FCPE-SPI

1

Generated by Doxygen 1.8.14

Contents

Chapter 1

FCPE - Service Provider Interface

This service provider interface enables the possibility to write custom pre- and post-processing exits for columns as part of the table support in FLAM. It can be used to implement arbitrary validation and transformation (e.g. tokenization, masking, encryption) tasks based on data items of a column.

The service provider interface can be invoked as one of the pre- or post-processing steps for a column. See example below:

```
read.record(... table(... row(...
col(format.fix(...)
    process.exit(library='libfcpe', function='validate_string', parameter='format=IBAN')
    process.exit(library='libfcpe', function='mask_string', parameter='offset=4 length=6')
    process.exit(library='libfcpe', function='add_checksum', parameter='method=crc')
    type.string(...)
))))
```

The different functions can have different parameter strings. All of them depend on the implementation of the custom column processing exit according to this service provider interface.

1.1 Implementation overview

The interface that must be implemented to create a custom exit is described in the FCPE function section. This interface function must be an external function of a DLL/SO or load module (z/OS). The (`PROCESS.EXIT()`) object is used to configure how to call the custom exit routine.

First, the DLL/SO is loaded based on the specified library name or default name. Then the function address is determined based on the provided function name or default name). For load modules on z/OS, the function name is the load module name and the library name must be empty. It is also possible to use DLLs on z/OS.

If the function pointer is determined, the exit driver calls the exit function with the function code for opening (see function codes). A function-code-specific parameter structure ((see FcpeOpn struct)) contains the parameter string from the exit call specification (`PROCESS.EXIT(PARAMETER="xxx")`). Additionally, a read or write indication flag and the table and column name are provided as input. If the column processing is invertible, an inverse parameter string can be returned by the exit implementation while opening which must be suitable to configure the exit so that the changes made by this exit are reverted. For text data, the CCSID (IBM - Coded Character Set Identifier) with the combined character form ((see this section)) is provided as 32 bit integer. If the character set is changed by the exit, then the CCSID parameter must be adjusted by the exit.

If the exit only performs validation tasks or updates the input data in-place, then a flag bit must be set to indicate that the output buffer is not used (see here).

The first parameter of the exit function contains a handle. When it is first called with the function code for opening (FCPE_FUNC_OPN), the handle's value is NULL. This handle can be used to store state information that may be needed during data processing by the exit (i.e. subsequent calls with the FCPE_FUNC_RUN code). The exit function is responsible for allocating memory for the handle (if needed) when called with the function code for opening (FCPE_FUNC_OPN) and must release it (and all other resources) when called with the function code for closing the exit (FCPE_FUNC_CLS).

When called with the run function code (FCPE_FUNC_RUN), the exit function gets the input data length and a pointer to the input data as input (see FcpeRun) and writes the output data and its length to the respective fields. If the exit function set the flag FCPE_FLAG_NOCPY during the opening call to indicate that it does not write any data to the output buffer (e.g. the data is only validated), the exit driver sets the output pointer to NULL and expects that the exit pointer sets the output pointer to the input pointer in the run function.

Nonetheless, the exit function is called once more with the FCPE_FUNC_CLS function code to allow the release of all resources.

All function code specific parameter structures contain a length and a buffer of size 1024 bytes for an error message. The error message is displayed to the user if the return code differs from zero.

1.2 Interface standards

The interface is built on one function with four parameters, a classic load module interface. All parameters of the function are call-by-reference. The function does not have a return value. The return code is the second parameter of the function. There are only two types of parameters:

```
POINTER:   pointer to an address (usually 32 (PIC S9(9) COMP) or 64 bit)
INTEGER:   pointer to a 32 bit number in two's complement (PIC S9(9) COMP)
```

The type INTEGER has local endianness. On mainframes, this is usually big endian and little endian on x86 platforms.

The POINTER type is used for the handle and the parameter structures. The handle is a black box to FLAM. The parameter structures are unions with the function code acting as selector (see structures section).

To support 64 bit RISC architectures, a strong alignment to 64 bit in the parameter structures is required. This requires a 32 bit dummy field behind each 32 bit length field. The trick with the dummy fields makes the parameter structures compatible between the different supported architectures of FLAM and prevents bus errors on certain platforms.

All strings passed to the exit function are null-terminated and the corresponding lengths are set to the string length in bytes without the null character. The exit driver does not expect null-termination from the exit function, but the length fields must be set correctly. For the error message or the inverse string, the length can be set to the size of the error message buffer (1024) or inverse string buffer, respectively, if a null-terminated string is returned by the exit function.

