALUE 1.
     88  LAST-PARAMETER             VALUE 0.
 77  OPENMODE                   PIC S9(8) COMP SYNC
                                    VALUE 2.
     88  OPEN-INPUT                 VALUE 0.
     88  OPEN-OUTPUT                VALUE 1.
     88  OPEN-INOUT                 VALUE 2.
     88  OPEN-OUTIN                 VALUE 3.
 77  DDNAME                     PIC X(8)
                                    VALUE "FLAMFILE".
 77  STATIS                     PIC S9(8) COMP SYNC
                                    VALUE 1.
     88  STATISTICS                 VALUE 1.
*
*    PARAMETER FOR FLMOPD
*
 77  NAMELEN                    PIC S9(8) COMP SYNC
                                    VALUE 54.
 77  FILENAME                   PIC X(54)
                                    VALUE SPACES.
 77  DSORG                      PIC S9(8) COMP SYNC
                                    VALUE 1.
 77  RECFORM                    PIC S9(8) COMP SYNC.
 77  MAXSIZE                    PIC S9(8) COMP SYNC
                                    VALUE 512.
 77  RECDELIM                   PIC X(4).
 77  BLKSIZE                    PIC S9(8) COMP SYNC.
 77  CLOSDISP                   PIC S9(8) COMP SYNC
                                    VALUE 0.
 77  DEVICE                     PIC S9(8) COMP SYNC
                                    VALUE 0.
*
*    PARAMETER FOR FLMOPF
*
 77  VERSION                    PIC S9(8) COMP SYNC.
     88  VERSION-1                  VALUE 100.
     88  VERSION-1-1                VALUE 101.
     88  VERSION-2                  VALUE 200.
 77  FLAMCODE                   PIC S9(8) COMP SYNC.
     88  EBC-DIC                    VALUE 0.
     88  ASCII                      VALUE 1.
 77  COMPMODE                   PIC S9(8) COMP SYNC.
     88  CX8                        VALUE 0.
     88  CX7                        VALUE 1.
     88  VR8                        VALUE 2.
 77  MAXBUFF                    PIC S9(8) COMP SYNC.
 77  HEADER                     PIC S9(8) COMP SYNC
                                    VALUE 1.
```

```
     88  NOHEADER                    VALUE 0.
     88  FILEHEADER                  VALUE 1.
 77  MAXREC                      PIC S9(8) COMP SYNC
                                     VALUE 255.
*
*    SCHLUESSELBESCHREIBUNG DER FLAMFILE
*    KEY DESCRIPTION OF THE FLAMFILE
*
 01  KEYDESC.
   05  KEYFLAGS                 PIC S9(8) COMP SYNC
                                     VALUE 1.
   05  KEYPARTS                 PIC S9(8) COMP SYNC
                                     VALUE 1.
   05  KEYENTRY1.
     10  KEYPOS1                PIC S9(8) COMP SYNC
                                     VALUE 1.
     10  KEYLEN1                PIC S9(8) COMP SYNC
                                     VALUE 9.
     10  KEYTYPE1               PIC S9(8) COMP SYNC
                                     VALUE 1.
   05  KEYENTRY-2-BIS-8               OCCURS 7 TIMES.
     10  KEYPOS                 PIC S9(8) COMP SYNC.
     10  KEYLEN                 PIC S9(8) COMP SYNC.
     10  KEYTYPE                PIC S9(8) COMP SYNC.
*
 77  BLKMODE                    PIC S9(8) COMP SYNC.
     88  UNBLOCKED                   VALUE 0.
     88  BLOCKED                     VALUE 1.
 77  EXK20                      PIC X(8)
                                     VALUE SPACES.
 77  EXD20                      PIC X(8)
                                     VALUE SPACES.
*
*    PARAMETER FOR FLMPHD
*
 77  NAMELEN-ORIG               PIC S9(8) COMP SYNC
                                     VALUE 54.
 77  FILENAME-ORIG              PIC X(54)
                                     VALUE SPACES.
 77  DSORG-ORIG                 PIC S9(8) COMP SYNC
                                     VALUE 1.
 77  RECFORM-ORIG               PIC S9(8) COMP SYNC.
 77  RECSIZE-ORIG               PIC S9(8) COMP SYNC
                                     VALUE 512.
 77  RECDELIM-ORIG              PIC X(4).
 77  BLKSIZE-ORIG               PIC S9(8) COMP SYNC.
 77  PRCTRL-ORIG                PIC S9(8) COMP SYNC
                                     VALUE 0.
     88  NO-CONTROL-CHAR             VALUE 0.
     88  ASA-CONTROL-CHAR            VALUE 1.
     88  MACH-CONTROL-CHAR           VALUE 2.
 77  SYSTEM-ORIG                PIC X(2)
                                     VALUE LOW-VALUES.
 77  LASTPAR-PHD                PIC S9(8) COMP SYNC
                                     VALUE 1.
     88  LAST-PARAMETER-PHD          VALUE 0.
*
*    SCHLUESSELBESCHREIBUNG DER ORIGINALDATEI
```

```
*    KEY DESCRIPTION OF THE ORIGINAL DATA SET
*
 01  KEYDESC-ORIG.
   05  KEYFLAGS-ORIG            PIC S9(8) COMP SYNC
                                    VALUE 1.
   05  KEYPARTS-ORIG            PIC S9(8) COMP SYNC
                                    VALUE 1.
   05  KEYENTRY1-ORIG.
     10  KEYPOS1-ORIG           PIC S9(8) COMP SYNC
                                    VALUE 1.
     10  KEYLEN1-ORIG           PIC S9(8) COMP SYNC
                                    VALUE 8.
     10  KEYTYPE1-ORIG          PIC S9(8) COMP SYNC
                                    VALUE 1.
   05  KEYENTRY-2-TO-8-ORIG         OCCURS 7 TIMES
                                    INDEXED BY KEYDESC-INDEX.
     10  KEYPOS-ORIG            PIC S9(8) COMP SYNC.
     10  KEYLEN-ORIG            PIC S9(8) COMP SYNC.
     10  KEYTYPE-ORIG           PIC S9(8) COMP SYNC.
*
 77  KEYDESC-INDICATOR          PIC X(1)
                                    VALUE "Y".
     88  KEYDESC-DEFINES            VALUE "Y".
*
*    PARAMETER FOR FLMPUH
*
 77  UATTRLEN                   PIC S9(8) COMP SYNC.
 77  USERATTR                   PIC X(80).
*
*    PARAMETER FLMGET / FLMPUT
*
 77  RECLEN                     PIC S9(8) COMP SYNC
                                          VALUE 80.
 01  REC-ORD.
   05  BYTE                     PIC X(1)
                                    OCCURS  32767 TIMES
                                    INDEXED BY REC-INDEX.
 01  RECORD-DISPLAY REDEFINES REC-ORD
                                PIC X(80).
 01  RECORD-KEY-DISPLAY.
     02 RECORD-KEY-BYTE         PIC X(1)   OCCURS 80
                                    INDEXED BY KEY-INDEX.
 77  BUFLEN                     PIC S9(8) COMP SYNC
                                    VALUE 32767.
 77  CHECKMODE                  PIC S9(8) COMP SYNC
                                    VALUE 0.
 77  RECNO                      PIC S9(8) COMP SYNC
                                    VALUE 0.
*
*    VARIABLES TO EDIT THE RETURN CODE
*
 77  LEN-RETCO                  PIC S9(8) COMP SYNC
                                    VALUE 4.
 01  RETCO-HEX.
     05  FILLER                 PIC X(4).
     05  RETCO-DISP             PIC X(4).
*
*    VARIABLEN ZUM EINLESEN UND AUFBEREITEN VON ZAHLEN
```

```
*    VARIABLES TO READ IN, AND EDIT NUMBERS
*
 01  INPUT.
     05  BYTE-INP               PIC X(1)
                                OCCURS 9 TIMES
                                INDEXED BY INP-INDEX.
 01  INPUT-NUM                PIC S9(8).
 01  EINGABE-RED REDEFINES INPUT-NUM.
     05  BYTE-RED               PIC X(1)
                                OCCURS 8 TIMES
                                INDEXED BY RED-INDEX.
*
*  AUSGEWAEHLTE FUNKTION
*  SELECTED FUNCTION
*
 01  FUNCTION                   PIC X(8).
     88  FLMOPD                    VALUES "FLMOPD" "OPD".
     88  FLMOPF                    VALUES "FLMOPF" "OPF".
     88  FLMCLS                    VALUES "FLMCLS" "CLS" "C".
     88  FLMFLU                    VALUES "FLMFLU" "FLU".
     88  FLMGET                    VALUES "FLMGET" "GET" "G".
     88  FLMGTR                    VALUES "FLMGTR" "GTR".
     88  FLMGKY                    VALUES "FLMGKY" "GKY".
     88  FLMFKY                    VALUES "FLMFKY" "FKY".
     88  FLMGRN                    VALUES "FLMGRN" "GRN".
     88  FLMFRN                    VALUES "FLMFRN" "FRN".
     88  FLMPUT                    VALUES "FLMPUT" "PUT" "P".
     88  FLMPKY                    VALUES "FLMPKY" "PKY".
     88  FLMPOS                    VALUES "FLMPOS" "POS".
     88  FLMDEL                    VALUES "FLMDEL" "DEL" "D".
     88  FLMUPD                    VALUES "FLMUPD" "UPD" "U".
     88  FLMPHD                    VALUES "FLMPHD" "PHD".
     88  FLMPUH                    VALUES "FLMPUH" "PUH".
     88  FLMGHD                    VALUES "FLMGHD" "GHD".
     88  FLMGUH                    VALUES "FLMGUH" "GUH".
*
*  AREAS FOR FLMCLS AND FLMFLU
*
 77  CPUTIME                    PIC 9(8) COMP.
 77  REC-ORDS                   PIC 9(8) COMP.
 01  BYTEFIELD.
     05  BYTEOFL                PIC 9(8) COMP SYNC.
     05  BYTES                  PIC 9(8) COMP SYNC.
 01  BYTECNT REDEFINES BYTEFIELD PIC S9(18) COMP SYNC.
 77  CMPRECS                    PIC 9(8) COMP.
 01  CMPBYFIELD.
     05  CMPBYOFL               PIC 9(8) COMP SYNC.
     05  CMPBYTES               PIC 9(8) COMP SYNC.
 01  CMPBYCNT REDEFINES CMPBYFIELD
                                PIC S9(18) COMP SYNC.
*
 77  STATIS-DIS                 PIC ZZZ,ZZZ,ZZZ,ZZZ,ZZZ,ZZ9.
*
*  ARBEITSVARIABLEN
*  WORK FIELDS
*
 77  INDEX-DISPLAY              PIC 9(8).
 77  KEY-IND-DISP               PIC S9(8) COMP.
```

```
 77  GET-COUNT                  PIC 9(8).
 77  GET-INDEX                 PIC S9(8) COMP SYNC.
 77  REL-POSITION              PIC S9(8) COMP SYNC.
     88  FILE-END                  VALUE  99999999.
     88  FILE-START                VALUE -99999999.
 77  DIGIT                     PIC 9.
 01  HEXDATA                   PIC 9(8) COMP SYNC.
 01  HEXDATA-BYTES REDEFINES HEXDATA.
     05  BYTE-1-2-HEX          PIC X(2).
     05  BYTE-3-4-HEX          PIC X(2).
 77  HEX-QUOTIENT              PIC 9(8) COMP SYNC.
 77  HEX-REMAINDER             PIC 9(8) COMP SYNC.
 01  HEXDIGITS                 PIC X(16)
                                   VALUE "0123456789ABCDEF".
 01  HEXTAB REDEFINES HEXDIGITS.
     05  DIGIT-HEX             PIC X(1)
                                   OCCURS 16 TIMES
                                   INDEXED BY HEX-INDEX.
 01  CHARDATA                  PIC X(8).
 01  CHARDATA-BYTES REDEFINES CHARDATA.
     05  BYTE-1-CHAR           PIC X(2).
     05  BYTE-2-4-CHAR.
         10  BYTE-2-CHAR       PIC X(2).
         10  BYTE-3-4-CHAR     PIC X(4).
 01  CHARDATA-TAB   REDEFINES CHARDATA.
     05  BYTE-CHAR             PIC X(1)
                                   OCCURS 8 TIMES
                                   INDEXED BY CHAR-INDEX.
/
 PROCEDURE DIVISION.
*
*  DISPLAY START MESSAGE
*
 START-MESSAGE.
*
     DISPLAY " "                                  UPON TERMOUT.
     DISPLAY "START PROGRAM TO TEST FLAMREC"      UPON TERMOUT.
     DISPLAY " "                                  UPON TERMOUT.
*
*  OPEN FILE
*
 OPEN-INPUT.
*
     DISPLAY "ENTER PARAMETER FOR FLMOPN: "       UPON TERMOUT
     DISPLAY " "                                  UPON TERMOUT
     DISPLAY "OPENMODE (0=INPUT 1=OUTPUT 2=INOUT) ?"
                                                  UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO OPENMODE
     DISPLAY "DDNAME ?"                           UPON TERMOUT
     ACCEPT   DDNAME                              FROM TERMIN
     DISPLAY "STATIS (0=NO 1=YES) ?"              UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO STATIS
     DISPLAY "LASTPAR (0=YES 1=NO) ?"             UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO LASTPAR
*
```

```
     CALL    "FLMOPN" USING FLAMID, RETCO,
                            LASTPAR, OPENMODE,
                            DDNAME, STATIS
     IF   NOT OK
     THEN
          DISPLAY "ERROR ON 'OPEN' OF: ", DDNAME UPON TERMOUT
          PERFORM  ERROR-MESSAGE
          DISPLAY " "                                UPON TERMOUT
          DISPLAY "PROGRAM TERMINATED WITH ERRORS" UPON TERMOUT
          STOP RUN
     END-IF.
*
 OPEN-NEXT.
*
     IF   NOT LAST-PARAMETER
     THEN
          DISPLAY "PLEASE SELECT FUNCTION: FLMOPD FLMOPF"
                                                     UPON TERMOUT
          ACCEPT   FUNCTION                          FROM TERMIN
          IF   FLMOPD
          THEN
               DISPLAY " "                           UPON TERMOUT
               DISPLAY "ENTER PARAMETER FOR FLMOPD: "
                                                     UPON TERMOUT
               DISPLAY "DATA SET NAME ?"             UPON TERMOUT
               ACCEPT   FILENAME                     FROM TERMIN
               DISPLAY "NAMELEN (0 - 54) ?"          UPON TERMOUT
               PERFORM  NUMERIC-INPUT
               MOVE     INPUT-NUM TO NAMELEN
               IF   OPEN-OUTPUT
               THEN
                    DISPLAY "DSORG   (0=SEQ 1=INDEX ...) ?"
                                                     UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM TO DSORG
                    DISPLAY "RECFORM (0=VAR 1=FIX ...) ?"
                                                     UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM TO RECFORM
                    DISPLAY "MAXSIZE (80 - 32768) ?"
                                                     UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM TO MAXSIZE
                    DISPLAY "KEYDESC FOR ORIGINAL DATA SET ?"
                                                     UPON TERMOUT
                    PERFORM  KEYDESC-INPUT
                    MOVE     KEYDESC-ORIG TO KEYDESC
                    DISPLAY "BLKSIZE (0 - 32768) ?"
                                                     UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM TO BLKSIZE
               END-IF
               DISPLAY "CLOSDISP (0=REWIND 1=UNLOAD ...) ?"
                                                     UPON TERMOUT
               PERFORM  NUMERIC-INPUT
               MOVE     INPUT-NUM TO CLOSDISP
               DISPLAY "DEVICE (0=DISK 1=TAPE ...) ?"
                                                     UPON TERMOUT
```

```
            PERFORM  NUMERIC-INPUT
            MOVE     INPUT-NUM TO DEVICE
            DISPLAY "LASTPAR (0=YES 1=NO) ?"    UPON TERMOUT
            PERFORM  NUMERIC-INPUT
            MOVE     INPUT-NUM TO LASTPAR
            CALL    "FLMOPD" USING FLAMID, RETCO,
                             LASTPAR, NAMELEN, FILENAME,
                             DSORG, RECFORM, MAXSIZE,
                             RECDELIM, KEYDESC, BLKSIZE,
                             CLOSDISP, DEVICE
            IF   NOT OK
            THEN
                 DISPLAY "ERROR ON 'OPEN' OF: ",
                          FILENAME              UPON TERMOUT
                 PERFORM  ERROR-MESSAGE
                 DISPLAY " "                UPON TERMOUT
                 DISPLAY "PROGRAM TERMINATED WITH ERRORS"
                                               UPON TERMOUT
                 STOP RUN
            ELSE
                 DISPLAY "NAMELEN  ", NAMELEN   UPON TERMOUT
                 DISPLAY "DATA SET ", FILENAME  UPON TERMOUT
                 DISPLAY "DSORG    ", DSORG     UPON TERMOUT
                 DISPLAY "RECFORM  ", RECFORM   UPON TERMOUT
                 DISPLAY "MAXSIZE  ", MAXSIZE   UPON TERMOUT
                 IF   DSORG  0 AND KEYPARTS  0
                 THEN
                      DISPLAY "KEYDESC OF FLAMFILE"
                                               UPON TERMOUT
                      DISPLAY "KEYFLAGS ", KEYFLAGS
                                               UPON TERMOUT
                      DISPLAY "KEYPARTS ", KEYPARTS
                                               UPON TERMOUT
                      DISPLAY "KEYPOS1  ", KEYPOS1
                                               UPON TERMOUT
                      DISPLAY "KEYLEN1  ", KEYLEN1
                                               UPON TERMOUT
                      DISPLAY "KEYTYPE1 ", KEYTYPE1
                                               UPON TERMOUT
                 END-IF
                 DISPLAY "BLKSIZE  ", BLKSIZE   UPON TERMOUT
                 DISPLAY "CLOSDISP ", CLOSDISP  UPON TERMOUT
                 DISPLAY "DEVICE   ", DEVICE    UPON TERMOUT
            END-IF
       ELSE
            IF   FLMOPF
            THEN
                 MOVE     1        TO LASTPAR
                 MOVE     DDNAME TO FILENAME
            ELSE
                 DISPLAY  " UNKNOWN FUNCTION: ", FUNCTION
                                               UPON TERMOUT
                 GO TO    OPEN-NEXT
            END-IF
       END-IF
*
       IF   NOT LAST-PARAMETER
       THEN
```

```
DISPLAY " "                          UPON TERMOUT
DISPLAY "ENTER PARAMETER FOR FLMOPF :"
                                     UPON TERMOUT
IF   OPEN-OUTPUT
THEN
     DISPLAY "FLAMCODE (0=EBCDIC 1=ASCII) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO FLAMCODE
     DISPLAY "COMPMODE (0=CX8 1=CX7 2=VR8) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO COMPMODE
     DISPLAY "MAXBUFF (0 - 32768) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO MAXBUFF
     DISPLAY "HEADER (0=NO 1=YES) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO HEADER
     DISPLAY "MAXREC (1 - 255) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO MAXREC
     IF   FLMOPF
     THEN
          DISPLAY "KEYDESC FOR ORIGINAL DATA SET ?"
                                     UPON TERMOUT
          PERFORM  KEYDESC-INPUT
     END-IF
     DISPLAY "BLKMODE (0=UNBLK 1=BLK) ?"
                                     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO BLKMODE
     DISPLAY "EXK20 ?"               UPON TERMOUT
     ACCEPT   EXK20                  FROM TERMIN
ELSE
     IF   FLMOPF
     THEN
          DISPLAY "HEADER (0=NO 1=YES) ?"
                                     UPON TERMOUT
          PERFORM  NUMERIC-INPUT
          MOVE     INPUT-NUM TO HEADER
          IF   OPEN-INOUT
          THEN
               DISPLAY "MAXREC (1 - 255) ?"
                                     UPON TERMOUT
               PERFORM  NUMERIC-INPUT
               MOVE     INPUT-NUM TO MAXREC
               DISPLAY "EXK20 ?"     UPON TERMOUT
               ACCEPT   EXK20        FROM TERMIN
          END-IF
         DISPLAY "KEYDESC FOR ORIGINAL DATA SET ?"
                                     UPON TERMOUT
          PERFORM  KEYDESC-INPUT
     END-IF
     DISPLAY "EXD20 ?"               UPON TERMOUT
```

```
                ACCEPT   EXD20                FROM TERMIN
             END-IF
             CALL    "FLMOPF" USING FLAMID, RETCO,
                              VERSION, FLAMCODE, COMPMODE,
                              MAXBUFF, HEADER, MAXREC,
                              KEYDESC-ORIG, BLKMODE,
                              EXK20, EXD20
*
             IF   NOT OK
             THEN
                DISPLAY "ERROR ON OPEN OF: ",
                         FILENAME             UPON TERMOUT
                PERFORM  ERROR-MESSAGE
                DISPLAY " "                   UPON TERMOUT
                DISPLAY "PROGRAM TERMINATED WITH ERRORS"
                                              UPON TERMOUT
                STOP RUN
              ELSE
                DISPLAY "VERSION  ", VERSION   UPON TERMOUT
                DISPLAY "FLAMCODE ", FLAMCODE  UPON TERMOUT
                DISPLAY "COMPMODE ", COMPMODE  UPON TERMOUT
                DISPLAY "MAXBUFF  ", MAXBUFF   UPON TERMOUT
                DISPLAY "HEADER   ", HEADER    UPON TERMOUT
                DISPLAY "MAXREC   ", MAXREC    UPON TERMOUT
                PERFORM  KEYDESC-OUTPUT
                DISPLAY "BLKMODE  ", BLKMODE   UPON TERMOUT
                DISPLAY "EXK20    ", EXK20     UPON TERMOUT
                DISPLAY "EXD20    ", EXD20     UPON TERMOUT
              END-IF
          END-IF
     END-IF.
*
*****************************************************************
*  PROCESSING LOOP                                              *
*****************************************************************
*
     PERFORM UNTIL FLMCLS
        DISPLAY "PLEASE SELECT FUNCTION: "
                "GET GTR GKY FKY GRN FRN "
                "PUT PKY POS DEL UPD GHD GUH PHD PUH FLU CLS"
                                              UPON TERMOUT
        ACCEPT  FUNCTION                      FROM TERMIN
        IF   FLMGET
        THEN PERFORM SEQUENTIAL-READ
        ELSE
         IF   FLMGTR
         THEN PERFORM SEQUENTIAL-READ-BACKWARD
         ELSE
          IF   FLMPOS
          THEN PERFORM POSITION
          ELSE
           IF   FLMDEL
           THEN PERFORM DELETE
           ELSE
            IF   FLMGKY
            THEN PERFORM READ-KEY
            ELSE
             IF   FLMFKY
```

```
            THEN PERFORM POSITION-KEY
            ELSE
             IF   FLMGRN
             THEN PERFORM READ-RECORD-NUMBER
             ELSE
              IF   FLMFRN
              THEN PERFORM POSITION-RECORD-NUMBER
              ELSE
               IF   FLMPUT
               THEN PERFORM WRITE
               ELSE
                IF   FLMPKY
                THEN PERFORM WRITE-KEY
                ELSE
                 IF   FLMUPD
                 THEN PERFORM UPDATE
                 ELSE
                  IF   FLMPHD
                  THEN PERFORM WRITE-HEADER
                  ELSE
                   IF   FLMPUH
                   THEN PERFORM WRITE-USER-HEADER
                   ELSE
                    IF   FLMGHD
                    THEN PERFORM READ-HEADER
                    ELSE
                     IF   FLMGUH
                     THEN PERFORM READ-USER-HEADER
                     ELSE
                      IF   FLMFLU
                      THEN PERFORM CONCLUDE-MATRIX
                      ELSE
                       IF   FLMCLS
                       THEN DISPLAY FILENAME,
                                    " WILL BE CLOSED"
                                                UPON TERMOUT
                       ELSE DISPLAY FUNCTION,
                                    " UNKNOWN"
                                                UPON TERMOUT
                       END-IF
                      END-IF
                     END-IF
                    END-IF
                   END-IF
                  END-IF
                 END-IF
                END-IF
               END-IF
              END-IF
             END-IF
            END-IF
           END-IF
          END-IF
         END-IF
        END-IF
       END-IF
     END-PERFORM.
*
```

```
 CLOSE FILE-NAME.
*
     CALL   "FLMCLS" USING FLAMID, RETCO CPUTIME REC-ORDS
                           BYTES BYTEOFL CMPRECS CMPBYTES
                           CMPBYOFL.
     IF NOT OK
        DISPLAY "ERROR DURING CLOSE"               UPON TERMOUT
        PERFORM ERROR-MESSAGE
     ELSE
        IF STATISTICS
        THEN
           DISPLAY " "                             UPON TERMOUT
           MOVE     REC-ORDS  TO STATIS-DIS
           DISPLAY "ORG.  RECORDS    ", STATIS-DIS UPON TERMOUT
           MOVE     BYTECNT   TO STATIS-DIS
           DISPLAY "ORG.  BYTES      ", STATIS-DIS UPON TERMOUT
           MOVE     CMPRECS   TO STATIS-DIS
           DISPLAY "COMP. RECORDS    ", STATIS-DIS UPON TERMOUT
           MOVE     CMPBYCNT  TO STATIS-DIS
           DISPLAY "COMP. BYTES      ", STATIS-DIS UPON TERMOUT
        END-IF
        DISPLAY " "                                UPON TERMOUT
        DISPLAY "PROGRAM NORMAL TERMINATION"       UPON TERMOUT
     END-IF.
     STOP RUN.
*
*****************************************************************
*  VERARBEITUNGSFUNKTIONEN                                      *
*  PROCESSING FUNCTIONS                                         *
*****************************************************************
*
 SEQUENTIAL-READ.
*
     DISPLAY "NUMBER RECORDS TO READ ?"            UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO GET-COUNT.
     MOVE    0 TO RETCO.
     PERFORM VARYING GET-INDEX FROM 0 BY 1
             UNTIL GET-INDEX = GET-COUNT OR NOT OK
        MOVE    SPACES TO RECORD-DISPLAY
        CALL   "FLMGET" USING FLAMID, RETCO,
                              RECLEN, REC-ORD, BUFLEN
        IF GAP
           DISPLAY "*** FOUND EMPTY SLOT ***"      UPON TERMOUT
           MOVE    0 TO RETCO
        ELSE
           IF OK OR CUT
              DISPLAY RECORD-DISPLAY               UPON TERMOUT
           END-IF
        END-IF
     END-PERFORM.
     IF NOT OK
        DISPLAY "ERROR ON 'GET RECORD'"            UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 SEQUENTIAL-READ-BACKWARD.
*
```

```
     DISPLAY "NUMBER RECORDS TO READ ?"           UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO GET-COUNT.
     MOVE    0 TO RETCO.
     PERFORM VARYING GET-INDEX FROM 0 BY 1
             UNTIL GET-INDEX = GET-COUNT OR NOT OK
        MOVE    SPACES TO RECORD-DISPLAY
        CALL   "FLMGTR" USING FLAMID, RETCO,
                        RECLEN, REC-ORD, BUFLEN
        IF GAP
           DISPLAY "*** FOUND EMPTY SLOT ***"     UPON TERMOUT
           MOVE    0 TO RETCO
        ELSE
           IF OK OR CUT
              DISPLAY RECORD-DISPLAY              UPON TERMOUT
           END-IF
        END-IF
     END-PERFORM.
     IF NOT OK
        DISPLAY "ERROR ON 'READ BACKWARD'"        UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 READ-RECORD-NUMBER.
*
     DISPLAY " "                                  UPON TERMOUT.
     DISPLAY "RECORD NUMBER ?"                    UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO RECNO.
     MOVE    SPACES TO RECORD-DISPLAY
     CALL "FLMGRN" USING FLAMID, RETCO, RECLEN, REC-ORD
                                        BUFLEN, RECNO.
     IF GAP
        DISPLAY "*** FOUND EMPTY SLOT ***"        UPON TERMOUT
        MOVE    0 TO RETCO
     ELSE
        IF OK OR CUT
           DISPLAY RECORD-DISPLAY                 UPON TERMOUT
        END-IF
     END-IF
     IF NOT OK
        DISPLAY "ERROR ON 'POSITION TO RECORD NUMBER'"
                                                  UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 POSITION-RECORD-NUMBER.
*
     DISPLAY " "                                  UPON TERMOUT.
     DISPLAY "RECORD NUMBER ?"                    UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO RECNO.
     DISPLAY "CHECKMODE (0/1/2) ?"                UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO CHECKMODE.
     CALL "FLMFRN" USING FLAMID, RETCO, RECNO, CHECKMODE.
     IF NOT OK
```

```
        DISPLAY "ERROR ON 'POSITION TO RECORD NUMBER'"
                                                    UPON TERMOUT
        PERFORM ERROR-MESSAGE
     ELSE
        DISPLAY "RECORD NUMBER: ", RECNO            UPON TERMOUT
     END-IF.
*
 POSITION.
*
     DISPLAY " "                                    UPON TERMOUT.
     DISPLAY "RELATIVE POSITION ?"                  UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO REL-POSITION.
     CALL "FLMPOS" USING FLAMID, RETCO, REL-POSITION.
     IF NOT OK
        DISPLAY "ERROR ON 'POSITION'"               UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 DELETE.
*
     CALL   "FLMDEL" USING FLAMID, RETCO,
     IF NOT OK
        DISPLAY "ERROR ON 'DELETE'"                 UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 READ-KEY.
*
     DISPLAY "RECORD KEY ?"                         UPON TERMOUT.
     MOVE    SPACES TO REC-ORD.
     ACCEPT  RECORD-KEY-DISPLAY                     FROM TERMIN.
     SET     KEY-INDEX      TO   1.
     SET     REC-INDEX      TO   KEYPOS1-ORIG.
     PERFORM VARYING KEY-IND-DISP FROM 0 BY 1
             UNTIL KEY-IND-DISP = KEYLEN1-ORIG
        MOVE  RECORD-KEY-BYTE(KEY-INDEX)  TO  BYTE(REC-INDEX)
        SET   KEY-INDEX  UP  BY  1
        SET   REC-INDEX  UP  BY  1
     END-PERFORM.
     CALL   "FLMGKY" USING FLAMID, RETCO,
                     RECLEN, REC-ORD, BUFLEN.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'READ WITH KEY'"          UPON TERMOUT
        PERFORM ERROR-MESSAGE
        MOVE    RECORD-KEY-DISPLAY  TO  RECORD-DISPLAY
        DISPLAY "RECORD BUFFER: "                   UPON TERMOUT
        DISPLAY RECORD-DISPLAY                      UPON TERMOUT
     ELSE
        DISPLAY RECORD-DISPLAY                      UPON TERMOUT
     END-IF.
*
 POSITION-KEY.
*
     DISPLAY "RECORD KEY ?"                         UPON TERMOUT.
     MOVE    SPACES TO REC-ORD.
     ACCEPT  RECORD-KEY-DISPLAY                     FROM TERMIN.
```

```
     DISPLAY "CHECKMODE (0/1/2) ?"                 UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO CHECKMODE.
     SET     KEY-INDEX      TO   1.
     SET     REC-INDEX      TO   KEYPOS1-ORIG.
     PERFORM VARYING KEY-IND-DISP FROM 0 BY 1
             UNTIL KEY-IND-DISP = KEYLEN1-ORIG
        MOVE  RECORD-KEY-BYTE(KEY-INDEX)  TO  BYTE(REC-INDEX)
        SET   KEY-INDEX  UP  BY  1
        SET   REC-INDEX  UP  BY  1
     END-PERFORM.
     CALL   "FLMFKY" USING FLAMID, RETCO,
                     RECLEN, REC-ORD, CHECKMODE.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'FIND WITH KEY'"         UPON TERMOUT
        PERFORM ERROR-MESSAGE
        MOVE    RECORD-KEY-DISPLAY  TO  RECORD-DISPLAY
        DISPLAY "RECORD BUFFER: "                  UPON TERMOUT
        DISPLAY RECORD-DISPLAY                     UPON TERMOUT
     END-IF.
*
 WRITE.
*
     DISPLAY "RECORD LENGTH ?"                     UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO RECLEN.
     DISPLAY "RECORD DATA ?"                       UPON TERMOUT.
     MOVE    SPACES TO RECORD-DISPLAY
     ACCEPT  RECORD-DISPLAY                        FROM TERMIN.
     CALL   "FLMPUT" USING FLAMID, RETCO,
                     RECLEN, REC-ORD.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'WRITE A RECORD'"        UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 WRITE-KEY.
*
     DISPLAY "RECORD LENGTH ?"                     UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO RECLEN.
     DISPLAY "RECORD INCLUSIVE KEY ?"              UPON TERMOUT.
     MOVE    SPACES TO RECORD-DISPLAY
     ACCEPT  RECORD-DISPLAY                        FROM TERMIN.
     CALL   "FLMPKY" USING FLAMID, RETCO,
                     RECLEN, REC-ORD.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'WRITE WITH KEY'"        UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 UPDATE.
*
     DISPLAY "RECORD LENGTH ?"                     UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
```

```
     MOVE    INPUT-NUM TO RECLEN.
     DISPLAY "RECORD INCLUSIVE KEY ?"              UPON TERMOUT.
     MOVE    SPACES TO RECORD-DISPLAY
     ACCEPT  RECORD-DISPLAY                        FROM TERMIN.
     CALL   "FLMUPD" USING FLAMID, RETCO,
                     RECLEN, REC-ORD, BUFLEN.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'UPDATE'"                UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 WRITE-HEADER.
*
     DISPLAY "DATA SET NAME ?"                     UPON TERMOUT
     ACCEPT   FILENAME-ORIG                        FROM TERMIN
     DISPLAY "NAMELEN (0 - 54) ?"                  UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO NAMELEN-ORIG
     DISPLAY "DSORG (0=SEQ 1=INDEX 2=REL ...) ?" UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO DSORG-ORIG
     DISPLAY "RECFORM (0=VAR 1=FIX 2=UNDEF ...) ?" UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO RECFORM-ORIG
     DISPLAY "RECSIZE (0 - 32768) ?"               UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO RECSIZE-ORIG
     DISPLAY "BLKSIZE (0 - 32768) ?"               UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO BLKSIZE-ORIG
     IF   NOT KEYDESC-DEFINES
     THEN
          PERFORM  KEYDESC-EINGABE
          MOVE    "N" TO KEYDESC-INDICATOR
     END-IF
     DISPLAY "PRCTRL (0=NO 1=MACHINE 2=ASA) ?"     UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO PRCTRL-ORIG
     MOVE     LOW-VALUES  TO SYSTEM-ORIG
     DISPLAY "LASTPAR (0=YES 1=NO) ?"              UPON TERMOUT
     PERFORM  NUMERIC-INPUT
     MOVE     INPUT-NUM TO LASTPAR-PHD
*
     CALL   "FLMPHD" USING FLAMID, RETCO,
                     NAMELEN-ORIG, FILENAME-ORIG,
                     DSORG-ORIG, RECFORM-ORIG,
                     RECSIZE-ORIG, RECDELIM-ORIG,
                     KEYDESC-ORIG, BLKSIZE-ORIG,
                     PRCTRL-ORIG, SYSTEM-ORIG,
                     LASTPAR-PHD.
     IF NOT OK
     THEN
          DISPLAY "ERROR ON 'WRITE HEADER'"        UPON TERMOUT
          PERFORM ERROR-MESSAGE
     ELSE
          IF NOT LAST-PARAMETER-PHD
          THEN
```

```
               DISPLAY " "                          UPON TERMOUT
               DISPLAY "WRITE USER HEADER: "        UPON TERMOUT
               PERFORM WRITE-USER-HEADER
          END-IF
     END-IF.
*
 WRITE-USER-HEADER.
*
     DISPLAY "HEADERLENGTH ?"                       UPON TERMOUT.
     PERFORM NUMERIC-INPUT.
     MOVE    INPUT-NUM TO UATTRLEN.
     DISPLAY "YOUR INPUT, PLEASE"                   UPON TERMOUT.
     ACCEPT  USERATTR                               FROM TERMIN.
     CALL   "FLMPUH" USING FLAMID, RETCO,
                     UATTRLEN, USERATTR.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'WRITE USER HEADER'"      UPON TERMOUT
        PERFORM ERROR-MESSAGE
     END-IF.
*
 READ-HEADER.
*
     MOVE   54       TO NAMELEN-ORIG.
     MOVE   SPACES   TO FILENAME-ORIG.
     CALL   "FLMGHD" USING FLAMID, RETCO,
                     NAMELEN-ORIG, FILENAME-ORIG,
                     DSORG-ORIG, RECFORM-ORIG,
                     RECSIZE-ORIG, RECDELIM-ORIG,
                     KEYDESC-ORIG, BLKSIZE-ORIG,
                     PRCTRL-ORIG, SYSTEM-ORIG.
     IF NOT OK
     THEN
        DISPLAY "ERROR ON 'READ HEADER'"            UPON TERMOUT
        PERFORM ERROR-MESSAGE
     ELSE
        DISPLAY "NAMELEN  ", NAMELEN-ORIG           UPON TERMOUT
        DISPLAY "DATA SET ", FILENAME-ORIG          UPON TERMOUT
        DISPLAY "DSORG    ", DSORG-ORIG             UPON TERMOUT
        DISPLAY "RECFORM  ", RECFORM-ORIG           UPON TERMOUT
        DISPLAY "RECSIZE  ", RECSIZE-ORIG           UPON TERMOUT
        PERFORM  KEYDESC-OUTPUT
        DISPLAY "BLKSIZE  ", BLKSIZE-ORIG           UPON TERMOUT
        DISPLAY "PRCTRL   ", PRCTRL-ORIG            UPON TERMOUT
        DISPLAY "RECSIZE  ", RECSIZE-ORIG           UPON TERMOUT
        MOVE     SYSTEM-ORIG TO BYTE-3-4-HEX
        PERFORM  HEX-TO-CHAR
        DISPLAY "SYSTEM   ", BYTE-3-4-CHAR          UPON TERMOUT
     END-IF.
*
 READ-USER-HEADER.
*
     MOVE   80       TO UATTRLEN.
     MOVE   SPACES   TO USERATTR.
     CALL   "FLMGUH" USING FLAMID, RETCO,
                     UATTRLEN, USERATTR.
     IF NOT OK
     THEN
```

```
        DISPLAY "ERROR ON 'READ USER HEADER'"      UPON TERMOUT
        PERFORM ERROR-MESSAGE
     ELSE
        DISPLAY "LENGTH USERHEADER ", UATTRLEN     UPON TERMOUT
        IF UATTRLEN  0
        THEN
           DISPLAY USERATTR                        UPON TERMOUT
        END-IF
     END-IF.
*
 CONCLUDE-MATRIX.
*
     CALL   "FLMFLU" USING FLAMID, RETCO CPUTIME REC-ORDS
                           BYTES BYTEOFL CMPRECS CMPBYTES
                           CMPBYOFL.
     IF NOT OK
        DISPLAY "ERROR ON 'FLUSH MATRIX'  "        UPON TERMOUT
        PERFORM ERROR-MESSAGE
     ELSE
        IF STATISTICS
        THEN
           DISPLAY " "                             UPON TERMOUT
           MOVE     REC-ORDS  TO STATIS-DIS
           DISPLAY "ORG.  RECORDS ", STATIS-DIS UPON TERMOUT
           MOVE     BYTECNT   TO STATIS-DIS
           DISPLAY "ORG.  BYTES   ", STATIS-DIS UPON TERMOUT
           MOVE     CMPRECS   TO STATIS-DIS
           DISPLAY "COMP. RECORDS ", STATIS-DIS UPON TERMOUT
           MOVE     CMPBYCNT  TO STATIS-DIS
           DISPLAY "COMP. BYTES   ", STATIS-DIS UPON TERMOUT
        END-IF
     END-IF.
*
*****************************************************************
*    HELP FUNCTIONS                                             *
*****************************************************************
*
 ERROR-MESSAGE.
*
     IF   INVALID
     THEN DISPLAY "ILLEGAL FUNCTION"               UPON TERMOUT
     ELSE
          IF   DVS-ERROR
          THEN
               MOVE     LOW-VALUE TO RETCO-INDICATOR
               MOVE     RETCO TO HEXDATA
               PERFORM  HEX-TO-CHAR
               DISPLAY "VSAM ERROR CODE: ", BYTE-2-4-CHAR
                                                  UPON TERMOUT
          ELSE
               DISPLAY "FLAM ERROR CODE: ", RETCO-FLAM
                       UPON TERMOUT
          END-IF
     END-IF.
*
 NUMERIC-INPUT.
*
     ACCEPT  INPUT                          FROM TERMIN.
```

```
     MOVE    0 TO INPUT-NUM.
     SET     RED-INDEX TO 8.
     PERFORM VARYING INP-INDEX
                FROM 9 BY -1 UNTIL INP-INDEX = 0
                             OR RED-INDEX = 0
             IF      BYTE-INP(INP-INDEX) NUMERIC
             THEN    MOVE BYTE-INP(INP-INDEX)
                       TO BYTE-RED(RED-INDEX)
                     SET  RED-INDEX DOWN BY 1
             END-IF
     END-PERFORM.
     IF      BYTE-INP(1) = "-"
     THEN    COMPUTE INPUT-NUM = -1 * INPUT-NUM
     END-IF.
*
 HEX-TO-CHAR.
*
      PERFORM VARYING CHAR-INDEX
                 FROM 8 BY -1 UNTIL CHAR-INDEX = 1
              DIVIDE HEXDATA BY 16 GIVING HEX-QUOTIENT
                 REMAINDER HEX-REMAINDER
              END-DIVIDE
              ADD  1             TO HEX-REMAINDER
              SET  HEX-INDEX     TO HEX-REMAINDER
              MOVE HEX-QUOTIENT TO HEXDATA
              MOVE DIGIT-HEX(HEX-INDEX)
                TO BYTE-CHAR(CHAR-INDEX)
      END-PERFORM.
*
 KEYDESC-INPUT.
*
      DISPLAY "KEYPARTS (0 - 8) ?"                 UPON TERMOUT
      PERFORM  NUMERIC-INPUT
      MOVE     INPUT-NUM TO KEYPARTS-ORIG
      IF   KEYPARTS-ORIG  0
      THEN
           DISPLAY "KEYFLAGS (0=NODUP 1=DUPKY) ?"  UPON TERMOUT
           PERFORM  NUMERIC-INPUT
           MOVE     INPUT-NUM TO KEYFLAGS-ORIG
           DISPLAY "KEYPOS1 (1 - 32767) ?"         UPON TERMOUT
           PERFORM  NUMERIC-INPUT
           MOVE     INPUT-NUM TO KEYPOS1-ORIG
           DISPLAY "KEYLEN1 (1 - 255) ?"           UPON TERMOUT
           PERFORM  NUMERIC-INPUT
           MOVE     INPUT-NUM TO KEYLEN1-ORIG
           DISPLAY "KEYTYPE1 (0=DISP 1=BINARY) ?"  UPON TERMOUT
           PERFORM  NUMERIC-INPUT
           MOVE     INPUT-NUM TO KEYTYPE1-ORIG
           PERFORM  VARYING KEYDESC-INDEX FROM 1 BY 1
                       UNTIL KEYDESC-INDEX = KEYPARTS-ORIG
                    SET      DIGIT TO KEYDESC-INDEX
                    ADD      1 TO DIGIT
                    DISPLAY "KEYPOS", DIGIT, " (1 - 32767) ?"
                                                 UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM
                      TO     KEYPOS-ORIG(KEYDESC-INDEX)
                    DISPLAY "KEYLEN", DIGIT, " (1 - 255) ?"
```

```
                                                UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM
                      TO     KEYLEN-ORIG(KEYDESC-INDEX)
                    DISPLAY "KEYTYPE", DIGIT, " (0=DISP 1=BIN) ?"
                                                UPON TERMOUT
                    PERFORM  NUMERIC-INPUT
                    MOVE     INPUT-NUM
                      TO     KEYTYPE-ORIG(KEYDESC-INDEX)
          END-PERFORM
      END-IF.
*
 KEYDESC-OUTPUT.
*
      IF   KEYPARTS-ORIG  0
      THEN
          DISPLAY "KEYDESC OF ORIGINAL DATA SET"  UPON TERMOUT
          DISPLAY "KEYPARTS ", KEYPARTS-ORIG      UPON TERMOUT
          DISPLAY "KEYFLAGS ", KEYFLAGS-ORIG      UPON TERMOUT
          DISPLAY "KEYPOS1  ", KEYPOS1-ORIG       UPON TERMOUT
          DISPLAY "KEYLEN1  ", KEYLEN1-ORIG       UPON TERMOUT
          DISPLAY "KEYTYPE1 ", KEYTYPE1-ORIG      UPON TERMOUT
          PERFORM  VARYING KEYDESC-INDEX FROM 1 BY 1
                      UNTIL KEYDESC-INDEX = KEYPARTS-ORIG
             SET      DIGIT TO KEYDESC-INDEX
             ADD      1 TO DIGIT
             DISPLAY "KEYPOS", DIGIT, "  ",
                      KEYPOS-ORIG(KEYDESC-INDEX)  UPON TERMOUT
             DISPLAY "KEYLEN", DIGIT, "  ",
                      KEYLEN-ORIG(KEYDESC-INDEX)  UPON TERMOUT
             DISPLAY "KEYTYPE", DIGIT, " ",
                      KEYTYPE-ORIG(KEYDESC-INDEX) UPON TERMOUT
          END-PERFORM
      END-IF.
```

5.3 User I/O interface

5.3.1 ASSEMBLER example

This example sets up a DUMMY device that returns immediately the return code END OF FILE during read. During write all records are accepted and always OK is returned without actually writing the records to a storage medium. The functions USRGKY and USRPOS always deliver the return code INVALID KEY or INVALID POSITION. The function USRDEL always delivers the return code INVALID FUNCTION.

This functionality is equivalent with a file assignment to DUMMY.

By filling in appropriate code into the sequences marked with three periods, this routine can be used as a template for specific user written I/O routines.