1.3 Return code handling

The return code of the function must be one of the defined FLAM return codes (see module return codes). If the exit function is called with the funtion code FCPE_FUNC_RUN, then FLMRTC_LEN must be returned if the output buffer provided by the exit driver is too small to write the result and set the output size to the minimum required buffer size. If the input is not formatted correctly, FLMRTC_FMT must be returned to support the format detection possibilities of FLAM. All other return codes will result in an error and FLAM aborts the remaining processing.

1.4 Coding example for an FCPE

The example below uses the memory to memory interface to implement a simple exit, where the from-to conversion string is used as parameter for the service provider. The example is written in C and it implements the three function codes as switch block. The service provider interface specification (FCPE.h) and the FLUC byte interface header (FLCBYT.h) must be included to implement the interface and use the byte interface for conversion, respectively. The exit function FCPETST implements the interface as specified here, using the FLUC memory to memory interface.

```
#include <stdio.h>

#include "FCPE.h"
#include "FLCBYT.h"

extern void FCPETST(
   void**         ppHdl,
   int*           piRetco,
   const int*     piFunc,
   TuFcpePar*     puPara)
{
   size_t         out;
   *piRetco=FLMRTC_OK;
   switch(*piFunc){
   case FCPE_FUNC_OPN:
      *ppHdl=fcbopenv(NULL,puPara->stOpn.pcPar);
      if(*ppHdl==NULL){
         *piRetco=fcberrno;
         snprintf(puPara->stOpn.acMsg,sizeof(puPara->stOpn.acMsg),"%s",fcberrtr());
         puPara->stOpn.uiMsg = strlen(puPara->stOpn.acMsg);
      }
      puPara->stOpn.uiIvr=0;
      puPara->stOpn.uiCcs=0;
      break;
   case FCPE_FUNC_RUN:
      out=puPara->stRun.uiOut;
      *piRetco=fcbconv(*ppHdl,puPara->stRun.uiInp,puPara->stRun.pcInp,&out,puPara->stRun.pcOut);
      puPara->stRun.uiOut=out;
      if(*piRetco){
         snprintf(puPara->stRun.acMsg,sizeof(puPara->stRun.acMsg),"%s",fcberrtr());
         puPara->stRun.uiMsg=strlen(puPara->stRun.acMsg);
      }
      break;
   case FCPE_FUNC_CLS:
      *piRetco = fcbclosev(*ppHdl);
      if(*piRetco){
         *piRetco=fcberrno;
         snprintf(puPara->stCls.acMsg,sizeof(puPara->stCls.acMsg),"%s",fcberrtr());
         puPara->stCls.uiMsg=strlen(puPara->stCls.acMsg);
      }
      break;
   default:
      *piRetco=FLMRTC_ITF;
      break;
   }
}
```

For simplicity of the example, it the CCSID is set to zero and no inverse string is returned. This simple implementation of the SPI is quite powerful, because it provides the complete conversion functionality of FLUC for each data item of a column.

This test implementation is also deployed as sample C source (FCPETST.c) in the installation package (SRCLIB↩ C/samples). It can be used as role model for other implementations.

The implementation of an exit does not have to be done in C, but the binary interface of the compiled and linked DLL/SO or load module must be compatible to the C calling convention.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Function codes

Function codes for column processing exit.

Macros

- #define FCPE_FUNC_OPN 1

  Function code for opening (used in the first call of the exit function)
- #define FCPE_FUNC_RUN 2

  Function code for processing data.
- #define FCPE_FUNC_CLS 3

  Function code for closing (used in the last call of the exit function)

5.1.1 Detailed Description

Function codes for column processing exit.

The function codes below are passed to the exit function by the exit driver as the third parameter. It signals the current state to the exit function which may act accordingly. The function code also defines the format of the parameter structure. Only the union member of TuFcpePar may be accessed that matches the function code.

5.1.2 Macro Definition Documentation

5.1.2.1 FCPE_FUNC_CLS

```
#define FCPE_FUNC_CLS 3
```

Function code for closing (used in the last call of the exit function)

5.1.2.2 FCPE_FUNC_OPN

```
#define FCPE_FUNC_OPN 1
```

Function code for opening (used in the first call of the exit function)

5.1.2.3 FCPE_FUNC_RUN

```
#define FCPE_FUNC_RUN 2
```

Function code for processing data.

5.2 Default names

Default names used for library and function name.