```
FLAMUIO  START
         TITLE 'FLAMUIO: USER-I/O-MODULE FOR FLAM'
*********************************************************************
* NAME: FLAMUIO                                                     *
* FUNCTION:                                                         *
*        DUMMY MODULE AS EXAMPLE FOR AN USER-IO-MODULE              *
*  INTERFACES:                                                      *
*        USROPN   OPEN DATA SET                                     *
*        USRCLS   CLOSE DATA SET                                    *
*        USRGET   READ SEQUENTIAL                                   *
*        USRGKY   READ WITH KEY                                     *
*        USRPUT   WRITE SEQUENTIAL                                  *
*        USRPKY   WRITE WITH KEY                                    *
*        USRDEL   DELETE ACTUAL RECORD                              *
*        USRPOS   POSITION IN DATA SET                              *
*  NOTES:                                                           *
*        ALL FUNCTIONS ARE REENTRANT.                               *
*        WE NEED NO RUN TIME SYSTEM.                                *
*        INDEPENDENT FROM ANY /370-SYSTEM.                          *
*********************************************************************
*
*  ADDRESSING -/ RESIDENCY MODE
*
FLAMUIO  AMODE ANY
FLAMUIO  RMODE ANY
*
*  RETURN CODES
*
OK       EQU   0            NO ERROR
*        EQU   -1           REQM-ERROR; INVALID HANDLE
*                            OR INVALID FUNCTION
CUT      EQU   1            RECORD TRUNCATED
EOF      EQU   2            END OF DATA SET
GAP      EQU   3            GAP IN RELATIVE DATA SET
FILL     EQU   4            RECORD PADDED
INVKEY   EQU   5            KEY NOT FOUND
```

```
RCEMPTY  EQU   30            INPUT DATA SET EMPTY
RCNEXIST EQU   31            DATA SET DOES NOT EXIST
RCOPENMO EQU   32            INVALID OPEN MODE
RCFCBTYP EQU   33            INVALID FILE FORMAT
RCRECFOR EQU   34            INVALID RECORD FORMAT
RCRECSIZ EQU   35            INVALID RECORD LENGTH
RCBLKSIZ EQU   36            INVALID BLOCK SIZE
RCKEYPOS EQU   37            INVALID KEY POSITION
RCKEYLEN EQU   38            INVALID KEY LENGTH
RCDSN    EQU   39            INVALID DATA SET NAME
*        EQU   X'0FXXXXXX'   OTHER ERRORS
*
***********************************
* REGISTER EQUATES                *
***********************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*        DC    C'*** MODULE FLAMUIO. '
         DC    C'USER-I/O-MODULE FOR FLAM '
         DC    C'TIME - DATE ASSEMBLED: '
         DC    C'&SYSDATE - &SYSTIME ***'
         TITLE 'USROPN'
USROPN   DS    0D
         ENTRY USROPN
         USING USROPN,R10
*************************************************************************
*  NAME: USROPN                                                         *
*  FUNCTION:                                                            *
*        OPEN DATA SET                                                  *
*  PARAMETER:                                                           *
*  1 <-> WORKAREA 256F   WORK AREA, INITIALIZED WITH X'00'.             *
*                        THIS AREA IS CONNECTED TO THIS DATA SET.       *
*                        USABLE AS WORK AREA DURING THE DIFFERENT CALLS *
*                        FOR THE ACTUAL DATA SET.                       *
*  2 <-  RETCO    F      RETURN CODE                                    *
*        = 0             NO ERROR                                       *
*        = 30            INPUT DATA SET IS EMPTY                        *
*        = 31            DATA SET NOT CONNECTED OR DOES NOT EXIST       *
*        = 32            ILLEGAL OPEN MODE                              *
*        = 33            ILLEGAL DSORG                                  *
*        = 34            ILLEGAL RECORD FORMAT                          *
*        = 35            ILLEGAL RECORD LENGTH                          *
*        = 36            ILLEGAL BLOCK SIZE                             *
*        = 37            ILLEGAL KEY POSITION                           *
```

```
*        = 38             ILLEGAL KEY LENGTH                              *
*        = -1             UNSUPPORTED FUNCTION; GETMAIN ERROR             *
*        = X'0FXXXXXX'    OTHER ERROR CODE                                *
*  3 ->  OPENMODE F       OPEN MODE                                       *
*        = 0              INPUT (SEQUENTIAL READ)                         *
*                         (DATA SET MUST EXIST)                           *
*        = 1              OUTPUT (SEQUENTIAL WRITE)                       *
*                         (DATA SET WILL BE OVERWRITTEN)                  *
*        = 2              INOUT (READ OR WRITE SEQUENTIAL OR WITH KEY)    *
*                         (DATA SET MUST EXIST)                           *
*        = 3              OUTIN (WRITE OR READ SEQUENTIAL OR WITH KEY)    *
*                         (DATA SET WILL BE OVERWRITTEN)                  *
*  4 -> DDNAME  CL8       DD-NAME                                         *
*  5 <-> DSORG    F       DATA SET ORGANIZATION                           *
*        = 0;  8; 16 ... SEQUENTIAL                                       *
*        = 1;  9; 17 ... INDEX SEQUENTAIL                                 *
*        = 2; 10; 18 ... RELATIVE                                         *
*        = 3; 11; 19 ... DIRECT                                           *
*        = 4; 12; 20 ... UNSTRUCTURED                                     *
*        = 5; 13; 21 ... LIBRARY                                          *
*  6 <->   RECFORM  F     RECORD FORMAT                                   *
*        = 0;  8; 16 ... VARIABELE (V)                                    *
*                         8 = BLOCKED  16 = BLOCKED/SPAN NED              *
*        = 1;  9; 17 ... FIX (F)                                          *
*                         9 = BLOCKED  17 = BLOCKED/SPANNED               *
*        = 2; 10; 18 ... UNDEFINED (U)                                    *
*                                                                         *
*        = 3; 11; 19 ... STREAM (S)                                       *
*                         11 = DELIMITER    19 RECORD DESCRIPTOR WORD     *
*  7 <-> RECSIZE  F       DATA LENGTH (WITHOUT DELIMTER OR RDW)           *
*        = 0 - 32767                                                      *
*          RECFORM = V:  MAX. RECORD LENGTH OR 0                          *
*          RECFORM = F:  RECORD LENGTH                                    *
*          RECFORM = U:  MAX. RECORD LENGTH OR 0                          *
*          RECFORM = S:  LENGTH DELIMITER OR RDW                          *
*  8 <->    BLKSIZE  F    BLOCK SIZE                                     *
*        = 0              UNBLOCKED                                      *
*  9 <->    KEYDESC  STRUCT KEY DESCRIPTION                              *
*                                                                         *
*        KEYFLAGS F       OPTIONS                                        *
*        = 0              NO DUPLICATE KEYS                              *
*        = 1              DUPLICATES ALLOWED                             *
*        KEYPARTS F       NUMBER OF KEY PARTS                            *
*        = 0 - 8                                                         *
*        KEYPOS1  F       1. BYTE OF 1. KEYPART                          *
*        = 1 - 32766                                                     *
*        KEYLEN1  F       LENGTH OF 1. KEYPART                           *
*        = 1 - 255                                                       *
*        KEYTYPE1 F       DATA TYPE OF 1. KEYPART                        *
*        = 0              PRINTABLE CHARACTER                            *
*        = 1              BINARY                                         *
*        .                                                               *
*        .                                                               *
*        .                                                               *
*        KEYPOS8  F       1. BYTE OF 8. KEYPART                          *
*        = 1 - 32766                                                     *
*        KEYLEN8  F       LENGTH OF 8. KEYPART                           *
*        = 1 - 255                                                       *
```

```
*        KEYTYPE8 F      DATA TYPE OF 8. KEYPART                          *
*        = 0             PRINTABLE CHARACTER                              *
*        = 1             BINARY                                           *
*                                                                         *
* 10 <->    DEVICE   F   DEVICE TYPE                                      *
*        = 7; 15; 23     USER DEFINED                                     *
* 11 <->    RECDELIM XL  RECORD DELIMITER                                 *
* 12 ->   PADCHAR  XL1   PADDING CHARACTER                                *
* 13 <->    PRCTRL   F   PRINTER CONTROL CHARACTER                        *
*        = 0             NONE                                             *
*        = 1             ASA-CHARACTER                                    *
*        = 2             MACHINE SPECIFIC CHARACTER                       *
* 14 ->   CLOSDISP F     CLOSE PROCESSING                                 *
*        = 0             REWIND                                           *
*        = 1             UNLOAD                                           *
*        = 2             RETAIN / LEAVE                                   *
* 15 ->   ACCESS   F     ACCESS METHOD                                    *
*        = 0             LOGICAL (RECORD BY RECORD)                       *
*        = 1             PHYSICAL                                         *
* 16 <->   DSNLEN   F    LENGTH OF DATA SET NAME OR BUFFER FOR NAME       *
* 17 <->   DSN      CL   DATA SET NAME                                    *
*                        (DATA SET NAME SHOULD BE RETURNED, IF 1. BYTE    *
*                        OF GIVEN NAME IS C' ' OR A DIFFERENT DATA SET    *
*                        IS ALLOCATED).                                   *
************************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R2,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  OPEN DATA SET
*
*        .
*        .
*        .
*
*  SET RETURN CODE TO 'NO ERROR'
*
         LA    R0,OK
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
```

```
         DROP  R12
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRCLS'
USRCLS   DS    0D
         ENTRY USRCLS
         USING USRCLS,R10
*********************************************************************
*  NAME: USRCLS                                                     *
*  FUNCTION:                                                        *
*        CLOSE DATA SET                                             *
*  PARAMETER:                                                       *
*  1 <->  WORKAREA 256F  WORK AREA                                  *
*  2 <-  RETCO    F      RETURN CODE                                *
*        = 0             NO ERROR                                   *
*        = -1            UNSUPPORTED FUNCTION                       *
*        = X'0FXXXXXX'   ELSE                                       *
*        OR              DMS-ERROR CODE                             *
*********************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R2,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  CLOSE DATA SET
*
*        .
*        .
*        .
*
*  SET RETURN CODE TO 'NO ERROR'
*
         LA    R0,OK
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
```

```
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRGET'
USRGET   DS    0D
         ENTRY USRGET
         USING USRGET,R10
*********************************************************************
*  NAME: USRGET                                                     *
*  FUNCTION:                                                        *
*        READ A RECORD (SEQUENTIAL)                                 *
*  PARAMETER:                                                       *
*  1 <-> WORKAREA 256F   WORK AREA                                  *
*  2 <-  RETCO    F      RETURN CODE                                *
*        = 0             NO ERROR                                   *
*        = 1             RECORD TRUNCATED                           *
*        = 2             END OF FILE                                *
*        = 3             EMPTY SLOT IN RELATIVE RECORD DATA SET     *
*        = -1            UNSUPPORTED FUNCTION                       *
*        = X'0FXXXXXX'   ELSE                                       *
*  3 <-  RECLEN   F      RECORD LENGTH IN BYTES                     *
*  4 <-  RECORD   XL     RECORD                                     *
*  5 ->  BUFLEN   F      LENGTH OF RECORD BUFFER IN BYTES           *
*********************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R5,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  READ A RECORD
*
*        .
*        .
*        .
*
*  HERE:  RETURN CODE 'END OF FILE'
*
         LA    R0,EOF
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
```

```
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
*
************************************************************************
*  LOCAL CONSTANTS                                                     *
************************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRGKY'
USRGKY   DS    0D
         ENTRY USRGKY
         USING USRGKY,R10
************************************************************************
*  NAME: USRGKY                                                        *
*  FUNCTION:                                                           *
*        READ RECORD WITH GIVEN RECORD-KEY                             *
*  PARAMETER:                                                          *
*  1 <-> WORKAREA 256F   WORK AREA                                     *
*  2 <-  RETCO    F      RETURN CODE                                   *
*        = 0             NO ERROR                                      *
*        = 1             RECORD TRUNCATED                              *
*        = 2             END OF FILE                                   *
*        = 5             KEY NOT FOUND                                 *
*        = -1            UNSUPPORTED FUNCTION                          *
*        = X'0FXXXXXX'   ELSE                                          *
*  3 <-  RECLEN   F      RECORD LENGTH IN BYTES                        *
*  4 <-  RECORD   XL     RECORD WITH SEARCH KEY                        *
*  5 ->  BUFLEN   F      LENGTH OF RECORD BUFFER IN BYTES              *
************************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R5,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  READ RECORD
*
*        .
*        .
*        .
*
*  HERE: RETURN CODE 'RECORD NOT FOUND'
*
         LA    R0,INVKEY
         ST    R0,0(R2)
*
```

```
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREAS REGISTER
*
         DROP  R12
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRPUT'
USRPUT   DS    0D
         ENTRY USRPUT
         USING USRPUT,R10
*********************************************************************
*  NAME: USRPUT                                                     *
*  FUNCTION:                                                        *
*        WRITE A RECORD (SEQUENTIAL)                                *
*  PARAMETER:                                                       *
*  1 <->  WORKAREA 256F  WORK AREA                                  *
*  2 <-  RETCO    F      RETURN CODE                                *
*        = 0             NO ERROR                                   *
*        = 1             RECORD TRUNCATED                           *
*        = 4             RECORD FILLED WITH PADDING CHARACTER       *
*        = -1            UNSUPPORTED FUNCTION                       *
*        = X'0FXXXXXX'   ELSE                                       *
*  3 ->  RECLEN   F      RECORD LENGTH IN BYTES                     *
*  4 ->  RECORD   XL     RECORD                                     *
*********************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R4,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  WRITE THE RECORD
*
*        .
*        .
*        .
*
*  RETURN CODE: 'NO ERROR'
*
         LA    R0,OK
```

```
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
*
*************************************************************************
*  LOCAL CONSTANTS                                                      *
 ************************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRPKY'
USRPKY   DS    0D
         ENTRY USRPKY
         USING USRPKY,R10
*************************************************************************
*  NAME: USRPKY                                                         *
*  FUNCTION:                                                            *
*        WRITE A RECORD WITH GIVEN KEY  (INDEX SEQUENTIAL)              *
*  PARAMETER:                                                           *
*  1 <->   WORKAREA 256F WORK AREA                                      *
*  2 <-   RETCO    F     RETURN CODE                                    *
*        = 0             NO ERROR                                       *
*        = 1             RECORD TRUNCATED                               *
*        = 4             RECORD FILLED WITH PADDING CHARACTER           *
*        = 5             INVALID KEY                                    *
*        = -1            UNSUPPORTED FUNCTION                           *
*        = X'0FXXXXXX'   ELSE                                           *
*  3 ->  RECLEN   F      RECORD LENGTH IN BYTES                         *
*  4 ->  RECORD   XL     RECORD                                         *
*  NOTES:                                                               *
*        IF THE GIVEN KEY IS THE SAME LIKE THE LAST KEY READ            *
*        THE RECORD SHALL BE OVERWRITTEN (REWRITE).                     *
*        OTHERWISE THE RECORD SHALL BE INSERTED.                        *
*************************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R5,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  WRITE THE RECORD
*
```

```
*        .
*        .
*        .
*
*  RETURN CODE:  'NO ERROR'
*
         LA    R0,OK
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRDEL'
USRDEL   DS    0D
         ENTRY USRDEL
         USING USRDEL,R10
*********************************************************************
*  NAME: USRDEL                                                     *
*  FUNCTION:                                                        *
*        DELETE ACTUAL RECORD                                       *
*  PARAMETER:                                                       *
*  1 <->  WORKAREA 256F  WORK AREA                                  *
*  2 <-  RETCO    F      RETURN CODE                                *
*        = 0             NO ERROR                                   *
*        = 5             NO ACTUAL RECORD READ                      *
*        = -1            UNSUPPORTED FUNCTION                       *
*        = X'0FXXXXXX'   ELSE                                       *
*********************************************************************
*
*  SAVE REGISTER AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
*
*  LOAD PARAMETER
*
         LM    R1,R2,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  DELETE RECORD
*
*        .
```

```
*        .
*        .
*
*  HERE:  RETURN CODE 'NO ACTUAL = RECORD READ'
*
         LA    R0,INVKEY
         ST    R0,0(R2)
*
*  RETURN TO CALLER
*
         LM    R14,R12,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'USRPOS'
USRPOS   DS    0D
         ENTRY USRPOS
         USING USRPOS,R10
*********************************************************************
*  NAME: USRPOS                                                     *
*  FUNCTION:                                                        *
*        POSITION IN DATA SET                                       *
*  PARAMETER:                                                       *
*  1 <->  WORKAREA F     WORK AREA                                  *
*  2 <-   RETCO    F     RETURN CODE                                *
*        = 0             OK                                         *
*        = 5             ILLEGAL POSITION                           *
*        = -1            UNSUPPORTED FUNCTION                       *
*        = X'0FXXXXXX'   ELSE                                       *
*  3 ->  POSITION F      RELATIVE POSITION                          *
*        = 0             NO NEW POSITION                            *
*        = - MAXINT      TO BEGINNING OF DATA SET                   *
*                        ( -2147483648  OR  X'80000000')            *
*        = + MAXINT      TO END OF DATA SET                         *
*                        ( +2147483647  OR  X'7FFFFFFF')            *
*        = - N           N RECORDS BACKWARD                         *
*        = + N           N RECORDS FORWARD                          *
*  NOTES:                                                           *
*        YOU CAN CREATE EMPTY SLOTS (GAPS) USING FORWARD POSITIONING *
*        IN A RELATIVE DATA SET IN OUTPUT MODE.                     *
*********************************************************************
*
*  SAVE REGISTERS AND LOAD PROGRAM REGISTER
*
         STM   R14,R12,12(R13)
         LR    R10,R15
```

```
*
*  LOAD PARAMETER
*
         LM    R1,R5,0(R1)
*
*  ADDRESS WORK AREA
*
         LR    R12,R1
         USING WORKAREA,R12
*
*  POSITION RECORD
*
*        .
*        .
*        .
*
*  HERE:  RETURN CODE -1  UNSUPPORTED FUNCTION
*
         LA    R0,0
         BCTR  R0,0
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R15,12(R13)
         BR    R14
*
*  RELEASE WORK AREA REGISTER
*
         DROP  R12
*
*********************************************************************
*  LOCAL CONSTANTS                                                  *
*********************************************************************
*
         LTORG
         DROP  R10
         TITLE 'FLAMUIO: DUMMY SECTIONS'
*********************************************************************
*  DUMMY SECTIONS                                                   *
*********************************************************************
*
*
WORKAREA DSECT
*********************************************************************
*  WORK AREA ON DOUBLE WORD BOUNDARY                                *
*********************************************************************
*
         DS    XL1024
*
LWORK    EQU   *-WORKAREA         LENGTH; MAXIMAL 1024 BYTES
         EJECT
```

```
*********************************************************************
*  DUMMY SECTION                                                    *
*********************************************************************
*
*
OPNPAR   DSECT
*********************************************************************
*  PARAMETERLIST FOR USROPN
*
*  NOTE:    ADDRESSES ARE GIVEN, NOT THE VALUES.
*********************************************************************
ADWORKA  DS    A                 WORK AREA
ADRETCO  DS    A                 RETCO
ADOPMO   DS    A                 OPENMODE
ADDDN    DS    A                 DDNAME
ADDSORG  DS    A                 DSORG
ADRECFO  DS    A                 RECFORM
ADRECSI  DS    A                 RECSIZE
ADBLKSI  DS    A                 BLKSIZE
ADKEYDE  DS    A                 KEYDESC
ADEVICE  DS    A                 DEVICE
ADRECDE  DS    A                 RECDELIM
ADPADC   DS    A                 PADCHAR
ADPRCTL  DS    A                 PRCNTRL
ADCLOSDI DS    A                 CLOSDISP
ADACC    DS    A                 ACCESS
ADDSNLEN DS    A                 LENGTH DSN
ADDSN    DS    A                 DATA SET NAME
         EJECT
*********************************************************************
*  DUMMY SECTION                                                    *
*********************************************************************
*
*
KEYDESC  DSECT
*
*  KEY DESCRIPTION
*
KEYFLAGS DS    F                 KEYFLAGS
KEYPARTS DS    F                 NUMBER OF KEYPARTS
KEYPOS1  DS    F                 KEYPOSITION OF 1. KEYPART
KEYLEN1  DS    F                 LENGTH      OF 1. KEYPART
KEYTYPE1 DS    F                 DATATYPE    OF 1. KEYPART
KEYPOS2  DS    F
KEYLEN2  DS    F
KEYTYPE2 DS    F
KEYPOS3  DS    F
KEYLEN3  DS    F
KEYTYPE3 DS    F
KEYPOS4  DS    F
KEYLEN4  DS    F
KEYTYPE4 DS    F
KEYPOS5  DS    F
KEYLEN5  DS    F
KEYTYPE5 DS    F
KEYPOS6  DS    F
KEYLEN6  DS    F
```

```
KEYTYPE6 DS    F
KEYPOS7  DS    F
KEYLEN7  DS    F
KEYTYPE7 DS    F
KEYPOS8  DS    F                KEYPOSITION OF 8. KEYPART
KEYLEN8  DS    F                LENGTH      OF 8. KEYPART
KEYTYPE8 DS    F                DATATYPE    OF 8. KEYPART
         END
```

5.3.2 COBOL example

The user I/O can also be implemented in COBOL or in another higher programming language. The following example implements two different functions that can be selected via the symbolic file name (LINKNAME or DDNAME).

Using the DD-name DATBASE ten records can be read with the content:

"THIS IS A DATA BASE RECORD FROM THE USER I/O"

Then return code END OF FILE is returned.

Using DD-name "USER..." 20 records can be read with the content:

"THIS IS A USER RECORD FROM THE USER I/O"

Then return code END OF FILE is returned.

In addition in both cases the call protocols are written to the terminal. This allows to observe precisely the sequence of the different calls.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  USERIO.
 AUTHOR.      LIMES DATENTECHNIK GMBH.
*
*  USERIO IS AN EXAMPLE FOR AN USER I/O MODULE TO CONNECT
*  TO FLAM.
*
*  THE PROGRAM IS WRITTEN TO SUPPORT 2 DIFFERENT DATA SETS IN
*  THE SAME MODULE DISTINGUISHED BY THE DD-NAME (DATBASE  OR
*                                                USER....)
*
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
*
 SPECIAL-NAMES.
     SYSOUT IS  OUT-PUT.
*
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
*
 77  ALL-OK               PIC S9(8) COMP VALUE 0.
 77  FUNCTION-ERR         PIC S9(8) COMP VALUE -1.
 77  REC-TRUNCATED        PIC S9(8) COMP VALUE 1.
 77  END-OF-FILE          PIC S9(8) COMP VALUE 2.
 77  REC-NOT-FOUND        PIC S9(8) COMP VALUE 5.
 77  NEW-HEADER           PIC S9(8) COMP VALUE 6.
 77  FILE-EMPTY           PIC S9(8) COMP VALUE 30.
 77  FILE-NOT-EXIST       PIC S9(8) COMP VALUE 31.
 77  OPEN-MODE-ERR        PIC S9(8) COMP VALUE 32.
 77  FILE-NAME-ERR        PIC S9(8) COMP VALUE 39.
```

```
*
 77  EXAMPLE-USER-RECORD  PIC X(72) VALUE
        "THIS IS A USER RECORD FROM THE USER I/O".
 77  EXAMPLE-DATBAS-RECORD PIC X(72) VALUE
        "THIS IS A DATA-BASE RECORD FROM THE USER I/O".
 77  RECLEN               PIC S9(8) COMP VALUE 80.
**************************************************
/
 LINKAGE SECTION.
*
 01  USER-WORK.
     03 W-DDNAME     PIC  X(8).
     03 W-COUNTER    PIC  S9(7) COMP-3.
     03 W-ELSE       PIC  X(1012).
 01  RETCO           PIC S9(8)  COMP.
 01  OPENMODE        PIC S9(8)  COMP.
     88  OP-INPUT    VALUE 0.
     88  OP-OUTPUT   VALUE 1.
 01  DDNAME.
     03 DDNAME-1     PIC  X(4).
     03 FILLER       PIC  X(4).
*
*  IN THIS EXAMPLE WE DO NOT NEED THE FOLLOWING PARAMETERS
*
*01  DSORG           PIC S9(8)  COMP.
*01  RECFORM         PIC S9(8)  COMP.
*01  RECSIZE         PIC S9(8)  COMP.
*01  BLKSIZE         PIC S9(8)  COMP.
*01  KEYDESC.
*    03 KEYFLAGS PIC S9(8) COMP.
*    03 KEYPARTS PIC S9(8) COMP.
*    03 KEYENTRY                 OCCURS 8 TIMES.
*       05 KEYPOS  PIC S9(8) COMP.
*       05 KEYLEN  PIC S9(8) COMP.
*       05 KEYTYPE PIC S9(8) COMP.
*01  DEVICE          PIC S9(8)  COMP.
*01  RECDELIM        PIC X(4).
*01  PADCHAR         PIC X.
*01  PRCTRL          PIC S9(8)  COMP.
*01  CLOSMODE        PIC S9(8)  COMP.
*01  ACCESS          PIC S9(8)  COMP.
*01  DSNLEN          PIC S9(8)  COMP.
*01  DATA-SET-NAME   PIC X(44).
*
*  USED FOR READING
*
 01  DATALEN         PIC S9(8)  COMP.
 01  DATA-AREA.
     03 DATA-1       PIC X(72).
     03 DATA-2       PIC X(8).
 01  BUFFLEN         PIC S9(8)  COMP.
*
```

```
/
 PROCEDURE DIVISION.
*
 USROPN-MAIN SECTION.
*
*  OPEN ROUTINE
*
 USROPN-MAIN-1.
     ENTRY "USROPN"  USING  USER-WORK, RETCO,
                            OPENMODE, DDNAME.
*
*  IN THIS EXAMPLE WE DO NOT USE THE OTHER PARAMETERS, SO IT IS
*  NOT NECESSARY TO MENTION THEM.
*  FLAM STANDARDS ARE USED:
*     SEQUENTIAL,
*     VARIABLE LENGTH UP TO 32752 BYTE (BUT WE ONLY USE 80 BYTE)
*
*
*  WE ONLY SUPPORT OPEN INPUT IN THIS EXAMPLE,
*  CHECK THE OPEN MODE
*
     IF  OP-INPUT
        THEN   NEXT SENTENCE
        ELSE   MOVE  OPEN-MODE-ERR  TO  RETCO
               DISPLAY "USER I/O CANNOT WRITE TO " DDNAME
                       UPON OUT-PUT
               GO TO USROPN-MAIN-99.
*
*  FOR FURTHER USE, WE STORE THE DD-NAME IN THE
*  GIVEN WORKAREA
*
     MOVE  DDNAME    TO   W-DDNAME.
*
*  WE SUPPORT DIFFERENT DATA SETS,
*  CHECK FOR DDNAME "DATBASE", OR THE FIRST 4 BYTE FOR "USER"
*
     IF  DDNAME  =  "DATBASE"
        THEN  PERFORM  OPN-DATBASE
        ELSE  IF  DDNAME-1 =  "USER"
               THEN  PERFORM  OPN-USER
               ELSE  MOVE FILE-NAME-ERR  TO  RETCO
                     DISPLAY "USER I/O DOES NOT SUPPORT " DDNAME
                             UPON  OUT-PUT.
 USROPN-MAIN-99.
*
*  GO BACK TO FLAM
*
     GO BACK.
/
 OPN-DATBASE SECTION.
*
*  OPEN-ROUTINE FOR A DATA BASE
*
 OPN-DATBASE-1.
*
*  HERE YOU HAVE TO PROCESS THE OPEN,
*
```

```
*
*  INITIALISE COUNTER-FIELD IN WORK AREA
*
     MOVE  ZERO     TO  W-COUNTER.
*
*  WE ONLY DISPLAY A MESSAGE
*
     DISPLAY "USER I/O: OPEN FOR DATABASE IS DONE"
             UPON  OUT-PUT.
 OPN-DATBASE-90.
*
*  SET THE RETURNCODE
*
     MOVE  ALL-OK   TO  RETCO.
 OPN-DATBASE-99.
     EXIT.
/
 OPN-USER SECTION.
*
*  OPEN ROUTINE FOR THE OTHER EXAMPLE
*
 OPN-USER-1.
*
*  HERE YOU HAVE TO PROCESS THE OPEN,
*
*  INITIALISE COUNTER-FIELD IN WORK AREA
*
     MOVE  ZERO     TO  W-COUNTER.
*
*  WE ONLY DISPLAY A MESSAGE
*
     DISPLAY "USER I/O:  OPEN FOR " DDNAME " IS DONE"
             UPON  OUT-PUT.
 OPN-USER-90.
*
*  SET THE RETURN CODE
*
     MOVE  ALL-OK   TO  RETCO.
 OPN-USER-99.
     EXIT.
/
 USRCLS-MAIN SECTION.
*
*  CLOSE ROUTINE
*
 USRCLS-MAIN-1.
     ENTRY "USRCLS"  USING  USER-WORK, RETCO.
*
*  WE SUPPORT DIFFERENT DATA SETS,
*  CHECK FOR DDNAME
*
     IF  W-DDNAME  =  "DATBASE"
        THEN  PERFORM  CLS-DATBASE
        ELSE  PERFORM  CLS-USER.
 USRCLS-MAIN-99.
*
*  GO BACK TO FLAM
*
```

```
     GO BACK.
/
 CLS-USER SECTION.
*
*  CLOSE ROUTINE FOR THE OTHER EXAMPLE
*
 CLS-USER-1.
*
*  HERE YOU HAVE TO PROCESS THE CLOSE,
*
*  WE ONLY DISPLAY A MESSAGE
*
     DISPLAY "USER I/O:  CLOSE FOR " W-DDNAME " IS DONE"
             UPON  OUT-PUT.
 CLS-USER-90.
*
*  SET THE RETURN CODE
*
     MOVE  ALL-OK   TO  RETCO.
 CLS-USER-99.
     EXIT.
/
 CLS-DATBASE SECTION.
*
*  CLOSE ROUTINE FOR A DATA BASE
*
 CLS-DATBASE-1.
*
*  HERE YOU HAVE TO PROCESS THE CLOSE,
*
*  WE ONLY DISPLAY A MESSAGE
*
     DISPLAY "USER I/O: CLOSE FOR DATA BASE IS DONE"
             UPON  OUT-PUT.
 CLS-DATBASE-90.
*
*  SET THE RETURNCODE
*
     MOVE  ALL-OK   TO  RETCO.
 CLS-DATBASE-99.
     EXIT.
/
 USRGET-MAIN SECTION.
*
*  ROUTINE FOR READING RECORDS
*
 USRGET-MAIN-1.
     ENTRY "USRGET"  USING  USER-WORK, RETCO,
                            DATALEN, DATA-AREA, BUFFLEN.
*
*  WE SUPPORT DIFFERENT DATA SETS,
*  CHECK FOR DDNAME
*
     IF  W-DDNAME  =   "DATBASE"
        THEN  PERFORM  GET-DATBASE
        ELSE  PERFORM  GET-USER.
 USRGET-MAIN-99.
*
```

```
*  GO BACK TO FLAM
*
     GO BACK.
/
 GET-DATBASE SECTION.
*
*  GET-ROUTINE FOR A DATA BASE
*
 GET-DATBASE-1.
*
*  WE RETURN ALWAYS THE SAME RECORD
*
*  AFTER THE 10. RECORD WE FINISH (EOF)
*
     IF  W-COUNTER    +10
        THEN MOVE  EXAMPLE-DATBAS-RECORD  TO  DATA-1
             MOVE  W-DDNAME               TO  DATA-2
             MOVE  RECLEN                 TO  DATALEN
             ADD   +1                     TO  W-COUNTER
             MOVE  ALL-OK                 TO  RETCO
        ELSE MOVE  ZERO          TO  DATALEN
             MOVE  END-OF-FILE   TO  RETCO.
 GET-DATBASE-99.
     EXIT.
/
 GET-USER SECTION.
*
*  GET ROUTINE FOR THE OTHER EXAMPLE,
*
 GET-USER-1.
*
*  WE RETURN ALWAYS THE SAME RECORD,
*
*  AFTER THE 20. RECORD WE FINISH (EOF)
*
     IF  W-COUNTER    +20
        THEN MOVE  EXAMPLE-USER-RECORD  TO  DATA-1
             MOVE  W-DDNAME             TO  DATA-2
             MOVE  RECLEN               TO  DATALEN
             ADD   +1                   TO  W-COUNTER
             MOVE  ALL-OK               TO  RETCO
        ELSE MOVE  ZERO          TO  DATALEN
             MOVE  END-OF-FILE   TO  RETCO.
 GET-USER-99.
     EXIT.
```

5.4 How to use the user exits

5.4.1 EXK10/EXD10-user exits

The following exit routine can be used for compression as well as for decompression and has the purpose of modifying fields within records.

The sample code is found in the library FLAM.SRCLIB.

```
         TITLE 'SEPARATE: EXIT FOR FLAM COMPRESSION'
SEPARATE CSECT
SEPARATE AMODE ANY
SEPARATE RMODE ANY
*********************************************************************
*        THIS PROGRAM SEPARATES FIELDS WITHIN RECORDS WHICH CAN
*        BE SEPARATED BY DELIMITER CHARACTERS INTO DIFFERENT FLAM RECORDS.
*        THIS ENABLES A BETTER COMPRESSION.
*        THE DESIGN OF THE PROGRAM ALLOWS TO MODIFY THE DELIMITER AND
*        EVEN THE LENGTH OF THE DELIMITER BY CHANGING ONLY ONE STATEMENT.
*
*        THE DELIMITERS ARE REMOVED FROM THE RECORD AND ARE REPLACED
*        BY FLAM SYNTAX.
*        IF THE RECORD DOES NOT CONTAIN DELIMITERS IT IS PASSED TO
*        FLAM WITHOUT MODIFICATIONS.
*
*        THE ROUTINE SEPARATE IS ACTIVATED VIA PARAMETER 'EXK10=SEPARATE'
*        DURING THE CALL OF FLAM OR FLAMUP.
*
*        THE FIELDS CONSIST OF PRINTABLE CHARACTERS SEPARATED BY
*        A TWO BYTE LONG DELIMITER (X'0D25').
*
*        THE DATA COMPRESSED IN THIS WAY IS TRANSMITTED VIA
*        FILE TRANSFER TO A PC, DECOMPRESSED FIELD BY FIELD
*        USING FLAM AND WRITTEN TO THE STORAGE MEDIUM (WITH
*        DELIMITER OF THE OPERATING SYSTEM,  X'0D0A' WITH MSDOS
*        OR X'0A' WITH UNIX).
*
*  NOTE:
*
*        FOR DECOMPRESSION ON A MAINFRAME A FILE WITH
*        VARIABLE RECORD LENGTH MUST BE SPECIFIED.
*        EACH FIELD SEPARATED DURING COMPRESSION IS WRITTEN
*        AS A SEPARATE RECORD. THE DELIMITERS ARE NOT CONTAINED
*        IN THE DECOMPRESSED RECORDS.
*        THIS MEANS THAT THE ORIGINAL FILE CANNOT BE RECONSTRUCTED
*        ON A MAINFRAME.
*
*        THIS MODULE IS REENTRANT AND REUSABLE
*
*
*--------------------------------------------------------------------
*
*  AUTHOR:          LIMES DATENTECHNIK GMBH
*                   PHILIPP-REIS-PASSAGE 2
*                   D-61381 FRIEDRICHSDORF/TS.
```

```
*                  TEL. 06172-5919-0
*                  FAX  06172-5919-39
*********************************************************************
*
*  INTERFACE:  R1 POINTS TO PARAMETER LIST
*
*   0(R1)   -   A(FUNCTION CODE)
*   4(R1)   -   A(RETURN CODE)
*   8(R1)   -   A(A(RECORD))  RECORD POINTER
*  12(R1)   -   A(RECORD LENGTH)
*  16(R1)   -   A(WORK AREA)       NEW WITH FLAM V2.5
*
*********************************************************************
         EJECT
         STM   R14,R12,12(R13)    SAVE REGISTERS
         LR    R12,R15            ENTRY ADDRESS USED AS PROGRAM BASE
         USING SEPARATE,R12       ASSIGN BASE REGISTER
         USING WORKAREA,R2        BASE REGISTER WORK AREA
         LA    15,0               INITIALISE RETURN CODE WITH 0
*
         L     R3,0(,R1)          LOAD A(FC)
         CLC   0(4,R3),FCSATZ     PASS RECORD ?
         BE    SATZUEB            == YES
         CLC   0(4,R3),FCOPEN     OPEN ?
         BNE   RET                == NO
*
*  AT OPEN TIME RESET WORK AREA FIELDS
*
         L     R2,16(,R1)        A(WORKAREA)
         MVI   FLAG,X'00'        RESET FLAGS
         B     RET
SATZUEB  DS    0H
*
*        RECORD WAS PASSED
*
         L     R10,8(,R1)        A(A(RECORD)) TO R10
         L     R4,0(,R10)        LOAD A(RECORD)
         L     R11,12(,R1)       A(RECORD LENGTH)
         L     R5,0(,R11)        LOAD RECORD LENGTH
         LA    R9,0(R5,R4)       A(RECORD END)
         L     R2,16(,R1)        A(WORK AREA)
*
         TM    FLAG,SATZDA       RECORD ALREADY PRESENT ?
         BNO   BEGINN            == NO
         TM    FLAG,LOESCH       DELETE RECORD ?
         BO    LOESATZ           == YES
*
BEGINNA  DS    0H                RECORD WAS ALREADY PROCESSED
         L     R4,SATZPTR        A(FIELD) FROM LAST TIME
*
BEGINN   DS    0H
         OI    FLAG,SATZDA       INDICATE RECORD PRESENT
         LR    R7,R4             SAVE A(FIELD BEGIN)
         LR    R6,R9             A(FIELD END)
         SR    R6,R7             - A(FIELD BEGIN) = L'REMAINDER
         BZ    LEERSATZ          L'= 0, PASS EMPTY RECORD
         C     R6,LTRENNKZ
         BNL   SUCH              L'  L'DELIMITER - HAS NO DELIMITER
```

```
         OI    FLAG,LOESCH       INDICATE DELETE OPERATION FOR NEXT RUN
         LR    R4,R9             A(RECORD END)
         B     SUCHEND
SUCH     DS    0H
         LA    R8,1              INCREMENT FOR BX INSTRUCTION
         S     R9,LTRENNKZ       FOR BX INSTR. SET RECORD END -L'
SUCHLOOP DS    0H
*
*  SEARCH STRING IS (DELIMITER)
*
         CLC   0(L'TRENNKZ,R4),TRENNKZ   DELIMITER ?
         BE    ISTDA             == YES
         BXLE  R4,R8,SUCHLOOP    NEXT CHARACTER
*
         OI    FLAG,LOESCH       INDICATE DELETE OPERATION FOR NEXT RUN
         LA    R4,L'TRENNKZ-1(R4) FIELD IS BIGGER BY L'-1
         B     SUCHEND
*
ISTDA    DS    0H
         LA    R6,L'TRENNKZ(R4)  INCREMENT RECORD POINTER
         ST    R6,SATZPTR        SAVE RECORD POINTER
SUCHEND  DS    0H
*
*  FILL FLAM PARAMETER LIST
*
         SR    R4,R7             FIELD LENGTH
         ST    R4,0(R11)         IS RECORD LENGTH FLAM
         ST    R7,0(R10)         RECORD ADDR FOR FLAM
         LA    R15,8             RETURN CODE: INSERT RECORD
*
RET      DS    0H
*
*        RETURN TO FLAM
*
         L     R3,4(,R1)          LOAD A(RC)
         ST    R15,0(,R3)         PASS RC
         L     R14,12(R13)        RESTORE REGISTERS
         LM    R0,R12,20(R13)
         BR    R14                RETURN
*
LOESATZ  DS    0H
         LA    R15,4              RETURN CODE: DELETE RECORD
         MVI   FLAG,X'00'         RESET FLAG
         B     RET                AND RETURN
*
LEERSATZ DS    0H                 AFTER DELIMITER AT RECORD END
         OI    FLAG,LOESCH        INDICATE DELETE OPERATION FOR NEXT RUN
         LA    R4,0               RECORD IS EMPTY
         ST    R4,0(R11)          RECORD LENGTH FOR FLAM
         LA    R15,8              RETURN CODE: INSERT RECORD
         B     RET                AND RETURN
*
*  CONSTANTS AND WORK AREAS
*
*
FCSATZ   DC    F'4'               FUNCTION CODE RECORD PASSED
FCOPEN   DC    F'0'                          OPEN
LTRENNKZ DC    A(L'TRENNKZ)       LENGTH OF DELIMITER
```

```
*-------------------------------------------------------------------
*
*  IN CASE OF DIFFERENT DELIMITER MAKE MODIFICATIONS HERE
*
TRENNKZ  DC    XL2'0D25'          DELIMITER TO BE FOUND
*-------------------------------------------------------------------
*
*  REGISTER
*
R0       EQU   0
R1       EQU   1                  PARAMETER ADDRESS
R2       EQU   2                  BASE REGISTER FOR WORK AREA
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12                 BASE REGISTER
R13      EQU   13                 A(SAVE AREA)
R14      EQU   14                 RETURN ADDRESS
R15      EQU   15                 ENTRY ADDRESS
*
         LTORG
*
         DC    C'*** MODULE SEPARATE V1.02 FOR FLAM '
         DC    C' COPYRIGHT (C) 1990-91 BY LIMES DATENTECHNIK GMBH. '
         DC    C'DATE, TIME ASSEMBLED: '
         DC    C'&SYSDATE , &SYSTIME '
         DC    C'***'
*
*  WORKAREA IS PROVIDED BY FLAM (1024 BYTES)
*
WORKAREA DSECT
*
DDNAME   DS    CL8                DD-NAME OF CURRENT FILE
SATZPTR  DS    A                  RECORD POINTER
FLAG     DS    X                  INDICATORS FOR PROCESSING
SATZDA   EQU   1                  RECORD ALREADY PRESENT
LOESCH   EQU   2                  DELETE RECORD
         END
```

5.4.2 EXK20/EXD20-user exits

Since FLAM protects compressed files against manipulations by applying a checksum, it is possible to provide encryptions within the user exits with a very low overhead.

Because the compressed data is already encrypted, simple deterministic character swapping within the compressed data cannot be detected easily by an unauthorized user.

During decompression this character swapping - if not redone by an authorized user - will lead to a check sum error and the compressed file cannot be read.

The symmetric construction of the user exits allows to use the same routine for encryption as well as for decryption provided that algorithms are used that will restore the original data when executed twice. This is the case with mutual character swapping.

Similar results can be obtained with translate tables for cyclic (cycle length 2) character code exchange.

```
         TITLE 'EX20 (B) | VERSION 1.00:06/25/91 | '
**********************************
* COLUMBUS-ASSEMBLER             *
**********************************
* SYMBOLIC CONDITIONS FOR #IF,#WHEN,#WHIL(E),#TOR,#AND,#OR
#LT      EQU   4   LESS THAN
#GT      EQU   2   GREATER THAN
#EQ      EQU   8   EQUAL
#NE      EQU   7   NOT EQUAL
#LE      EQU   13  LESS OR EQUAL
#GE      EQU   11  GREATER OR EQUAL
#LZ      EQU   4   LESS THAN ZERO
#GZ      EQU   2   GREATER THAN ZERO
#ZE      EQU   8   ZERO
#NZ      EQU   7   NOT ZERO
#ON      EQU   1   ONES
#MI      EQU   4   MIXED
#ZO      EQU   11  ZEROS OR ONES
#ZM      EQU   14  ZEROS OR MIXED
#OM      EQU   7   ONES OR MIXED
#F       EQU   15  TRUE IN ANY CASE
* FLOATING POINT REGISTERS, GENERAL REGISTERS, COLUMBUS REGISTERS
FA       EQU   0
FB       EQU   2
FC       EQU   4
FD       EQU   6
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
```

```
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
R#PAR    EQU   R1
R#BASE   EQU   R10
R#STACK  EQU   R13
R#EXIT   EQU   R14
R#PASS   EQU   R15
         EJECT
EX20     CSECT
         USING EX20,R#PASS
*********************************************************************
*  NAME: EX20                                      VERSION: 13.03.91 *
*  FUNCTION:                                                         *
*        FLAMFILE IS ENCRYPTED AND DECRYPTED IN A SIMPLE WAY.        *
*                                                                    *
*        THE 16TH AND 17TH CHARACTER ARE SWAPPED WHICH               *
*        CHANGES THE CHECKSUM. THE COMPRESSED DATA                   *
*        CAN ONLY DECOMPRESSED IF THE SWAP OPERATION                 *
*        DONE AGAIN.                                                 *
*        PARAMETER                                                   *
* 1 ->   ID       F    IDENTIFICATION                                *
* 2 <-   RETCO    F    RETURN CODE                                   *
* 3 ->   RECPTR   A    RECORD POINTER                                *
* 4 ->   RECLEN   F    RECORD LENGTH                                 *
*********************************************************************
*
*  SAVE REGISTERS AND LOAD BASE REGISTERS
*
         STM   R14,R12,12(R13)
*
*  LOAD PARAMETERS
*
         LM    R1,R4,0(R1)
*  PASS COMPRESSED RECORD
         CLC   0(4,R1),F4
         BC    #F-#EQ,#F1001
*  LOAD RECORD LENGTH
         L     R4,0(R4)
*  RECORD LENGTH GREATER 16
         LA    R14,16
         CR    R4,R14
         BC    #F-#GT,#F1002
*
*  SWAP 16TH AND 17TH CHARACTER
*
         L     R3,0(R3)
         LA    R14,0(R3,R14)
         IC    R5,0(R14)
         MVC   0(1,R14),1(R14)
         STC   R5,1(R14)
```

```
#F1002   DS    0H
#F1001   DS    0H
*
*  RETURN CODE = ACCEPT RECORD / NO ERROR
*
         LA    R0,0
         ST    R0,0(R2)
*
*  RETURN
*
         LM    R14,R12,12(R13)
         BR    R#EXIT
*
*  LOCAL CONSTANTS
*
F4       DC    F'4'
F16      DC    F'16'
         LTORG
         DS    0D
         DROP  R#PASS
         END
```

5.5 Using FLAM with other products

5.5.1 Integration with NATURAL

The necessary software for integration between NATURAL and FLAM was developed in cooperation with Software AG in Darmstadt.

Beginning with version 2.2, NATURAL is able to write and read it's workfiles using FLAM. This allows the user to create and process compressed files directly using NATURAL programs in a transparent way. This new facility also supports file formats that were not allowed as workfiles under NATURAL until now (VSAM files).

The decision about using the FLAM access method for a NATURAL workfile is made via JCL. No changes to NATURAL programs are necessary.

The linkage module NATFLAM is part of the FLAM installation tape for all /390 systems. It must be linked with the corresponding module from Software AG.

For further information please refer to your distributor or contact directly the manufacturers Software AG or limes datentechnik gmbh.