Macros

- #define FCPE_DEFLIB_NAME "libfcpe"

  Default library name.
- #define FCPE_DEFFUC_NAME "FLAMCPE"

  Default function name.

5.2.1 Detailed Description

Default names used for library and function name.

The default names for the DLL/SO and the function name (load module name) are used if no library name or function name are specified in the `process.exit()` object.

5.2.2 Macro Definition Documentation

5.2.2.1 FCPE_DEFFUC_NAME

```
#define FCPE_DEFFUC_NAME "FLAMCPE"
```

Default function name.

5.2.2.2 FCPE_DEFLIB_NAME

```
#define FCPE_DEFLIB_NAME "libfcpe"
```

Default library name.

5.3 Combined character forms

Defines the combined character form.

Macros

- #define FCPE_CMBFRM_NON 0U

  No combined character form.
- #define FCPE_CMBFRM_NFD 1U

  Normalization Form Canonical Decomposition.
- #define FCPE_CMBFRM_NFC 2U

  Normalization Form Canonical Composition.

5.3.1 Detailed Description

Defines the combined character form.

The CCSID field in the parameter structure TsFcpeOpn contains one of these combined character forms (Unicode equivalence) in the 8 highest order bits. If the form is not known then FCPE_CMBFRM_NON should be used.

5.3.2 Macro Definition Documentation

5.3.2.1 FCPE_CMBFRM_NFC

```
#define FCPE_CMBFRM_NFC 2U
```

Normalization Form Canonical Composition.

5.3.2.2 FCPE_CMBFRM_NFD

```
#define FCPE_CMBFRM_NFD 1U
```

Normalization Form Canonical Decomposition.

5.3.2.3 FCPE_CMBFRM_NON

```
#define FCPE_CMBFRM_NON 0U
```

No combined character form.

5.4 Flags for FCPE processing

Definitions for the flag word in the parameter structure used for open.

Macros

- #define FCPE_FLAG_WRITE 0x00000001U

  [INPUT] If set by exit driver, this signals that the exit is called as post-processing on the writing side
- #define FCPE_FLAG_NOCPY 0x00000002U

  [OUTPUT] Must be set by the exit function if it does not write data to the output buffer (validation or in-place manipulation of input)

5.4.1 Detailed Description

Definitions for the flag word in the parameter structure used for open.

In the open function the first word of the parameter structure contains few bit settings.

5.4.2 Macro Definition Documentation

5.4.2.1 FCPE_FLAG_NOCPY

```
#define FCPE_FLAG_NOCPY 0x00000002U
```

[OUTPUT] Must be set by the exit function if it does not write data to the output buffer (validation or in-place manipulation of input)

5.4.2.2 FCPE_FLAG_WRITE

```
#define FCPE_FLAG_WRITE 0x00000001U
```

[INPUT] If set by exit driver, this signals that the exit is called as post-processing on the writing side

5.5 Function structures

Definition of the function code dependent structures for FCPE.

Data Structures

- struct FcpeOpn

  FCPE parameter structure for function code FCPE_FUNC_OPN.
- struct FcpeRun

  FCPE Run structure.
- struct FcpeCls

  FCPE Close structure.
- union FcpePar

  FCPE Union of function structures.

Typedefs

- typedef struct FcpeOpn TsFcpeOpn

  FCPE parameter structure for function code FCPE_FUNC_OPN.
- typedef struct FcpeRun TsFcpeRun

  FCPE Run structure.
- typedef struct FcpeCls TsFcpeCls

  FCPE Close structure.
- typedef union FcpePar TuFcpePar

  FCPE Union of function structures.

5.5.1 Detailed Description

Definition of the function code dependent structures for FCPE.

5.5.2 Typedef Documentation

5.5.2.1 TsFcpeCls

```
typedef struct FcpeCls TsFcpeCls
```

FCPE Close structure.

This structure is used when the exit function is called with the function code FCPE_FUNC_CLS. It is always called after data processing is finished, no matter if there was error or not.

The only member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

5.5.2.2 TsFcpeOpn

```
typedef struct FcpeOpn TsFcpeOpn
```

FCPE parameter structure for function code FCPE_FUNC_OPN.

This structure is used when the exit function is called with the function code FCPE_FUNC_OPN.

The first member contains flags that can be set in both directions. If the exit is opened for converting output data, the flag bit FCPE_FLAG_WRITE is set. If it is not set, the exit is opened for converting input data. The exit function may set the FCPE_FLAG_NOCPY flag if it does not write to the output buffer in the TsFcpeRun structure (i.e. it only validates or changes the data in-place).