5.5.2 Integration with SIRON

In cooperation with Ton Beller AG in Bensheim (Germany) a FLAM access module was developed for the product SIRON®.

This allows to create and process compressed FLAM files using SIRON queries.

Only slight changes are necessary for the SIRON queries.

No changes in the queries are necessary if an entry for FLAM is made within GENAT for the corresponding files.

JCL changes are not necessary.

One possibility is to use the NIMM interface:

HOLE file (NIMM=HZFLAM), LIES file (NIMM=HZFLAM),

SCHREIBE file ... (NIMM=HZFLAM)

The other possibility is to specify FLAM within the GENAT entry for the DD name of the file:

HIN ddname ... MODUL='HZFLAM'

By using this GENAT entry, data is compressed or decompressed with each access automatically.

The necessary module HZFLAM is distributed by Ton Beller AG. It must be linked to the FLAM modules.

For further information please refer to your distributor or contact directly the manufacturers Ton Beller GmbH or limes datentechnik gmbh.

FLAM (MVS)

User Manual

Chapter 6:

Installation

Content

6. Installation

6.1 FLAM licence

FLAM is protected against unauthorized use. The authorized usage of FLAM is only possible with a licence number provided by limes datentechnik.

A licence number allows the usage of FLAM on one or multiple computers.

To obtain a licence number you need the name of the host and the serial number of the CPU. This information can be obtained with the installation programs.

There is a difference between test licences for a limited period and unlimited production licences.

A test licence allows the testing and benchmarking of FLAM with all functions for a defined period (e.g., 30 days).

- Test programs must not be given to a third party.
- During the test period no backup copies from the test. programs are allowed.
- After the test period expired all test programs must be deleted.

A production licence allows the unlimited usage of FLAM on each computer for that a licence was obtained.

FLAM has a lock built in, that recognizes and inhibits unauthorized access. Copying FLAM from one computer to another computer is not allowed and is inhibited by the program.

The licence protection mechanisms were developed according to the principles of computer centre operation. For that reason a technically possible usage not according to the licence contracts cannot be regarded as a valid usage under the existing licence.

FLAM compresses structure oriented using an algorithm that is a part of the Frankenstein-Limes method. This method has been patented in the Federal Republic of Germany and the United States of America and at the European Patent Bureau, registered by the inventor at the 19.7.1985.

FLAM®, FLAMFILE® and limes datentechnik® are registered trademarks.

6.2 Component list

FLAM comes with the following components:

FLAM.INSTALL	Installation procedure
FLAM.LOAD	Load library with FLAM modules
FLAM.OBJ	Library with object modules
FLAM.JOBLIB	JCL Library for examples and installation
FLAM.SRCLIB	Library with source code examples (see FLAM manual, chapter 5)
FLAM.PANELS	Library for FLAM panels
FLAM.CLIST	Library for FLAM CLIST procedures
FLAM.SKELS	Library for FLAM skeletons
FLAM.MSG	Library for FLAM messages

The content of the libraries may vary depending on the maintenance level of the actual delivery.

Each delivery comes with a table of contents of the installation tape.

6.3 Installation of FLAM

Usually FLAM is delivered on a CD-ROM. README text files are included as guides for installation.

Manuals are stored as RTF- and PDF-documents for reading on the appropriate system (Windows, Unix, …).

The data for the MVS/ZOS operating system are stored as a FLAMFILE. A binary file transfer (without any CRLF or ASCII translation) and decompression with FLAM stores all libraries and data to disk.

For new user (without having FLAM) a special installation file is provided.

6.4 Generation of default parameters

FLAM can be adapted easily to specific tasks by supplying it with suitable parameters. Many tasks will have similar characteristics so that the parameters will be equal. For that reason FLAM allows to specify default parameters that are used with each execution of a FLAM module.

The default parameters are stored within a module (FLAMPAR) in the load library.

It is only necessary to regenerate that module if the delivered module FLAMPAR with it's predefined default values shall not be used.

After the default parameters have been changed FLAM (and also user programs calling FLAM) must be linked again.

This allows to use different sets of parameters for different applications.

The program FLAMGEN can be used to change the default parameters. To do so, enter the modified parameters according to the FLAM syntax into the file GENPAR (see chapter PARAMETER, page ...). The parameters are then generated into module FLAMPAR by the program FLAMGEN. It is not necessary to assemble the module FLAMPAR. Parameters that were not specified in file GENPAR are not changed and keep their old value.

Entries made in the PARM=... instruction are used for the control of FLAMGEN (like SHOW=..., MSGDISP=...). These entries are not used as FLAM parameters.

The specification 'INFO=HOLD,MSGDISP=MSGFILE' as a PARM entry forces FLAMGEN to display the generated default parameters. For job control reasons FLAMGEN will terminate with condition code 4 in this case.

The procedure FLAM.JOBLIB(INST02) contains the JCL for parameter generation. It must be adapted to the user requirements.

Example:

```
-------- JOB09128  IEF097I FLAM25I2 - USER FLAM27   ASSIGNED
11.17.55 JOB09128  ICH70001I FLAM27   LAST ACCESS AT 11:15:50 ON TUESDAY, JULY
11.17.55 JOB09128  $HASP373 FLAM27I2 STARTED - INIT  A - CLASS A - SYS
11.17.55 JOB09128  IEF403I FLAM27I2 - STARTED - TIME=11.17.55
11.17.56 JOB09128  -                                   --TIMINGS (MINS.)--
11.17.56 JOB09128  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB
11.17.56 JOB09128  -FLAM27I2 STEP1                00    197    187    .00
11.18.03 JOB09128  -FLAM27I2 STEP2                00    606   1440    .00
11.18.03 JOB09128  -FLAM27I2 STEP3                04     22     74    .00    .
11.18.03 JOB09128  IEF404I FLAM27I2 - ENDED - TIME=11.18.03
11.18.03 JOB09128  -FLAM27I2 ENDED.  NAME-LIMES-06172/5919-0   TOTAL TCB CPU TI
11.18.03 JOB09128  $HASP395 FLAM27I2 ENDED
0
1 //FLAM27I2 JOB XXXXXXXX,'LIMES-06172/5919-0',CLASS=A,TIME=(,8),
  //            MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=FLAM27
  ***********************************************************
  ***   GENERATION OF FLAM DEFAULT PARAMETER *   INST02     *
  ***-----------------------------------------------------*
  ***                                                     *
  ***  ALL FLAM PARAMETER CAN BE ALTERED TO YOUR DEFAULT    *
  ***  VALUE.                                             *
  ***  THE NOT GIVEN PARAMETER REMAIN AS THEY WERE BEFORE.  *
  ***                                                     *
  ***  INFO=HOLD AS PARM-VALUE FOR FLAMGEN DISPLAYS THE     *
  ***  ACTUAL FLAM DEFAULT PARAMETER.                     *
  ***                                                     *
  ***  THE JOB CONTAINS THE FOLLOWING STEPS:              *
  ***                                                     *
  ***    1. GENERATES NEW DEFAULT PARAMETER               *
  ***    2. LINKS NEW MODULES                             *
  ***    3. SHOWS THE GENERATED PARAMETER                 *
  ***                                                     *
  ***********************************************************
  ***-------------------------------------------------------
  ***  STEP 1:  ALTER DEFAULT PARAMETER
  ***-------------------------------------------------------
2 //STEP1    EXEC PGM=FLAMGEN
3 //STEPLIB  DD   DSN=FLAM27.FLAM.LOAD,DISP=SHR
4 //FLPRINT  DD   SYSOUT=*
5 //FLAMOBJ  DD   DSN=&&GENDAT,DISP=(NEW,PASS),
  //             SPACE=(80,(200,100)),UNIT=SYSDA
  ***
  *** THIS DATA SET CONTAINS YOUR NEW DEFAULT PARAMETER:
  ***
6 //GENPAR   DD   *
```

```
  ***-------------------------------------------------------------
  *** STEP 2:  LINK  MODULES
  ***          (AMODE, RMODE ARE ALLOWED TO CHANGE TO YOUR USAGE)
  ***-------------------------------------------------------------
  ***     FLAMPAR, FLAMREC, FLAMUP                               *
  ***-------------------------------------------------------------
7 //STEP2    EXEC PGM=HEWL,PARM='RENT,REUS,LIST,MAP',
  //             COND=(4,LT,STEP1)
8 //SYSPRINT DD   SYSOUT=*
9 //SYSUT1   DD   DSN=&&SYSUT1,SPACE=(1024,(200,40)),
  //            UNIT=SYSDA
  ***  FOR AUTOMATIC CALL:
10 //SYSLIB   DD   DSN=*.STEP1.STEPLIB,DISP=SHR
   ***  OUTPUT MODULE LIBRARY:
11 //SYSLMOD  DD   DSN=*.STEP1.STEPLIB,DISP=SHR
   ***  SECONDARY INPUT DATA SETS:
12 //GENOBJ   DD   DSN=&&GENDAT,DISP=(OLD,PASS)
13 //FLMOBJ   DD   DSN=FLAM27.FLAMV27.OBJ,DISP=SHR
   ***  PRIMARY INPUT DATA SET:
14 //SYSLIN   DD   *
   ***-------------------------------------------------------
   ***  STEP 3:  SHOW GENERATED PARAMETER
   ***-------------------------------------------------------
15 //STEP3    EXEC PGM=FLAMGEN,PARM='INFO(HOLD),MSGDISP(MSGFILE)'
16 //STEPLIB  DD   DSN=*.STEP1.STEPLIB,DISP=SHR
17 //FLPRINT  DD   SYSOUT=*
18 //GENPAR   DD   DUMMY

FLM0448 COPYRIGHT (C) 1989-1999 BY LIMES DATENTECHNIK          *)
FLM0448 ACCESS  =LOG      BLKMODE =YES      CLIMIT  =       0
FLM0448 MODE    =CX8      CODE    =EBCDIC   FILEINFO=YES
FLM0448 HEADER  =YES      INFO    =YES      KEYDISP =OLD
FLM0448 LOOP    =NO       MAXBUFF =   32768 MAXREC  =     255
FLM0448 MAXSIZE =     512 MSGDISP =MSGFILE  NAMEDISP=NEW
FLM0448 OPENMODE=OUTPUT   TRUNCATE=NO       TRANSLAT=
FLM0448 EXD10   =         EXD20   =         EXK10   =
FLM0448 EXK20   =         FLAMDDN =FLAMFILE IDDN    =FLAMIN
FLM0448 ODDN    =FLAMOUT  MSGDDN  =FLPRINT  PARDDN  =FLAMPAR
FLM0448 CLOSDISP=REWIND   DSORG   =SEQUENT  RECFORM =FIXBLK
FLM0448 KEYLEN  =       8 BLKSIZE =    6144 DEVICE  =DISK
FLM0448 ICLOSDIS=REWIND   IDSORG  =SEQUENT  IRECFORM=VARBLK
FLM0448 IRECSIZE=   32752 IRECDEL =00000000 IKEYPOS =       1
FLM0448 IKEYLEN =       8 IBLKSIZE=   32760 IDEVICE =DISK
FLM0448 OCLOSDIS=REWIND   ODSORG  =SEQUENT  ORECFORM=VARBLK
FLM0448 ORECSIZE=   32752 ORECDEL =00000000 OKEYPOS =       1
FLM0448 OKEYLEN =       8 OBLKSIZE=   32760 ODEVICE =DISK
FLM0448 FLAMFILE=
FLM0448 FLAMIN  =
FLM0448 FLAMOUT =
FLM0448 MSGFILE =
FLM0448 PARFILE =
FLM0428 RECEIVED: INFO=YES,MSGDISP=MSGFILE,PARDDN=GENPAR      **)
FLM0410 DATA SET NAME : JES2.JOB09128.I0000101                ***)
FLM0428 RECEIVED:   MODE(CX8),MAXBUFFER(1)                    ***)
FLM0440 FLAM COMPRESSION NORMAL END
             .
             .      Messages of STEP2 (Linkage Editor)
```

.

```
FLM0448 COPYRIGHT (C) 1989-1999 BY LIMES DATENTECHNIK GMBH
FLM0448 ACCESS  =LOG      BLKMODE =YES      CLIMIT  =       0
FLM0448 MODE    =CX8      CODE    =EBCDIC   FILEINFO=YES
FLM0448 HEADER  =YES      INFO    =YES      KEYDISP =OLD
FLM0448 LOOP    =NO       MAXBUFF =   32768 MAXREC  =     255
FLM0448 MAXSIZE =     512 MSGDISP =MSGFILE  NAMEDISP=NEW
FLM0448 OPENMODE=OUTPUT   TRUNCATE=NO       TRANSLAT=
FLM0448 EXD10   =         EXD20   =         EXK10   =
FLM0448 EXK20   =         FLAMDDN =FLAMFILE IDDN    =FLAMIN
FLM0448 ODDN    =FLAMOUT  MSGDDN  =FLPRINT  PARDDN  =FLAMPAR
FLM0448 CLOSDISP=REWIND   DSORG   =SEQUENT  RECFORM =FIXBLK
FLM0448 KEYLEN  =       8 BLKSIZE =    6144 DEVICE  =DISK
FLM0448 ICLOSDIS=REWIND   IDSORG  =SEQUENT  IRECFORM=VARBLK
FLM0448 IRECSIZE=   32752 IRECDEL =00000000 IKEYPOS =       1
FLM0448 IKEYLEN =       8 IBLKSIZE=   32760 IDEVICE =DISK
FLM0448 OCLOSDIS=REWIND   ODSORG  =SEQUENT  ORECFORM=VARBLK
FLM0448 ORECSIZE=   32752 ORECDEL =00000000 OKEYPOS =       1
FLM0448 OKEYLEN =       8 OBLKSIZE=   32760 ODEVICE =DISK
FLM0448 FLAMFILE=
FLM0448 FLAMIN  =
FLM0448 FLAMOUT =
FLM0448 MSGFILE =
FLM0448 PARFILE =
FLM0440 FLAM COMPRESSION NORMAL END
```

*) Display of the old default parameters.

**) Here the parameters specified for FLAMGEN are recorded.

***) The FLAM parameters as read from file JES2.JOB09128.I0000101 are recorded. Because direct entry was used (GENPAR DD *) the job name generated by JES is displayed.

FLAM (MVS)

User Manual

Chapter 7:

Technical data

Content

7. Technical data

7.1 System environment

FLAM can be executed under the operating systems MVS/XA, MVS/ESA, OS/390 and z/OS from IBM.

FLAM does not need authorization and does not need to be started out of an authorizised library.

FLAM is independent from the address mode (24- or 31 bit) and from the load address (upper/lower address space). But you cannot use data from above the bar (2 GB).

Because of compatibility (calling application modules may reside in the lower address space) FLAM modules are loaded into the lower address space. The address mode is inherited from the calling program.

As an option, FLAM can be configured to be loaded into the upper address space (see installation procedure FLAM.JOBLIB (INST02)).

If FLAM runs in the upper address space, it is still possible to access non-VSAM files via FLAM.

Compressed files created with earlier version of FLAM can be decompressed with this version. Within version 3 FLAM is upwards as well as downwards compatible always supporting the functional range of the lower version.

7.2 Memory requirements

The components of FLAM require static memory for object code. Additional dynamic memory requests are issued for variables and working areas. Additionally the operating system will allocate I/O buffers for files. The listed values are approximated.

	static	dynamic	matrix
FLAM / FLAMUP with subroutines	360 KB	80-160 KB	6-5300 KB
record level interface with subroutines	290 KB	60-140 KB	6-5300 KB
BIFLAMK	30 KB		
BIFLAMD	30 KB		

The dynamic memory needed depends on the length of the records to be processed and the file access method.

All memory requests allocate memory below or above the 16 MB line according to the current address mode.

7.3 Performance

The following benchmarks can give a clue what compression effects can be achieved:

Typical user files (like FIBU, MATDAT) 70 - 90%

Diverse listings (like ASSEMBLER listings) 65 - 85%

Electronic data interchange files (DTAUS) 70%

In principle the compression effect depends on the file and record structure as well as on the actual data. Also the compression mode and the parameters specified have an influence.

7.4 Statistics

If the parameter SHOW=ALL is specified FLAM and FLAMUP will display statistical data concerning the execution of compression or decompression.

FLAM calculates and displays the number of records and bytes and the compression ratio. During compression the number of records and bytes is calculated both for input and output, and the compression ratio is calculated as the relation between number of input bytes and number of output bytes, expressed as a percentage. The number of bytes is calculated from the net (true) lengths of the data records, i.e. not taking the record length field into account.

The compression effect is always computed as the relation between input bytes and output bytes.

If user exits modify the record number or record length the statistics may not be correct.

During decompression the number of records and bytes on the FLAMFILE is evaluated. Also the number of decompressed records and the number of bytes in these records is displayed. The number obtained during compression and decompression are identical if no user exits are used.

FLAM also displays the elapsed time of the process. This includes the mounting times during tape I/O as well. In addition the CPU time used is displayed. Files can be separated during decompression.

When group files are being compressed or decompressed, intermediate statistics comprising the number of records and bytes of the original and compressed records are displayed for all partially compressed data.

At the end of a group file, overall statistics comprising the number of records and bytes, the compression effect and the time values are displayed. The file name of the compressed file is repeated before these overall statistics; if necessary, a message is displayed informing the user that not all files could be processed.

When group files are being decompressed, only the number of records and bytes in the compressed records that have been processed are listed in the overall statistics; the values for the original records are only listed in the intermediate statistics for the individual files. When a file set is being processed, the statistics are displayed separately for each file. Only the time values are displayed altogether at the end of the program run.

FLAM (MVS)

User Manual

Chapter 8:

Messages

Content

8. Messages

8.1 Messages from the Utility

Messages are only printed by the FLAM utility or by the subprogram FLAMUP. No messages are printed by the record level interface FLAMREC.

With parameter MSGDISP it is possible to control the output medium for messages.

MSGDISP=TERMINAL Currently not supported.

MSGDISP=MSGFILE The messages are written into a catalogued file. The DDNAME of this file is FLPRINT by default and can be modified using parameter MSGDDN=<name>.

MSGDISP=SYSTEM The messages are issued using the WTO macro to the operator console (route code 11).

8.2 Message Listing

FLAM messages

FLM0400 **FLAM COMPRESSION VERSION ... ACTIVE**

Explanation The FLAM compression system was activated. FLAM means: Frankenstein-Limes-Access-Method.
FLAM is a registered trademark ™ copyright © by limes datentechnik® gmbh.

Action None.

FLM0401 **PARAMETER REJECTED. INVALID VALUE: ...**

Explanation The specified parameter has an invalid value.

Action Correct parameter according to FLAM documentation and start again.

FLM0402 **PARAMETER REJECTED. SYNTAX ERROR**

Explanation The command was rejected because it contained a syntax error. The wrong command was protocolled with message FLM0428.

Action Enter command with corrected syntax.

FLM0403 **PARAMETER REJECTED. INVALID KEYWORD**

Explanation The command was rejected because it contained an invalid keyword. The valid keywords and their abbreviations are documented in the interface documentation.

Action Correct the invalid keyword and start again.

FLM0404 **PARAMETER REJECTED. PARAMETER VALUE NOT DECIMAL**

Explanation The command was rejected because it contained a non numeric operand where a numeric operand was expected. The wrong command was protocolled with message FLM0428.

Action Repeat the command using a numeric operand.

FLM0405 **PARAMETER REJECTED. OPERAND IS TOO LONG**

Explanation The command was rejected because the value of an operand was too long. The wrong command was protocolled with message FLM0428.

Action Repeat the command with corrected value assignment.

FLM0406 **INPUT RECORDS / BYTES: ...**

Explanation Number of records and bytes compressed with FLAM.

Action None.

FLM0407 **OUTPUT RECORDS / BYTES: ...**

Explanation Number of records and bytes in the compressed file (FLAMFILE).

Action None.

FLM0408 **CPU - TIME: ...**

Explanation CPU-time consumed by FLAM for compression.

Action None.

FLM0409 **RUN - TIME: ...**

Explanation Elapsed time for compression with FLAM.
This includes the time needed for tape mounting.

Action None.

FLM0410 **DATA SET NAME: ...**

Explanation Name of the file compressed with FLAM (FLAMIN) and of the compressed file (FLAMFILE) or of the parameter file (FLAMPAR).

Action None.

FLM0411 **DATA SET ORGANIZATION NOT SUPPORTED**

Explanation The input file is not compressed because FLAM does not support this file organization.

Action Assign a file that is supported by FLAM.

FLM0413 **COMPRESSION ERROR CODE: ...**

Explanation Compression aborted. Explanation of error code:

15 =	Record length greater than 32764 or negative
16 =	Record length greater than matrix size -4
20 =	Illegal open mode
21 =	Illegal size of matrix buffer
22 =	Illegal compression method
23 =	Illegal code in FLAMFILE
24 =	Illegal MAXREC specification
25 =	Illegal record length
26 =	Illegal character code
40 =	Module or table cannot be loaded
41 =	Module cannot be called
42 =	Module cannot be unloaded
43 - 49 =	Abortion caused by exit routine
98 =	Not all files were processed

Action Usually, invalid parameters have been transferred for FLAM (see chapter 3). Correct these parameters.

The error codes 15, 16, 25 and 40 - 49 are self explanatory.

For other error-codes please provide error documentation and contact your distributor.

FLM0414 **FLAMFILE SPLIT ACTIVE**

Explanation Splitting was activated. Creates or reads a number of fragments (files) of a FLAMFILE.

Action None.

FLM0415 **USED PARAMETER: ...**

Explanation Protocol of parameters used for compression.

Action None.

FLM0416 **COMPRESSION REDUCTION IN PERCENT: ...**

Explanation The input data was reduced around ... percent.

Action None.

FLM0421 **INPUT SUPPRESSED**

Explanation Input file was not processed.

Action None.

FLM0422 **INPUT DATA SET IS EMPTY**

Explanation The file to be compressed turned out to be empty.

Action None.

FLM0424 **ILLEGAL FUNCTION OR INSUFFICIENT MEMORY**

Explanation An invalid function was requested or the available memory is insufficient.
Possibly a licence error is detected, so all function calls are illegal.

Action Check memory space and increase the REGION entry if necessary.
Check your licence number (has your environment changed ?).

FLM0426 **MESSAGE NOT FOUND**

Explanation Error within the FLAM modules.

Action Please provide error documentation and contact your distributor.

FLM0428 **RECEIVED: ...**

Explanation Protocol of the entered compression parameters.

Action None.

FLM0429 **NAME GENERATION ERROR: NUMERIC RANGE OVERFLOW**

Explanation During split a new filename or DD-name has to be created but the numeric range overflows (e.g. JOE9: adding 1 to value 9 leads to 10, but there is only one byte to change, JOE09 is correct -> JOE10).

Action Use more numeric characters in the file (DD-) name.

FLM0431 **FLAMFILE SPLIT NO. nn MISSING**

Explanation On decompression a fragment of the splitted FLAMFILE is missing. The fragment has the number nn.

Action Check the filename, the catalog, is it free to read,... Correct the error and start again.

FLM0432 **FLAMFILE SPLIT SEQUENCE ERROR. FOUND NO. nn, NEED NO. mm**

Explanation On decompression a fragment of the serially splitted FLAMFILE is read. But the new file is number nn, expected number mm.

Action Check the ascending order of the files and start again.

FLM0433 **FLAMFILE SPLIT NO. nn IS NOT A CONTINUATION**

Explanation On decompression a fragment of a splitted FLAMFILE is read. But this new one is not an affiliation. It is an original fragment, but it belongs to a different FLAMFILE.

Action Check for the correct file and start again.

Note Please remember: each compression leads to an unique FLAMFILE. So it is not allowed to mix fragments of different runs.

FLM0435 **FLAMFILE MAC: nnnnnnnnnnnnnnn**
MEMBER MAC :

Explanation: Protocol of the calculated Hash-MACs of the entire FLAMFILE, or the member of the FLAMFILE.

Action: None.

Note: Using AES encryption, every FLAMFILE is secured with a couple of MACs. Every member of a GROUP-FLAMFILE is secured separately. These MACs are for safety and integrity of every level (matrix, member, file) in the FLAMFILE.

FLM0440 **FLAM COMPRESSION NORMAL END**

Explanation The FLAM compression has been completed successfully.

Action None.

FLM0441 **ERROR IN OPERATION: ...**

Explanation During this function an error occurred. The error code is documented in the following message.

Action None.

FLAMSYN	Syntax analysis for parameter input
FLAMREQM	Memory request
FLAMFREE	Memory release
FLAMSCAN	Analysis of a selection or conversion rule for file names
FLAMUP	Executive control
WCDxxx	Process file names in wildcard syntax
DYNxxx	Dynamic loading of modules and tables
TIOxxx	Terminal input/output
MSGxxx	Message output
TIMxxx	Timing
FIOxxx	File input/output
FLMxxx	FLAM record level interface

FLM0442 **DMS ERROR CODE: ... DD-NAME: ...**

Explanation During processing of the VSAM file or PO data set with the reported DD-name an error has occurred.

Action Analyze error code and correct file.

FLM0443 **FLAM ERROR CODE: ... DD-NAME: ...**

Explanation During processing of the file with the reported DD-name an error has occurred. Explanation of the error code:

30 =	Input file empty
31 =	Input file does not exist
32 =	Illegal open mode
33 =	Illegal file type
34 =	Illegal record format
35 =	Illegal record length
36 =	Illegal block length
37 =	Illegal key position
38 =	Illegal key length
39 =	Illegal file name
40 =	Module or table cannot be loaded
43 - 49 =	Abort caused by user exit
52 =	Too many or invalid keys
98 =	Not all files were processed

Action Analyze error code and correct file accordingly.

FLM0444 **COMPRESSION-LIMIT WARNING**

Explanation Compression ratio is worse than the specified limit (see CLIMIT, 3.11). Condition Code 80 is set.

Action None.

FLM0445 **. . . . . Message of the *KMEXIT module***

Explanation: Message data returned from the KMEXIT routine.

Action: None.

FLM0448 **COPYRIGHT (C) 1989-2005 BY LIMES DATENTECHNIK GMBH.**

Explanation Copyright message with customer licence number resp. expiration date of test installation.

Action None.

FLM0449 **FLAM COMPRESSION TERMINATED WITH ERRORS**

Explanation Compression has been completed with errors. Condition Code 8 or 12 or 16 is set.

Action None, resp. according to previous message.

FLM0450 **FLAM DECOMPRESSION VERSION ... ACTIVE**

Explanation The FLAM decompression system was activated. FLAM means: Frankenstein-Limes-Access-Method. FLAM is a registered trademark ™ copyright © by limes datentechnik® gmbh.

Action None.

FLM0456 **INPUT RECORDS/BYTES: ...**

Explanation Number of records and bytes in the compressed file (FLAMFILE).

Action None.

FLM0457 **OUTPUT RECORDS/BYTES: ...**

Explanation Number of records and bytes decompressed with FLAM.

Action None.

FLM0458 **CPU - TIME: ...**

Explanation CPU-time consumed by FLAM for decompression.

Action None.

FLM0459 **RUN - TIME: ...**

Explanation Elapsed time for decompression with FLAM. This includes the time needed for tape mounting.

Action None.

FLM0460 **DATA SET NAME: ...**

Explanation Name of the file to be decompressed (FLAMFILE) or of the target file (FLAMOUT).

Action None.

FLM0461 **DATA SET ORGANIZATION NOT SUPPORTED**

Explanation The target file cannot be created because FLAM does not support this file organization.

Action Assign an output file that FLAM can support.

FLM0462 **WRITTEN RECORDS/BYTES: ...**

Explanation Number of records and bytes written into the target file. A difference to the original file FLM0457 is caused by file conversion.

Action None.

FLM0463 **DECOMPRESSION ERROR CODE: ...**

Explanation Decompression was terminated with error code ... . (See also chapter 8.4)

10 = File none FLAMFILE

11 = FLAMFILE format error

12 = Record length error

13 = File length error

14 = Check sum error

20 = Invalid OPENMODE

21 = Invalid size of matrix buffer

22 = Invalid compression method

23 = Invalid code in FLAMFILE

24 = Invalid MAXRECORDS parameter

25 =	Invalid record length
26 =	Invalid character code
40 =	Module or table cannot be loaded
41 =	Module cannot be called
42 =	Module cannot be unloaded
43 - 49 =	Abortion caused by exit routine
52 =	Too many or invalid keys
57 =	Invalid partially compressed data length
60 - 78	FLAM syntax error
96 =	No file name found
98 =	Not all files were processed

Action In case of error code 10 - 14 the FLAMFILE has been modified. The error codes 40 - 49 are self explanatory.
In case of error code 60 - 78 please provide error documentation and contact your distributor.

FLM0465 **USED PARAMETER: ...**

Explanation Protocol of the decompression parameters used.

Action None.

FLM0468 **SPLIT RECORDS / BYTES: …**

Explanation Counter of records and Bytes of the actual fragment of the splitted FLAMFILE.
Cause of insertion of control and info bytes during split the counters differ from the numbers of compressed records/bytes (FLM0407, FLM0456).

Action None.

FLM0469 **COMPRESSED FILE FLAM-ID: ...**

Explanation FLAM system code of original file.
Some examples:

0080	MS-DOS
000E	Windows (all versions)

0101	IBM MVS
0102	IBM VSE/SP
0103	IBM VM
0104	IBM 81xx
0105	IBM DPPX/370
0106	IBM AIX
0107	IBM OS400
0109	Linux/S390
02xx	UNISYS
0301	DEC VMS
0302	DEC ULTRIX
0401	SIEMENS BS2000
0402	SIEMENS SINIX
0403	SIEMENS SYSTEM V
0501	NIXDORF 886x
0502	NIXDORF TARGON
06xx	WANG
07xx	PHILLIPS
08xx	OLIVETTI
09xx	TANDEM
0Axx	PRIME
0Bxx	STRATUS
11xx	INTEL 80286
12xx	INTEL 80386
13xx	INTEL 80486
15xx	Motorola 68000

xx04 UNIX

Action None.

FLM0470 **SPLIT ID: …**

Explanation To identify each fragment of a splitted FLAMFILE, a unique code is displayed.
The correspondig file name was displayed in FLM0410 or FLM0460.

Action None.

FLM0471 **OUTPUT SUPPRESSED**

Explanation Output file was not processed.

Action None.

FLM0472 **INPUT DATA SET IS EMPTY**

Explanation The file to be compressed (FLAMFILE) is empty.

Action Assign a FLAMFILE for decompression.

FLM0474 **ILLEGAL FUNCTION OR INSUFFICIENT MEMORY**

Explanation An illegal function was requested or the available memory space is insufficient.
Possibly a licence error is detected, so all function calls are illegal.

Action Check memory space and increase the REGION entry if necessary.
Check your licence number (has your environment changed ?).

FLM0475 **CRYPTOKEY WRONG OR MISSING**

Explanation The FLAMFILE has been encrypted and you entered a wrong key for decryption, or you forgot it at all.

Action Please enter the correct key for decryption (parameter CRYPTOKEY).

FLM0476 **NO. SPLITS EXCEEDS MAXIMUM OF nn**

Explanation A FLAMFILE has been splitted parallel in more fragments than the actual version is able to read. The actual version brings up to nn fragments together.

Action Please use the corresponding newer version of FLAM.

FLM0479 **DCB ATTRIBUTES CHANGED**

Explanation The file attributes for the target file differ from that of the original file. The file is converted into the new format.

Action None; or use other file format for target file.

FLM0480 **DCB PARAM OLD: ... NEW: ...**

Explanation Listing of the original file attributes and the file attributes used for decompression.

Action None, or define target file differently.

FLM0481 **RECORD TRUNCATED**

Explanation A record was truncated.The decompressed file contains one (ore more) records whose redord length is longer than the specified record length in the volume catalogue.
If TRUNCATE=NO is set the program is terminated with error.

Action To enforce conversion the program execution has to be repeated with parameter TRUNCATE=YES. Assign a file with a bigger record length for output.

FLM0482 **OLD ...**

Explanation Protocol of the FLAM file header.

OLD DSN **:** File name of original file

OLD CODE **:** Original file code

OLD DSORG **:** Original file organization

OLD RECFORM **:** Original file format

OLD RECSIZE **:** Original file record length

OLD BLKSIZE **:** Original file block size

OLD KEYPOS : Original file key position

OLD KEYLEN : Original file key length

Action None.

FLM0483 **ACTUAL FLAMFILE VERSION NOT SUPPORTED: nn**

Explanation: The actual FLAM version is unable to decompress/decrypt the FLAMFILE. On compression, new parameters or functions have been used that are not compatible to this actual version. nn identifies the FLAMFILE version.

Action: Please use the newest FLAM version.

FLM0485 **FLAMFILE MAC: nnnnnnnnnnnnnnn**
MEMBER MAC :

Explanation: Protocol of the calculated Hash-MACs of the entire FLAMFILE, or the member of the FLAMFILE.

Action: None.

Note: Using AES encryption, every FLAMFILE is secured with a couple of MACs. Every member of a GROUP-FLAMFILE is secured separately. These MACs are for safety and integrity of every level (matrix, member, file) in the FLAMFILE.

FLM0487 **USER HEADER: …**

Explanation Protocol of the user header, if any. The message ends with three points '…', if the line is too short for the entire data.

Action None..

FLM0488 **INPUT WAS NOT COMPRESSED BY FLAM**

Explanation The input data was not compressed with FLAM. Condition Code 88 is set.

Action Assign a compressed file that was compressed with FLAM.

FLM0490 **FLAM DECOMPRESSION NORMAL END**

Explanation The decompression with FLAM was completed successfully.

Action None.

FLM0491 **ERROR IN OPERATION: ...**

Explanation During this function an error occurred. The error code is documented in the following message.

Action None.

FLM0492 **DMS ERROR CODE: ... DD-NAME: ...**

Explanation During processing a VSAM file or PO data set with the specified DD-name an error has occurred.

Action Analyze error code and correct file.

FLM0493 **FLAM ERROR CODE: ... DD-NAME: ...**

Explanation During processing the file with the specified link name an error has occurred. Explanation of error code:

30 = Input file empty

31 = Input file does not exist

32 = Illegal open mode

33 = Illegal file type

34 = Illegal record format

35 = Illegal record length

36 = Illegal block length

37 = Illegal key position

38 = Illegal key length

39 = Illegal file name

Action Analyze error code and correct file.

FLM0499 **FLAM DECOMPRESSION TERMINATED WITH ERRORS**

Explanation Decompression with FLAM was terminated with error. Condition Code 8 or 12 or 16 is set.

Action Analyze error.

8.3 FLAM return codes

FLAM reports certain exceptional situations and errors via system-neutral return codes at the various interfaces (FLAMUP, FLAMREC and USERIO).

The values below are decimal numbers.

Where error codes relate to **files**, the file is marked in the most significant byte:

X'AF'	error on accessing	FLAMOUT
X'CF'		FLAMPAR
X'EF'		FLAMIN
X'FF'		FLAMFILE

FLAM uses this identifier to select a suitable message.

The last three bytes are the error code of the special data management routine (e.g. VSAM, PO-Data Sets).

Security violations are marked in the 2. byte:

00kkmmmm.

kk identifies where the error was detected, kk =

01	header
02	segment
03	membertrailer
04	filetrailer

mmmm describes the error (hexadecimal):

0001	MAC1, after encryption
0002	MAC2, sequence MAC
0004	MAC3, Mac on macs
0010	missing data
0020	data inserted
0040	data updated
0080	record counter compression
0100	byte counter compression
0200	record counter original data
0400	byte counter original data
0800	chaining on FLAM decryption

Multiple errors are or'ed (e.g. 0180, record and byte counter both in error).

Security violations are detected during decompression. If the error situation is well known and acceptable, use parameter SECUREINFO=IGNORE to ignore the error.

Positioning into a FLAMFILE and decompressing a member of a group FLAMFILE implies the usage of SECUREINFO=MEMBER (else an error code X'00030002 is returned, i.e. a member-MAC sequence error).

Return code

0 The function has been completely executed.

-1 The function has not been executed because it is illegal in this context (e.g. FLMGET without successful FLMOPN, FLAM has not been licensed) or because there is insufficient memory available when a file is opened.

Return codes between 1 and 9 are warnings.

The function has been partially executed. The user must decide whether the result is right or wrong.

1 A record has been shortened to the length of the record buffer; the data can be processed in the length specified.

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. The sequential read position is located on the record with the next-highest key.

When positioning, the position specified does not exist or the positioning desired is not possible. The current position is retained.

When deleting, there is no current record.

6 When reading in a group file, a new file is starting; no data is transferred. The file header can be read if necessary. The sequential read position is located on the first record of the new file.

7 Password / cryptokey missing on decompression. FLAMFILE was created using a password / cryptokey. Pass it via FLMPWD.

8 not used

9 When compressing with the statistics switched on, FLAMUP or FLAM reports that the compressed file is larger than the original file (expansion).

Return codes 10 and higher are errors.

The function has not been executed or has been aborted. (Exception: return code 98 from FLAMUP or FLAM)

10 During decompression, the input file has not been recognized as being a FLAM compressed file. The very beginning of the file is corrupted to such an extent that the FLAM syntax cannot be recognized.

Possible causes of this error are:

- The input file is not a compressed file or it was not compressed using FLAM.
- The very first record has been shortened or data has been inserted in front of the FLAM compressed file.

This error is often caused by incorrectly set file transfers:

When 8-bit compressed files are transferred, a file transfer for printable data is used and the characters of the compressed file are corrupted as a result.

When index sequential compressed files are transferred from DEC-VMS to a different system (such MVS, BS2000, etc.), the key length of the compressed file must be increased by the record and block counters (1, 2 or 4 bytes).

Compressed records are shortened, lengthened or wrapped while being transferred.

Note: Some of these transformations are now recognized and automatically corrected by FLAM.

Padding with identical characters is tolerated for all compression methods.

With 8-bit compressed files, it is possible to wrap the compressed records as long as no exit is active for the compressed records (EXD20) during decompression.

11 The format of the FLAMFILE is wrong.

Errors have been detected in the syntax of the compressed data while decompressing a FLAMFILE. For example, entire compressed records are missing or headers are corrupted.

12 A compressed record has been shortened so that part of the compressed data is missing.

13 The compressed file has been shortened. Entire compressed records are missing at the end of the file. This error can arise while creating, copying or transferring compressed files, if there is not enough disk space available for the compressed file and therefore to the compression, copying or file transfer is terminated too early. Any other abort of these processes can also result in an incomplete compressed file.

14 The checksum of a compressed record is wrong. The compressed file has been corrupted by recoding or some other form of intervention.

15 FLAM can only process records with a maximum length of 32,764 bytes. The original file contains at least one record that is longer and can therefore not be compressed.

16 The matrix size must be at least 4 bytes longer than the longest record length in the original file. The matrix size should be at least 16 times the record length in order to achieve good compression results. The file can be compressed again using a larger matrix buffer.

17 not used

18 not used

19 not used

20 Invalid OPENMODE.

Only index sequential compressed files can be opened with OPENMODE=INOUT. Sequential compressed files can only be read (INPUT) or written (OUTPUT).

21 Invalid size of matrix buffer.

During decompression, the matrix buffer required cannot be requested due to a lack of memory. If it is not possible to make any more memory available, the original file must be compressed using a smaller matrix buffer.

Note: As of version 2.5, a matrix buffer of twice the size is required. If necessary, the compressed file can be decompressed using version 2.1, so that it can then be compressed again using a smaller matrix buffer.

22 Invalid compression method.

The compressed file has been created with a more recent version of FLAM using a compression method that is not yet supported by this version.

23 Invalid code in FLAMFILE.

The compressed file has been created in a character code (neither ASCII nor EBCDIC) that is not yet supported by this version of FLAM.

24 Invalid maximum number of records.

The MAXRECORDS or MAXREC parameter contains a value greater than 255 or less than 1.

25 Invalid record length.

The MAXSIZE parameter contains a value less than 80 or greater than 32,768 for 8-bit compressed data. With CX7, MAXSIZE must not be greater than 4096.

26 Invalid character code.

The original data uses a character code (neither ASCII nor EBCDIC) that is not yet supported by this version of FLAM.

27 not used

28 not used

29 Password missing or invalid (passed by FLMPWD).

30 Input file is empty. The input file exists but does not contain any data.

31 Input file does not exist.

32 Invalid OPENMODE.

The file cannot be opened with the OPEN mode selected. For example, a sequential file cannot be opened for update.

33 Invalid file type.

The file format desired can not or can not yet be processed by FLAM.

34 Invalid record format.

The record format cannot be processed by FLAM or it is not valid for the file format specified.

35 Invalid record length.

The record length cannot be processed by FLAM or it is not valid for the file format and record format specified.

36 Invalid block length.

The block length cannot be processed by FLAM or it is not valid for the file format and record format specified.

37 Invalid key position.

The key position in an index sequential FLAMFILE is not 1. For an original file, the key position is not valid for the file format specified.

38 Invalid key length.

The key length cannot be processed by FLAM or it is not valid for the file format and record format specified.

39 Invalid file name.

The file name has been specified in an invalid notation for a file or a library element, or an invalid wildcard specification has been used for a set of files and library elements

or this wildcard specification cannot be processed by FLAM.

40 Module or table cannot be loaded.

A user exit or a conversion table cannot be loaded. It may be that the library is not assigned.

41 Module cannot be called.

A user exit cannot be called.

42 Module or table cannot be loaded.

43 - 49 Abort caused by exit routine.

A user exit has returned the return code 16 or an invalid return code.

50-51 not used

52 Too many or invalid duplicate keys.

During compression into an index sequential FLAMFILE, the original file contains duplicate keys, even though duplicate keys are not allowed in the KEYFLAGS field of the KEYDESC (key description) when opening the FLAMFILE. Or the number of duplicate keys in the original is greater than 255 * MAXREC.

53-56 not used

57 Invalid partially compressed data length.

The compressed data of a matrix has been stored in several parts with their own length fields. During decompression, an inconsistency of these length fields is detected, without an invalid checksum having been found.

This error arises if entire records have been deleted from a compressed file.

58 not used

59 not used

60 - 78 Errors 60 to 78 describe all of the errors possible in the compressed data.

These errors identify program errors in FLAM itself and therefore must not arise during operation.

Since the ability to detect a corruption in a compressed file via checksums is limited to a certain level of probability, it is possible that in few cases a decompression error is reported inappropriately, even though there is a corruption.

If a decompression error arises, it should be reported to the manufacturer (enclose error documentation).

79 not used

80 Syntax error during parameter input.

The syntax of the parameter string is incorrect. If a number of parameters have been transferred at a time, the error can be localized by shortening the parameter string by one parameter each time.

81 Unknown key word.

The parameter string contains an unknown key word or a parameter value is interpreted as a key word due to a syntax error.

82 Unknown parameter value.

An invalid value has been specified for a parameter with a fixed range of allowed values, such as MODE.

83 Parameter value not decimal.

A non-numeric value has been specified for a parameter whose range of allowed values consists of numbers only.

84 Parameter value too long.

The value specified for a parameter is too long. Numeric values can contain a maximum of 8 digits and fixed values may only comprise a maximum of 8 characters, too. The lengths of parameters which can contain names are specified in the respective parameter description. Link names, module names and the names of tables are also only allowed to be a maximum of 8 characters long. File names for individual files and those which contain wildcards can be a maximum of 54 characters long.

85-89 unused

90 Parameter is not allowed at this moment (e.g. CRYPTO-MODE after function FLMOPF).

91 Unknown parameter

92 Unknown parameter value

93-95 unused

96 No file name found or error when determining file name. This error can arise during compression in connection with file name specifications in wildcard syntax or file lists.

During decompression, this error is due to a selection or conversion rule being specified for the output and the FLAMFILE not containing a name for the original file.

97 not used

98 Not all files were processed.

While processing group files not all of the files have been processed, because errors have been detected when opening the original files. All of the files that have been processed, have been processed without error.

111 Called for serial split, but 0 split size.

112 Called for parallel split, but split number < 2.

119 Length error in a split FLAMFILE. Please send any error protocols.

120 The filename is in error. The number-characters are too short to generate a new number. E.g. after 9, the number 10 should be created but the number-character has one byte only.

121 On decompression, one fragment of a split FLAMFILE is missing.

122 On decompression, a fragment of a serial split FLAMFILE is not in ascending order.

123 A fragment of a split FLAMFILE does not belong to the actual split.

124 A FLAMFILE has been split parallel into more fragments than the actual version is able to read.

125 A formal error in the last record of the actual fragment of the split FLAMFILE.

126-129 unused

130 Security violation. The FLAMFILE has been changed (e.g. update, concatenation).

If acceptable, use SECUREINFO=IGNORE on decompression.

131 Security violation. Missing records or complete member in a group FLAMFILE.

If acceptable, use SECUREINFO=IGNORE on decompression.

132 Security violation. A new member has been inserted in a group FLAMFILE.

If acceptable, use SECUREINFO=IGNORE on decompression.

133 Security violation. The record sequence has been changed.

If acceptable, use SECUREINFO=IGNORE on decompression.

134 Unexpected security information in a standard FLAMFILE.

	You cannot ignore this error situation. Perhaps you have concatenated a ‘normal’ and a ‘secure’ FLAMFILE?
135-998	unused
999	as –1
> 65535	marked errors (see the beginning of the chapter)

8.4 Condition codes

FLAM sets the following condition codes for executive control:

Condition codes

0	Error-free execution
4	Not all input/output files have been processed during the processing of group files
8	Simple errors (such as parameter errors) have been detected
12	Usually, DMS errors are present
16	Serious error during compression/decompression
80	The compression ratio was worse than the specified limit (see CLIMIT parameter)
88	The file assigned is not a FLAMFILE

Processing has been executed correctly only when condition code 0 or 80 is set. In all other cases, it may be that either a corrupt compressed file has been created or no compressed file at all. We recommend that this file is re-catalogued, so that it is not used for further processing.

If a condition code greater than 0 is returned, FLAM has already displayed an appropriate error message.

If errors with condition code 16 are reported, there may be an error in FLAM.

FLAM (MVS)

User Manual

Chapter 9:

The FLAM user interface

Content

9. The FLAM user interface

9.1 Summary

Everyday usage of FLAM is greatly simplified under TSO/ISPF by calling procedures and panels.

CLIST procedures enable FLAM to be called directly when a file list is available (panel 3.4 in ISPF). Thus, directory contents of a FLAMFILE can be displayed, or files can be compressed, decompressed, viewed or edited.

FLAM can also be incorporated as a menu item in a selection panel (e.g. ISRUTIL).

The panels, CLIST procedures and messages belonging to the FLAM user interface are delivered in PO libraries in legible form. This allows the user to modify them and adapt them to their specific requirements.