The second member contains the CCSID in the low order 24 bits. The high order 8 bits contain the combined character form. The member is set by the exit driver to a value corresponding to the input data and must be changed by the exit function only if its output is in a different character set. A CCSID of 0 indicates binary data. For example, if the exit function implements encryption of strings, the input text may be in UTF-8 (CCSID=1208). Since the encryption produces binary output, the CCSID must be set to 0.

The next three member pairs contain the length and a pointer to the corresponding string with the table name, the column name and the parameter string. The parameter string must be interpreted by the service provider.

The following member pair may be set by the exit function if the operation is invertible. The exit function must the pointer to the inverse parameter string and is responsible for allocating and freeing memory for this string. If the length is zero or the pointer is NULL, the exit is considered not invertible.

The last member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

5.5.2.3 TsFcpeRun

```
typedef struct FcpeRun TsFcpeRun
```

FCPE Run structure.

This structure is used when the exit function is called with the function code FCPE_FUNC_RUN. It is called for each data item of this column, allowing the exit to manipulate the data as needed.

The exit drivers passes the length and a pointer to the input data of the current column as the first pair of members of the structure.

The second member pair is the size and a pointer to an already allocated buffer where the exit function may write its output to and set the output length accordingly. The input and output buffers are managed by the exit driver. If the output buffer is too small, then FLMRTC_LEN must be set as return code and the output length must contain the minimum required buffer length. The exit driver then repeats the call with a buffer of at least the desired size. This is repeated until the exit function no longer sets the FLMRTC_LEN error code. If the FCPE_FLAG_NOCPY flag is set, the output pointer and length is set to NULL/0 by the exit driver and the exit function must set the output length to the input length and the output pointer to the input pointer for plausibility checks by the exit driver.

If the FCPE_FLAG_NOCPY flag has been set during the FCPE_FUNC_RUN function code call, the input buffer may be modified in-place. However, the output pointer must be set to the input pointer. This prevents unnessary copy operation if the exit function only perform minor changes or none at all.

The flag word can be used to set indication flags bits when reading and to use these flags when writing. Currently, the bit definitions below are defined:

FLMELM_TYPTAB_FLAG_NULLID FLMELM_TYPTAB_FLAG_EMPTID

At normal pre- or post-processing the flag word is not touched.

The last member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

5.5.2.4 TuFcpePar

```
typedef union FcpePar TuFcpePar
```

FCPE Union of function structures.

This union must be used to access structure fields depending on the function code that was passed to exit function. Only the member that corresponds to the function code may be accessed. Failing to do so results in undefined behavior!

5.6 FCPE function

This is the only function which must be provided by a DLL/SO or load module to fulfill the FLAM Column Processing Exit (FCPE) Service Provider Interface (SPI).

Typedefs

- typedef void(∗ TpfFcpExit) (void ∗∗ppHdl, int ∗piRetco, const int ∗piFunc, TuFcpePar ∗puPara)

  FCPE function.

5.6.1 Detailed Description

This is the only function which must be provided by a DLL/SO or load module to fulfill the FLAM Column Processing Exit (FCPE) Service Provider Interface (SPI).

5.6.2 Typedef Documentation

5.6.2.1 TpfFcpExit

```
typedef void(* TpfFcpExit) (void **ppHdl, int *piRetco, const int *piFunc, TuFcpePar *puPara)
```

FCPE function.

This is the function which must be implemented for user specific column processing. The same handle is passed to consecutive calls and may be used by the exit function to store internal state.

The return code must be set as output parameter by the exit function. The function code is an input parameter and contains one of the values defined in this section. Depending on the provided function code, a pointer to a specific parameter structure is given as last parameter.

Parameters

in,out	*ppHdl*	POINTER Service provider handle
out	*piRetco*	INTEGER Return code
in	*piFunc*	INTEGER Function code (OPN/RUN/CLS)
in,out	*puPara*	POINTER Parameter structure

5.7 Output format

Constants for setting the output format for some functions.

Macros

- #define FLMOUT_FORMAT_NON 0

  Default format (FLMOUT_FORMAT_LST)
- #define FLMOUT_FORMAT_LST 1

  Simple list of lines with each line separated by '\n' and the whole string terminated by 0x00.
- #define FLMOUT_FORMAT_XML 2

  As XML-formatted, null-terminated string (in local charset) with each line separated by '\n'.

5.7.1 Detailed Description

Constants for setting the output format for some functions.