Please keep in mind that maintenance and warranty only applies to the delivered version. Note also that modifications and adaptations made by the customer must be applied again by the customer himself in case of version changes.

9.2 FLAM panels

The term FLAM panels refers to the user interface that can be linked into a selection panel.

The task of these panels is the compression and decompression of files for execution in dialog (TSO) or in batch.

No knowledge of JCL is necessary. All the necessary commands are generated automatically.

All panels provide help information when pressing the PF1 key. You can find general information as well as detailed explanations following error messages. All messages and help texts are written in English.

For execution in batch, control is transferred to JES. In consequence the function 3.8 can be used from ISPF or, e.g. SDSF, to control the batch job.

After execution under TSO, control is passed automatically to the result list of FLAM.

Start of FLAM panels

To start the FLAM user interface please enter under ISPF (e.g. in the command line):

```
------------------------    ISPF/PDF PRIMARY OPTION MENU  -----------------
 OPTION    ===> tso exec pref(flam)

      0  ISPF PARMS       - Specify terminal and user parameters
      1  BROWSE           - Display source data or output listings
      2  EDIT             - Create or change source data
      3  UTILITIES        - Perform utility functions
      .
      .
      .
```

With 'pref' as file name prefix of the FLAM CLIST library (according TSO conventions).

Then the FLAM user interface panel will be displayed.

We recommend to modify an existing ISPF panel (e.g. ISRUTIL) instead. In the following example the changes are printed in bold characters:

```
%------------------------    UTILITY SELECTION MENU  ------------------
%OPTION    ===_ZCMD
%
%      1 +LIBRARY    - Compress or print data set.  Print index listing.
+                        Print, rename, delete, or browse members
%      2 +DATASET    - Allocate, rename, delete, catalog, uncatalog, or
+                        display information of an entire data set
%      3 +MOVE/COPY  - Move, copy, or promote members or data sets
%      4 +DSLIST     - Print or display (to process) list of data set
+                        Print or display VTOC information
%      5 +RESET      - Reset statistics for members of ISPF library
%      6 +HARDCOPY   - Initiate hardcopy output
%      8 +OUTLIST    - Display, delete, or print held job output
%      9 +COMMANDS   - Create/change an application command table
%     10 +CONVERT    - Convert old format menus/messages to new format
%     11 +FORMAT     - Format definition for formatted data Edit/Browse
%     12 +SUPERC     - Compare data sets (Standard dialog)
%     13 +SUPERCE    - Compare data sets (Extended dialog)
%     14 +SEARCH-FOR - Search data sets for strings of data
%     15 +FLAM       - Data Compression Utility
)INIT
    .HELP = ISR30000
)PROC
    &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                  1,'PGM(ISRUDA) PARM(ISRUDA1)'
                  2,'PGM(ISRUDA) PARM(ISRUDA2)'
                  3,'PGM(ISRUMC)'
                  4,'PGM(ISRUDL) PARM(ISRUDLP)'
                  5,'PGM(ISRURS)'
                  6,'PGM(ISRUHC)'
                  8,'PGM(ISRUOLP)'
                  9,'PANEL(ISPUCMA)'
                 10,'PGM(ISRQCM) PARM(ISRQCMP)'
                 11,'PGM(ISRFMT)'
                 12,'PGM(ISRSSM)'
                 13,'PGM(ISRSEPRM) NOCHECK'
                 14,'PGM(ISRSFM)'
                 15,'CMD(EXEC pref.CLIST(FLAM))'
                ' ',' '
                  *,'?' )
    &ZTRAIL = .TRAIL
)END
```

If this modified panel is concatenated by using a transparent file (DD name ISPPLIB), the FLAM start menu can be called with function 3.15 !

FLAM start panel

After the TSO EXEC command or 3.15 was entered in the command line the FLAM start panel will be displayed:

```
--------------------------------      F L A M    -------------------------
OPTION    ===>

     C - Compress data set or member       I -  FLAMFILE-info
     D - Decompress data set or member     O -  Processing options

   Specify original data set or member (blank for DUMMY):
     DATA SET NAME ===>

   Specify FLAMFILE data set or member (blank for DUMMY):
     DATA SET NAME ===>

   Reuse existing data sets:  N      (Y/N  yes/no)

   Specify Listing (* for temporary, blank for none)
     DATA SET NAME === *

   FLAM Parameter:      ===>
     ===>

   Submit:  F      (F/B  Foreground or Batch)
-------------------------------------------------------------------------
```

By specifying an option the function is chosen:

- compression (C),

- decompression (D),

- display of informations from the FLAMFILE (I),

- jump to another panel for JCL generation (O).

File names can be entered according to TSO conventions. If a file name is not enclosed in quotes, a file from the own identification is assumed. Files from foreign identifications (Userid) must be specified with the full file name enclosed in quotes. For execution in batch the file name is prefixed with the identification if the quotes are missing.

E.g., while running under identification FLAM, a data set name of FLAM.DAT.SMF is generated from the input DAT.SMF for the execution in batch. But if 'SYS2.DAT.SMF' is entered, the generated data set name will be SYS2.DAT.SMF.

If no file name is entered, a DUMMY statement will be generated. This means that no file is read or created by FLAM. This can be useful to evaluate compression ratios, test and debug jobs, or to use user exits with own I/O routines.

The file name is checked for syntax. If the file is not catalogued, the allocation panel will be called to allow to catalogue and allocate the file; an error message is given otherwise.

The FLAM protocol is principally stored into a file if no other instructions are given by generation or via parameter (see parameter MSGDISP).

Also for this file a file name can be specified. If '*' (default) is specified, a temporary file will be allocated and released after execution. For batch the statement SYSOUT=* will be generated - so the protocol will appear in the JCL result list.

With the specification of 'Reuse existing Data Set : N '

an overwrite of an existing file can be inhibited. An error message is given instead. This applies also for the protocol file !

Up to 100 characters can be entered for FLAM parameters in total. For execution only one string is generated. So a comma must be given if the parameters are continued on the second input line. The parameters are not checked by the panel. They are simply passed to FLAM and checked by FLAM during execution.

Execution may take place under TSO (foreground) or as a batch process. The necessary JCL is generated automatically.

After execution has finished control is passed automatically to the FLAM protocol file under TSO. All commands of the ISPF browse mode are allowed (like positioning, etc.) Function key PF3 will terminate the display and will give control back to the FLAM start menu.

FLAM options:

When option ' **o** ' is entered default values for the JCL generation shall be specified. Control is given to the next panel:

```
------------------------------    FLAM - Options  -----------------------
FLAM Load Library (Data Set Name)
    === FLAMV30A.LOAD
New Data Set Defaults     FLAMFILE               Original Data Set

     Record Format   ===> FB           (F,FB,V,VB)
     Record Length   ===> 2048         (80 to 32760 Byte)
     Block Size      ===> 26624        (80 to 32760 Byte)
     Space Unit      ===> TRKS         (BLKS, TRKS, or CYLS) ===> TRKS
     Primary Quant.  ===> 10             (in above units)    ===> 20
     Secondary Qu.   ===> 5              (in above units)    ===> 4
     Volume          ===>                                    ===>
     Unit            ===> SYSDA                              ===> SYSDA

JOB Statement Information      (required for batch-processing only)
   ===> //FLAM30A JOB 7021000F,'LIMES-496172/59190',
   ===> //        CLASS=A,MSGLEVEL=2,MSGCLASS=X,
   ===> //        NOTIFY=FLAM30

   Press  ENTER for return,    PF3 or PF4 will cancel.
```

For demonstration purposes the panel is already supplied with values. The values will be used for each FLAM execution. So the user doesn't have to fill in the same values again and again.

The specification of a FLAM load module library is mandatory. Without such a library no execution is possible. Usually the library name has been determined during installation, but it is always possible to change this name to support version changes or test environments.

By default the FLAMFILE is created as a sequential PS file. Here it is possible to define a certain file format, file size and storage medium as default values.

Because the output file during decompression can be of any kind, only default values for size and storage medium are required.

All these values can be changed during execution. Here only the default values are defined.

The JOB card is only used for execution in batch. If no default JOB card is provided, each batch JCL generation will ask for the job card specification.

After pressing the ENTER key, control will be given back to the FLAM start menu. All inputs are stored into the ISPF PROFILE file and will be available for future calls of FLAM panels.

The function keys PF3 and PF4 will cancel the process - the inputs are discarded and not stored.

9.2.1 Example for compression:

For compression '**c**' is specified as option:

```
--------------------------------      F L A M    -------------------------
OPTION    ===> c

     C - Compress data set or member          I - FLAMFILE-info
     D - Decompress data set or member        O - Processing options

   Specify original data set or member (blank for DUMMY):
     DATA SET NAME ===> dat.fb

   Specify FLAMFILE data set or member (blank for DUMMY):
     DATA SET NAME ===> dat.cmp

   Reuse existing data sets:  N      (Y/N  yes/no)

   Specify Listing (* for temporary, blank for none)
     DATA SET NAME ===> *

   FLAM Parameter:
     ===> mode=vr8
     ===>

   Submit:  F      (F/B  Foreground or Batch)
```

The file DAT.FB of the users identification shall be compressed into file DAT.CMP of the users identification. Since this file shall not exist already, overwrite is inhibited (as a protection for mistakes during entry). The compression shall use compression mode VR8.

After the ENTER key is pressed, the file name is checked and the existence of the file is tested.

Because the compression file DAT.CMP is a new file and not yet allocated, control is passed to the FLAM allocation panel:

```
---------------------    F L A M   Data Set Specification  --------------

 Data Set  DAT.CMP

        Special attributes or allocation for new  FLAMFILE
        For standard processing press <ENTER> without any input.

 Organization      ===> PS             (PS/PO/ESDS/KSDS/RRDS/LDS)
 Record Format     ===> FB             (F/FB/V/VB, with S,A,M, or /U)
 Record Length     ===> 2048           (up to 32760 Byte, avg. max for VSAM
 Block Size        ===> 26624          (up to 32760 Byte, CISZ for VSAM
 Key Position      ===>                            ( VSAM KSDS
 Key Length        ===>                (up to 255) (        only
 No.Dir.Blocks     ===>                (PO Data Set only)
 Space Unit        ===> TRKS           (BLKS, TRKS, CYLS, or RECS)
 Primary Quantity  ===> 10             (in above units)
 Secondary Quant.  ===> 5              (in above units)
 Volume Serial     ===>
 Generic Unit      ===> SYSDA
```

This panel contains already the default values as specified in the FLAM option panel. They may be modified if required. E.g., a VSAM file could be created instead.

In this example we want to create a compressed file for transmission to a PC with IND$FILE. We change the record and block sizes to appropriate values:

```
---------------------    F L A M   Data Set Specification  -------------

 Data Set  DAT.CMP

        Special attributes or allocation for new  FLAMFILE
        For standard processing press <ENTER> without any input.

 Organization      ===> PS           (PS/PO/ESDS/KSDS/RRDS/LDS)
 Record Format     ===> FB           (F/FB/V/VB, with S,A,M, or /U)
 Record Length     ===> 128          (up to 32760 Byte, avg. max for VSAM
 Block Size        ===> 1280         (up to 32760 Byte, CISZ for VSAM)
 Key Position      ===>                         ( VSAM KSDS
 Key Length        ===>              (up to 255) (        ONLY
 No.Dir.Blocks     ===>              (PO Data Set only)
 Space Unit        ===> TRKS         (BLKS, TRKS, CYLS, or RECS)
 Primary Quantity  ===> 1            (in above units)
 Secondary Quant.  ===> 1            (in above units)
 Volume Serial     ===>
 Generic Unit      ===> SYSDA
```

After ENTER is pressed, processing starts.

Depending on file size and CPU workload the FLAM processing can take a while. Therefore the following panel is displayed.

```
------------------------------      F L A M    --------------------------
OPTION    ===> C

     C  -  Compress data set or member         I  -  FLAMFILE-info
     D  -  Decompress data set or member       O  -  Processing options

   Specify original data set or member (blank for DUMMY):
     DATA SET NAME ===> DAT.FB

   Specify FLAMFILE data set or member (blank for DUMMY):
     DATA SET NAME ===> DAT.CMP

   Reuse existing data sets:  N      (Y/N  yes/no)

   Specify Listing (* for temporary, blank for none)
     DATA SET NAME ===> *

                             F L A M    is working now

   Submit:  F      (F/B  Foreground or Batch)
```

After compression has finished control is automatically passed to the result display. All commands allowed in the BROWSE mode can be used, like positioning, searching, etc.

```
 BROWSE - FLAM30.FLAM.TEMPLIST -------------------- LINE 00000000 COL
 COMMAND ===>
******************************** TOP OF DATA ***********************
FLM0448 COPYRIGHT (C) 1989-1999 BY LIMES DATENTECHNIK GMBH TEST 2000182
FLM0428 RECEIVED: C,MODE=VR8
FLM0400 FLAM COMPRESSION VERSION 3.0A00 ACTIVE
FLM0410 DATA SET NAME : FLAM30.DAT.FB
FLM0415 USED PARAMETER: ACCESS    : LOG
FLM0415 USED PARAMETER: IDSORG    : SEQUENT
FLM0415 USED PARAMETER: IRECFORM  : FIXBLK
FLM0415 USED PARAMETER: IRECSIZE  :       80
FLM0415 USED PARAMETER: IBLKSIZE  :     3120
FLM0410 DATA SET NAME : FLAM30.DAT.CMP
FLM0415 USED PARAMETER: MODE      : VR8
FLM0415 USED PARAMETER: MAXBUFF   :    32768
FLM0415 USED PARAMETER: MAXREC    :      255
FLM0415 USED PARAMETER: MAXSIZE   :      128
FLM0415 USED PARAMETER: DSORG     : SEQUENT
FLM0415 USED PARAMETER: RECFORM   : FIXBLK
FLM0415 USED PARAMETER: BLKSIZE   :     1280
FLM0408 CPU - TIME:        0.0390
FLM0409 RUN - TIME:        0.3325
FLM0406 INPUT   RECORDS/BYTES:         27 /       2,160
FLM0407 OUTPUT  RECORDS/BYTES:          9 /       1,152
FLM0416 COMPRESSION REDUCTION IN PERCENT:     46.67
FLM0440 FLAM COMPRESSION NORMAL END
******************************** BOTTOM OF DATA ****************
```

When PF3 is pressed control is given back to the start menu.

9.2.2 Example for decompression

Here no output file is specified, the according input field remains empty. This will cause a complete decompression, but without the creation of an output file.

```
--------------------------------      F L A M    ------------------------
 OPTION    ===> d

      C - Compress data set or member          I - FLAMFILE-info
      D - Decompress data set or member        O - Processing options

    Specify original data set or member (blank for DUMMY):
      DATA SET NAME ===>

    Specify FLAMFILE data set or member (blank for DUMMY):
      DATA SET NAME ===> DAT.CMP

    Reuse existing data sets:  N      (Y/N  yes/no)

    Specify Listing (* for temporary, blank for none)
      DATA SET NAME ===> *

    FLAM Parameter:
      ===>
      ===>
    Submit:  F      (F/B  Foreground or Batch)
```

The result:

```
 BROWSE - FLAM30.FLAM.TEMPLIST -------------------- LINE 00000000 COL
 COMMAND ===>                                                  SCROLL
******************************** TOP OF DATA ************************
FLM0448 COPYRIGHT (C) 1989-1999 BY LIMES DATENTECHNIK GMBH TEST 2000182
FLM0428 RECEIVED: D,
FLM0450 FLAM DECOMPRESSION VERSION 3.0A00 ACTIVE
FLM0460 DATA SET NAME : FLAM30.DAT.CMP
FLM0465 USED PARAMETER: MODE     : VR8
FLM0465 USED PARAMETER: VERSION  :      200
FLM0465 USED PARAMETER: MAXBUFF  :    32768
FLM0465 USED PARAMETER: CODE     : EBCDIC
FLM0465 USED PARAMETER: DSORG    : SEQUENT
FLM0465 USED PARAMETER: RECFORM  : FIXBLK
FLM0465 USED PARAMETER: RECSIZE  :      128
FLM0465 USED PARAMETER: BLKSIZE  :     1280
FLM0482 OLD ODSN     : FLAM30.DAT.FB
FLM0482 OLD ODSORG   : SEQUENT
FLM0482 OLD ORECFORM : FIXBLK
FLM0482 OLD ORECSIZE :         80
FLM0482 OLD OBLKSIZE :       3120
FLM0469 COMPRESSED FILE FLAM-ID: 0101
FLM0460 DATA SET NAME : NULLFILE
FLM0465 USED PARAMETER: ACCESS    : LOG
FLM0458 CPU - TIME:        0.0254
FLM0459 RUN - TIME:        0.0778
FLM0456 INPUT   RECORDS/BYTES:          9 /       1,152
FLM0457 OUTPUT  RECORDS/BYTES:         27 /       2,160
FLM0490 FLAM DECOMPRESSION NORMAL END
******************************** BOTTOM OF DATA *****************
```

9.2.3 Information about a FLAMFILE

The information of the compression file can be shown in 2 different ways:

The instruction SHOW=DIR for decompression

The protocol is to be stored into EXAMPLE.LIST:

```
--------------------------------      F L A M    -------------------------
OPTION    ===> d

     C - Compress data set or member        I - FLAMFILE-info
     D - Decompress data set or member      O - Processing options

   Specify original data set or member (blank for DUMMY):
     DATA SET NAME ===

   Specify FLAMFILE data set or member (blank for DUMMY):
     DATA SET NAME ===> DAT.CMP

   Reuse existing data sets:  N      (Y/N  yes/no)

   Specify Listing (* for temporary, blank for none)

     DATA SET NAME ===> example.list

   FLAM Parameter:
     ===> show=dir
     ===>
   Submit:  F      (F/B  Foreground or Batch)
 ------------------------------------------------------------------------
```

The protocol file is generated by default (in TSO on the disc assigned by the system administrator, in batch on the specified unit of the FLAMFILE in the FLAM option menu or on SYSDA.

And the result:

```
BROWSE - FLAM30.EXAMPLE.LIST ---------------------- LINE 00000000
COMMAND ===>
******************************** TOP OF DATA ***********************
FLM0448 COPYRIGHT (C) 1989-1999 BY LIMES DATENTECHNIK GMBH TEST 2000182
FLM0428 RECEIVED: D,SHOW=DIR
FLM0450 FLAMD DECOMPRESSION VERSION 3.0A00 ACTIVE
FLM0460 DATA SET NAME : FLAM30.DAT.CMP
FLM0465 USED PARAMETER: MODE    : VR8
FLM0465 USED PARAMETER: VERSION :      200
FLM0465 USED PARAMETER: MAXBUFF :    32768
FLM0465 USED PARAMETER: CODE    : EBCDIC
FLM0465 USED PARAMETER: DSORG   : SEQUENT
FLM0465 USED PARAMETER: RECFORM : FIXBLK
FLM0465 USED PARAMETER: RECSIZE :      128
FLM0465 USED PARAMETER: BLKSIZE :     1280
FLM0482 OLD ODSN    : FLAM30.DAT.FB
FLM0482 OLD ODSORG  : SEQUENT
FLM0482 OLD ORECFORM: FIXBLK
FLM0482 OLD ORECSIZE:       80
FLM0482 OLD OBLKSIZE:     3120
FLM0469 COMPRESSED FILE FLAM-ID: 0101
FLM0458 CPU - TIME:        0.0174
FLM0459 RUN - TIME:        0.0375
FLM0456 INPUT   RECORDS/BYTES:          1 /        128
FLM0457 OUTPUT  RECORDS/BYTES:          0 /          0
FLM0490 FLAM DECOMPRESSION NORMAL END
```

A decompression was not performed (output records = 0). Only FLAMFILE header information is displayed.

By input of option 'i'

```
--------------------------------      F L A M    -------------------------
 OPTION    ===> i

      C - Compress data set or member        I - FLAMFILE-info
      D - Decompress data set or member      O - Processing options

    Specify original data set or member (blank for DUMMY):
      DATA SET NAME ===>

    Specify FLAMFILE data set or member (blank for DUMMY):
      DATA SET NAME ===> DATASET.CMP

    Reuse existing data sets:  N      (Y/N  yes/no)

    Specify Listing (* for temporary, blank for none)

      DATA SET NAME ===>

    FLAM Parameter:
      ===>
      ===>
    Submit:  F      (F/B  Foreground or Batch)
  ------------------------------------------------------------------------
```

the content of FLAMFILE DATASET.CMP analogously ISPF 3.4 will be displayed:

```
FLAMFILE TOC DATASET.CMP                                        Row 1 of 2170
                 MODE VR8    MAXBUFFER 64      FLAMCODE EBCD
   Original Data Set Name                        Dsorg Recfm Lrecl Blksi Space
------------------------------------------------------------------------------
   FLAMT.AD0001NP.LIST                           SEQ   FBM   133   3059    300 K
   FLAMT.AD0001NP.CX8                            SEQ   FB    80    23440    50 K
   FLAMT.AD0191NP.LIST                           SEQ   FBM   133   3059    500 K
   FLAMT.AD0192NP.LIST                           SEQ   FBM   133   3059    250 K
   FLAMT.EXD4TO3.LIST                            SEQ   FBM   133   3059    150 K
   FLAMT.EXK1NUL.LIST                            SEQ   FBM   133   3059     50 K
   FLAMT.EXK3TO4.LIST                            SEQ   F     133   133     350 K
   FLAMT.FLAM.CMP                                SEQ   FB    512   23552 12800 K
   FLAMT.FLAMDIR.LIST                            SEQ   FBM   133   3059    200 K
   FLAMT.FLAMFLN.LIST                            SEQ   F     133   133    3150 K
   FLAMT.FLAMG001.LIST                           SEQ   FBM   133   3059   1250 K
   FLAMT.FLAMG002.LIST                           SEQ   F     133   133     500 K
   FLAMT.FLAMHELP.LIST                           SEQ   F     133   133     550 K
   FLAMT.FLAMNUC.LIST                            SEQ   F     133   133   11300 K
   FLAMT.FLAMTADC.LIST                           SEQ   FBM   133   3059    100 K
   FLAMT.FLAMTS.LIST                             SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                           SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                           SEQ   VB    260   23440 28500 K

COMMAND ===>
```

9.3 FLCOMP

This CLIST procedure is intended for use in panel 3.4 in ISPF (file list), but it can also be called directly. In the latter case, the file name is requested.

To compress the file specified in the line, FLCOMP branches to the FLAM panel routine. In this routine, the option and file name are already set in the panel and the user can enter further specifications for compression (see 9.2).

The command is entered in the same line as the file name to be compressed:

FLCOMP / **or %FLCOMP** /

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 15 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND      NAME                                TRACKS %USED XT DEVICE
------------------------------------------------------------------------
FLCOMP / USER.DAT.F                                   1  100   1  3390
         USER.DAT.FB                                  1  100   1  3390
         USER.DAT.KSDS
         USER.DAT.KSDS.DATA
         USER.DAT.KSDS.INDEX
```

9.4 FLDECO

This CLIST procedure is intended for use in panel 3.4 in ISPF (file list), but it can also be called directly. In the latter case, the file name is requested.

To decompress the file specified in the line, FLDECO branches to the FLAM panel routine. In this routine, the option and file name are already set in the panel and the user can enter further specifications for decompression (see 9.2).

The command is entered in the same line as the file name to be decompressed:

FLDECO / **or %FLDECO** /

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 14 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND      NAME                              TRACKS %USED XT  DEVICE
------------------------------------------------------------------------
FLDECO / USER.DAT.CMP                              1  100   1  3390
         USER.DAT.F                                1  100   1  3390
         USER.DAT.FB                               1  100   1  3390
         USER.DAT.KSDS
         USER.DAT.KSDS.DATA
         USER.DAT.KSDS.INDEX
```

9.5 FLDIR

This CLIST procedure is intended for use in panel 3.4 in ISPF (file list), but it can also be called directly. In the latter case, the file name is requested.

FLDIR displays the FLAMFILE information about the file specified in the line (same as option I in the FLAM panels). The file is not decompressed. If no FLAMFILE exists, an appropriate message is written to the result list of FLAM.

The command is entered in the same line as the file name to get information about:

FLDIR / **or %FLDIR** /

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 14 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND      NAME                              TRACKS %USED XT  DEVICE
------------------------------------------------------------------------
%FLDIR / USER.DAT.CMP                              1  100   1  3390
         USER.DAT.F                                1  100   1  3390
         USER.DAT.FB                               1  100   1  3390
         USER.DAT.KSDS
         USER.DAT.KSDS.DATA
         USER.DAT.KSDS.INDEX
```

9.6 FLDISP

This CLIST procedure is intended for use in panel 3.4 in ISPF (file list), but it can also be called directly. In the latter case, the file name is requested.

FLDISP displays the contents of the file specified in the line. If a FLAMFILE exists, it is decompressed into a temporary file and this file is displayed. An uncompressed file is displayed directly, i.e. the command can be used for all files that can be displayed (same as function 1 (BROWSE) in ISPF).

The call can be supplemented with FLAM parameters for decompression. The command is entered in the same line as the file name to be displayed:

FLDISP / **or %FLDISP** /

or with parameters:

FLDISP / **PARM('FLAM-parameter')**

Without parameters, a sequential (PS) file is created by default. If a PO library has been compressed, a PO library can be created again by specifying the parameter 'PO'.

FLDISP / **PO PARM('FLAM parameter')**

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 15 OF 134
COMMAND ===>                                          SCROLL === PAGE

 COMMAND      NAME                                TRACKS %USED XT  DEVICE
 ------------------------------------------------------------------------
 FLDISP / USER.DAT.F                                   1  100   1  3390
          USER.DAT.FB                                  1  100   1  3390
          USER.DAT.KSDS.CMP                            0    ?   0  3390
          USER.DAT.KSDS.CMP.DTA                        4    ?   1  3390
          USER.DAT.KSDS.CMP.IDX                        1    ?   1  3390
```

FLDISP can also be used to display a VSAM FLAMFILE using the BROWSE function. Since the file is decompressed into a temporary (PS) file, it is also possible to include parameters for FLAM if necessary.

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 15 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND  NAME                                 TRACKS %USED XT  DEVICE
------------------------------------------------------------------------
         USER.DAT.F                              1  100   1  3390
         USER.DAT.FB                             1  100   1  3390
%FLDISP / PARM('ORECS=512,TRUNC=YES')            0    ?   0  3390
         USER.DAT.KSDS.CMP.DTA                   4    ?   1  3390
         USER.DAT.KSDS.CMP.IDX                   1    ?   1  3390
```

9.7 FLEDIT

This CLIST procedure is intended for use in panel 3.4 in ISPF (file list), but it can also be called directly. In the latter case, the file name is requested.

FLEDIT displays the contents of the file specified in the line and allows modifications to be made. If a FLAMFILE exists, it is decompressed into a temporary file and this file is then edited. An uncompressed file is edited directly, i.e. the command can be used for all files that can be edited (same as function 2 (EDIT) in ISPF).

If editing is terminated by means of 'CANCEL', the modifications made are not incorporated and the file selected is left as it was (same as EDIT in ISPF).

The call can be supplemented with FLAM parameters for compression and decompression.

The command is entered in the same line as the file name to be edited:

FLEDIT / **or %FLEDIT** /

or with parameters:

FLEDIT / **PARM('FLAM parameter')**

Note: If the decompressed (temporary) file is compressed again after having been edited, the original file header information is lost! The values of this temporary file are then used for the file header.

Without parameters, a sequential (PS) file is created by default. If a PO library has been compressed, a PO library can be created again by specifying the parameter 'PO'.

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 14 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND      NAME                                TRACKS %USED XT  DEVICE
------------------------------------------------------------------------
FLEDIT / USER.DAT.CMP                               1  100   1  3390
         USER.DAT.F                                 1  100   1  3390
         USER.DAT.FB                                1  100   1  3390
         USER.DAT.KSDS
         USER.DAT.KSDS.DATA
         USER.DAT.KSDS.INDEX
```

9.8 FLTOC

This CLIST procedure is meant for the use in the panel 3.4 in ISPF (file list), will also be activated internally by option 'I' in the FLAM panel.

FLTOC displays the content of a collecitve FLAMFILE analogue the output of ISPF 3.4 and allows direct display of FLAMFILE members and the decompression.

The input of the command is made in the line with the wanted file name:

FLTOC / **or** **%FLTOC** /

Example:

```
DSLIST - DATA SETS BEGINNING WITH USER -------------  ROW 14 OF 134
COMMAND ===>                                          SCROLL === PAGE

COMMAND      NAME                                TRACKS %USED XT  DEVICE
------------------------------------------------------------------------
FLTOC  / USER.DAT.CMP                              45  100   1  3390
         USER.DAT.F                                 1  100   1  3390
         USER.DAT.FB                                1  100   1  3390
         USER.DAT.KSDS
         USER.DAT.KSDS.DATA
         USER.DAT.KSDS.INDEX
```

Analogue option 'I' of the FLAM panel this will be displayed:

```
FLAMFILE TOC DAT.CMP                                              Row 1 of 2170
                MODE VR8    MAXBUFFER 64      FLAMCODE EBCD
   Original Data Set Name                      Dsorg Recfm Lrecl Blksi Space
-------------------------------------------------------------------------------
   FLAMT.AD0001NP.LIST                          SEQ   FBM   133   3059    300 K
   FLAMT.AD0001NP.CX8                           SEQ   FB    80    23440    50 K
   FLAMT.AD0191NP.LIST                          SEQ   FBM   133   3059    500 K
   FLAMT.AD0192NP.LIST                          SEQ   FBM   133   3059    250 K
   FLAMT.EXD4TO3.LIST                           SEQ   FBM   133   3059    150 K
   FLAMT.EXK1NUL.LIST                           SEQ   FBM   133   3059     50 K
   FLAMT.EXK3TO4.LIST                           SEQ   F     133   133     350 K
   FLAMT.FLAM.CMP                               SEQ   FB    512   23552 12800 K
   FLAMT.FLAMDIR.LIST                           SEQ   FBM   133   3059    200 K
   FLAMT.FLAMFLN.LIST                           SEQ   F     133   133    3150 K
   FLAMT.FLAMG001.LIST                          SEQ   FBM   133   3059   1250 K
   FLAMT.FLAMG002.LIST                          SEQ   F     133   133     500 K
   FLAMT.FLAMHELP.LIST                          SEQ   F     133   133     550 K
   FLAMT.FLAMNUC.LIST                           SEQ   F     133   133   11300 K
   FLAMT.FLAMTADC.LIST                          SEQ   FBM   133   3059    100 K
   FLAMT.FLAMTS.LIST                            SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                          SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                          SEQ   VB    260   23440 28500 K

COMMAND ===>
```

9.8.1 Browse a FLAMFILE member

```
FLAMFILE TOC DAT.CMP                                          Row 1 of 2170
                MODE VR8    MAXBUFFER 64      FLAMCODE EBCD
   Original Data Set Name                      Dsorg Recfm Lrecl Blksi Space
-------------------------------------------------------------------------------
   FLAMT.AD0001NP.LIST                         SEQ   FBM   133   3059    300 K
   FLAMT.AD0001NP.CX8                          SEQ   FB    80    23440    50 K
   FLAMT.AD0191NP.LIST                         SEQ   FBM   133   3059    500 K
   FLAMT.AD0192NP.LIST                         SEQ   FBM   133   3059    250 K
   .
   .
   FLAMT.FLAMTS01.DAT1                         SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                         SEQ   VB    260   23440 28500 K

COMMAND ===>
```

Input of 'B' in line FLAMT.FLAMTS01.DAT:

```
   FLAMT.FLAMTS.LIST                           SEQ   F     133   133     400 K
B  FLAMT.FLAMTS01.DAT1                         SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                         SEQ   VB    260   23440 28500 K

COMMAND ===>
```

causes a decompression of this member and the display by means of the ISPF browse function:

```
Browse Member of FLAMFILE DAT.CMP
 originally compressed on MVS
 FLAMT.FLAMTS01.DAT                                  Lines 00000000 Col 001 080
-------------------------------------------------------------------------------
******************************** Top of Data *********************************
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
.
.
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
19980202L000050010060     000021112850      0123456780000001          0000001
 Command ===>                                            Scroll ===> CSR
```

By input of 'BA' the files will be converted from ASCII to EBCDIC before displayed (according to the internal translation table A/E).

By input of ‚**BP**' you can enter additional parameter for decompressing.

```
   FLAMT.FLAMTS.LIST                          SEQ   F     133   133     400 K
BP FLAMT.FLAMTS01.DAT1                        SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                        SEQ   VB    260   23440 28500 K
```

Will lead to:

```
 Old system  : MVS
 Old data set: FLAMT.FLAMTS01.DAT1

     Parameter for decompression to browse this file

Cryptokey               (to decrypt the FLAMFILE)
:
SecureInfo  : MEMBER    (Ignore/Member/Yes)
                        use MEMBER for an AES encrypted FLAMFILE
Translation :           (A/E, E/A, module name of transl. table)
```

You have to enter the key for decryption, if the FLAMFILE has been encrypted during compression. Using AES-encryption, SECUREINFO should be set to MEMBER to verify the security information for the member only.

9.8.2 Information about a FLAMFILE member

```
FLAMFILE TOC DAT.CMP                                              Row 1 of 2170
                 MODE VR8    MAXBUFFER 64      FLAMCODE EBCD
   Original Data Set Name                        Dsorg Recfm Lrecl Blksi Space
-------------------------------------------------------------------------------
   FLAMT.AD0001NP.LIST                           SEQ   FBM   133   3059    300 K
   FLAMT.AD0001NP.CX8                            SEQ   FB    80    23440    50 K
   FLAMT.AD0191NP.LIST                           SEQ   FBM   133   3059    500 K
   FLAMT.AD0192NP.LIST                           SEQ   FBM   133   3059    250 K
   FLAMT.EXD4TO3.LIST                            SEQ   FBM   133   3059    150 K
   .
   .
   FLAMT.FLAMTS.LIST                             SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                           SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                           SEQ   VB    260   23440 28500 K

COMMAND ===>
```

Input of 'I' in line FLAMT.AD0001NP.CX8:

```
   FLAMT.AD0001NP.LIST                           SEQ   FBM   133   3059    300 K
I  FLAMT.AD0001NP.CX8                            SEQ   FB    80    23440    50 K
   FLAMT.AD0191NP.LIST                           SEQ   FBM   133   3059    500 K
```

will give out more information about this FLAMFILE member:

```
FLAMFILE TOC DAT.CMP                                              Row 1 of 2170
  +----------------- FLAMFILE INFORMATION -----------------+
  ¦                                                       ¦ recl Blksi Space
- ¦ FLAMT.AD0001NP.CX8                                    ¦ ------------------
  ¦                                                       ¦ 33   3059    300 K
I ¦ Data Set was compressed on MVS                        ¦ 0    23440    50 K
  ¦                                                       ¦ 33   3059    500 K
  ¦ Organization     ===>  PS                             ¦ 33   3059    250 K
  ¦ Record Format    ===>  FB                             ¦ 33   3059    150 K
  ¦ Record Length    ===>  80                             ¦ 33   3059     50 K
  ¦ Block Size       ===>  23440                          ¦ 33   133     350 K
  ¦ Rel. Key Pos.    ===>                                 ¦ 33   133   12800 K
  ¦ Key Length       ===>                                 ¦ 33   133    3150 K
  ¦ No.Dir.Blocks    ===>                                 ¦ 33   133    1250 K
  ¦ Space Amount     ===>  50    KB       1     TRKS      ¦ 33   133     500 K
  ¦                                                       ¦ 33   3059    550 K
  +-------------------------------------------------------+ 33   133   11300 K
   FLAMT.FLAMTADC.LIST                           SEQ   FBM   133   3059    100 K
   FLAMT.FLAMTS.LIST                             SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                           SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                           SEQ   VB    260   23440 28500 K

COMMAND ===>
```

9.8.3 Decompression of a FLAMFILE member

Input of 'S' in line FLAMT.DAT.CMP causes a decompression of this member. If the file is already catalogued another display will ask you to allow an overwriting:

```
FLAMFILE TOC DAT.CMP                                          Row 1 of 2170
  +------------------------ FLAM DECOMPRESSION -------------------------+
  ¦                                                                     ¦ ace
- ¦                                                                     ¦ -----
  ¦ 'FLAMT.FLAM.CMP'                                                    ¦ 300 K
  ¦                                                                     ¦  50 K
  ¦ is already cataloged.                                               ¦ 500 K
  ¦                                                                     ¦ 250 K
  ¦                                                                     ¦ 150 K
  ¦                                                                     ¦  50 K
  ¦ Overwrite ? ===> N  (Y/N)                                           ¦ 350 K
S +---------------------------------------------------------------------+ 800 K
   FLAMT.FLAMDIR.LIST                        SEQ   FBM   133   3059    200 K
   FLAMT.FLAMFLN.LIST                        SEQ   F     133   133    3150 K
   FLAMT.FLAMG001.LIST                       SEQ   FBM   133   3059   1250 K
   FLAMT.FLAMG002.LIST                       SEQ   F     133   133     500 K
   FLAMT.FLAMHELP.LIST                       SEQ   F     133   133     550 K
   FLAMT.FLAMNUC.LIST                        SEQ   F     133   133   11300 K
   FLAMT.FLAMTADC.LIST                       SEQ   FBM   133   3059    100 K
   FLAMT.FLAMTS.LIST                         SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                       SEQ   V     260   264     350 K
   FLAMT.FLAMTS02.DAT2                       SEQ   VB    260   23440 28500 K
```

If the question above is answered with 'N' or the file was not catalogued the following display is shown:

```
FLAMFILE TOC DAT.CMP                                          Row 1 of 2170
  +------------------------ FLAM DECOMPRESSION -------------------------+
  ¦                                                                     ¦ ace
- ¦ Old system  : MVS                                                  ¦ -----
  ¦ Old data set: FLAMT.FLAM.CMP                                        ¦ 300 K
  ¦                                                                     ¦  50 K
  ¦ New data set: 'FLAMT.FLAM.CMP'                                      ¦ 500 K
  ¦                                                                     ¦ 250 K
  ¦   Reuse existing data set:  N   (Y/N)                               ¦ 150 K
  ¦                                                                     ¦  50 K
  ¦   Record truncation:  N         (Y/N) allowed / not allowed)        ¦ 350 K
S ¦   Translation      :            (A/E, module name)                  ¦ 800 K
  ¦   SecureInfo       :  MEMBER    (Ignore/Member/Yes)                 ¦ 200 K
  ¦   CryptoKey                                                        ¦ 150 K
  ¦   :                                                                ¦ 250 K
  ¦   Submit:  F   (F/B, Foreground or Batch)                          ¦ 500 K
  ¦                                                                     ¦ 550 K
  ¦ Command ===>                                                        ¦ 300 K
  +---------------------------------------------------------------------+ 100 K
   FLAMT.FLAMTS.LIST                         SEQ   F     133   133     400 K
   FLAMT.FLAMTS01.DAT1                       SEQ   V     260   264     350 K
```

You have to enter the key for decryption, if the FLAMFILE has been encrypted during compression. Using AES-encryption, SECUREINFO should be set to MEMBER to verify the security information for the member only.

Above a new file name FLAM3.NEWDAT.LIST was entered, in the next display new attributes for this file can be allocated:

```
FLAMFILE TOC DAT.CMP                                          Row 1 of 2170
  +----------------------- FLAM DECOMPRESSION -------------------------+
  ¦ Data Set  'FLAM3.NEWDAT.LIST'                                      ¦ ace
- ¦                                                                     ¦ -----
  ¦   DATA SET WAS COMPRESSED ON MVS                                   ¦ 300 K
  ¦                                                                    ¦  50 K
  ¦ Organization     ===> PS       (PS/PO/ESDS/KSDS/RRDS/LDS)          ¦ 500 K
  ¦ Record Format    ===> FB       (F/FB/V/VB, with S,A,M, or /U)      ¦ 250 K
  ¦ Record Length    ===> 512         (up to 32760 Byte, avg. max VSAM) ¦ 150 K
  ¦ Block Size       ===> 23552    (up to 32760 Byte, CISZ for VSAM)    ¦  50 K
  ¦ Rel.Key.Pos.     ===>                       ( VSAM KSDS             ¦ 350 K
S ¦ Key Length       ===>          (up to 255) (        ONLY            ¦ 800 K
  ¦ No.Dir.Blocks    ===>          (PO data sets only)                  ¦ 200 K
  ¦ Space Unit       ===> TRKS     (BLKS, TRKS, CYLS, or RECS)          ¦ 150 K
  ¦ Primary Quantity ===> 256      (in above units)                     ¦ 250 K
  ¦ Secondary Quant. ===>          (in above units)                     ¦ 500 K
  ¦ Volume Serial    ===> MVSWK1                                       ¦ 550 K
  ¦ Generic Unit     ===> 3380                                         ¦ 300 K
  ¦                                                                    ¦ 100 K
  ¦ COMMAND ====>                                                      ¦ 400 K
  +----------------------------------------------------------------------+ 350 K
   EULER.FLAMTS02.DAT2                          SEQ   VB    260   3059  28500 K

 COMMAND ===>
```

If the decompression was regular it will be branched into the display menu. Otherwise the FLAM error protocol will be displayed.

FLAM (MVS)

User Manual

Appendix

Appendix

A.1 Code translation tables

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	
0.	NUL 00	SOH 01	STX 02	ETX 03	1A	HT 09	1A	DEL 7F	1A	1A	1A	VT 0B	FF 0C	CR 0D	SO 0E	SI 0F	**0.**
1.	DLE 10	DC1 11	DC2 12	DC3 13	1A	1A	BS 08	1A	CAN 18	EM 19	1A	1A	FS 1C	GS 1D	RS 1E	US 1F	**1.**
2.	1A	1A	1A	1A	1A	LF 0A	ETB 17	ESC 1B	1A	1A	1A	1A	1A	ENQ 05	ACK 06	BEL 07	**2.**
3.	1A	1A	SYN 16	1A	1A	1A	1A	EOT 04	1A	1A	1A	1A	DC4 14	NAK 15	1A	SUB 1A	**3.**
4.	SP 20	1A	1A	1A	1A	1A	1A	1A	1A	1A	[5B	. 2E	< 3C	(28	+ 2B	! 21	**4.**
5.	& 26	1A	1A	1A	1A	1A	1A	1A	1A	1A	] 5D	$ 24	* 2A	) 29	; 3B	5E	**5.**
6.	- 2D	/ 2F	1A	1A	1A	1A	1A	1A	1A	1A	¦ 7C	, 2C	% 25	_ 5F	> 3E	? 3F	**6.**
7.	1A	1A	1A	1A	1A	1A	1A	1A	1A	60	: 3A	# 23	@ 40	' 27	= 3D	" 22	**7.**
8.	1A	a 61	b 62	c 63	d 64	e 65	f 66	g 67	h 68	i 69	1A	1A	1A	1A	1A	1A	**8.**
9.	1A	j 6A	k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72	1A	1A	1A	1A	1A	1A	**9.**
A.	1A	~ 7E	s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A	1A	1A	1A	1A	1A	1A	**A.**
B.	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	**B.**
C.	{ 7B	A 41	B 42	C 43	D 44	E 45	F 46	G 47	H 48	I 49	1A	1A	1A	1A	1A	1A	**C.**
D.	} 7D	J 4A	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52	1A	1A	1A	1A	1A	1A	**D.**
E.	\ 5C	1A	S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A	1A	1A	1A	1A	1A	1A	**E.**
F.	0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37	8 38	9 39	1A	1A	1A	1A	1A	1A	**F.**
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	

Translation table from EBCDIC to ASCII

(TRANSLATE = E/A)

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	
0.	NUL 00	SOH 01	STX 02	ETX 03	EOT 37	ENQ 2D	ACK 2E	BEL 2F	BS 16	HT 05	LF 25	VT 0B	FF 0C	CR 0D	SO 0E	SI 0F	0.
1.	DLE 10	DC1 11	DC2 12	DC3 13	DC4 3C	NAK 3D	SYN 32	ETB 26	CAN 18	EM 19	SUB 3F	ESC 27	FS 1C	GS 1D	RS 1E	US 1F	1.
2.	SP 40	! 4F	" 7F	# 7B	$ 5B	% 6C	& 50	' 7D	(4D	) 5D	* 5C	+ 4E	, 6B	- 60	. 4B	/ 61	2.
3.	0 F0	1 F1	2 F2	3 F3	4 F4	5 F5	6 F6	7 F7	8 F8	9 F9	: 7A	; 5E	< 4C	= 7E	> 6E	? 6F	3.
4.	@ 7C	A C1	B C2	C C3	D C4	E C5	F C6	G C7	H C8	I C9	J D1	K D2	L D3	M D4	N D5	O D6	4.
5.	P D7	Q D8	R D9	S E2	T E3	U E4	V E5	W E6	X E7	Y E8	Z E9	Ž 4A	\ E0	! 5A	^ 5F	_ 6D	5.
6.	` 79	a 81	b 82	c 83	d 84	e 85	f 86	g 87	h 88	i 89	j 91	k 92	l 93	m 94	n 95	o 96	6.
7.	p 97	q 98	r 99	s A2	t A3	u A4	v A5	w A6	x A7	y A8	z A9	{ C0	¦ 6A	} D0	~ A1	DEL 07	7.
8.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	8.
9.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	9.
A.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	A.
B.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	B.
C.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	C.
D.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	D.
E.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	E.
F.	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	3F	F.
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	

Translation table from ASCII to EBCDIC

(TRANSLATE = A/E)

Explanation of abbreviations

ACK	=	acknowledge (positive)
BEL	=	bell
BS	=	backspace
CAN	=	cancel
CR	=	carriage return
DC1	=	device control 1
DC2	=	device control 2
DC3	=	device control 3, stop output
DC4	=	device control 4
DEL	=	delete
DLE	=	data link escape
EM	=	end of medium
ENQ	=	enquiry, station call
EOT	=	end of transmission
ESC	=	escape
ETB	=	end of transmission block
ETX	=	end of text
FF	=	form feed
FS	=	file separator
GS	=	group separater
HT	=	horizontal tabulation
LF	=	line feed
NAK	=	negative acknowledge,
NUL	=	null, no operation
RS	=	record separator
SI	=	shift in, switch back character set
SO	=	shift out, switch character set
SOH	=	start of heading
SP	=	space, blank
STX	=	start of text
SUB	=	substitute character
SYN	=	synchronous idle
US	=	unit separator
VT	=	vertical tabulation