Pass one of the constants below to all functions requiring output format (e.g. fcbclose2()), for formatting of the output in different ways. This is mainly required for the statistic and other lists.

5.7.2 Macro Definition Documentation

5.7.2.1 FLMOUT_FORMAT_LST

```
#define FLMOUT_FORMAT_LST 1
```

Simple list of lines with each line separated by '\n' and the whole string terminated by 0x00.

5.7.2.2 FLMOUT_FORMAT_NON

```
#define FLMOUT_FORMAT_NON 0
```

Default format (FLMOUT_FORMAT_LST)

5.7.2.3 FLMOUT_FORMAT_XML

```
#define FLMOUT_FORMAT_XML 2
```

As XML-formatted, null-terminated string (in local charset) with each line separated by '\n'.

5.8 Syntax and help

Syntax and help constants for FLUC byte and record interface.

Macros

- #define FLC_READ_FILE 1
  File definition strings in read mode.
- #define FLC_READ_FORMAT 2
  Format data strings in read mode.
- #define FLC_WRITE_FILE 3
  File definition strings in write mode.
- #define FLC_WRITE_FORMAT 4
  Format data strings in write mode.
- #define FLC_CONV_READ 5
  Data conversion strings in read mode.
- #define FLC_CONV_WRITE 6
  Data conversion strings in write mode.
- #define FLC_CONV_FROM_TO 7
  Data conversion string in from-to mode.
- #define FLC_INPUT_FILE 8
  File definition strings in input mode.
- #define FLC_OUTPUT_FILE 9
  File definition strings in output mode.
- #define FLC_INFO 10
  Definition of the info string.
- #define FLC_STATE 11
  Definition of the state string.
- #define FLC_LOG 12
  Definition of the memory log string.

5.8.1 Detailed Description

Syntax and help constants for FLUC byte and record interface.

Pass this to fcbhelp()/FCRHLP() or fcbsyntax()/FCRSYN() as parameter 'what' to get help or syntax information about the different parameter strings used to read, write or format data or about the state string.

5.8.2 Macro Definition Documentation

5.8.2.1 FLC_CONV_FROM_TO

```
#define FLC_CONV_FROM_TO 7
```

Data conversion string in from-to mode.

5.8.2.2 FLC_CONV_READ

```
#define FLC_CONV_READ 5
```

Data conversion strings in read mode.

5.8.2.3 FLC_CONV_WRITE

```
#define FLC_CONV_WRITE 6
```

Data conversion strings in write mode.

5.8.2.4 FLC_INFO

```
#define FLC_INFO 10
```

Definition of the info string.

5.8.2.5 FLC_INPUT_FILE

```
#define FLC_INPUT_FILE 8
```

File definition strings in input mode.

5.8.2.6 FLC_LOG

```
#define FLC_LOG 12
```

Definition of the memory log string.

5.8.2.7 FLC_OUTPUT_FILE

```
#define FLC_OUTPUT_FILE 9
```

File definition strings in output mode.

5.8.2.8 FLC_READ_FILE

```
#define FLC_READ_FILE 1
```

File definition strings in read mode.

5.8.2.9 FLC_READ_FORMAT

```
#define FLC_READ_FORMAT 2
```

Format data strings in read mode.

5.8.2.10 FLC_STATE

```
#define FLC_STATE 11
```

Definition of the state string.

5.8.2.11 FLC_WRITE_FILE

```
#define FLC_WRITE_FILE 3
```

File definition strings in write mode.

5.8.2.12 FLC_WRITE_FORMAT

```
#define FLC_WRITE_FORMAT 4
```

Format data strings in write mode.

5.9 methods

Defines integer values for the different hash methods.

Macros

- #define FLCHSH_ALGO_NON 0U
  No valid hash method.
- #define FLCHSH_ALGO_MD5 1U
  MD5 method.
- #define FLCHSH_ALGO_RMD128 2U
  RipeMd-128 method.
- #define FLCHSH_ALGO_RMD160 3U
  RipeMd-160 method.
- #define FLCHSH_ALGO_SHA1 10U
  SHA1 method.
- #define FLCHSH_ALGO_SHA160 10U
  SHA1 method.
- #define FLCHSH_ALGO_SHA224 20U
  SHA224 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA256 21U
  SHA256 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA384 22U
  SHA384 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA512 23U
  SHA512 (SHA2 variant) method.
- #define FLCHSH_ALGO_CRC08 100U
  8 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC16 101U
  16 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC24 102U
  24 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC32 103U
  32 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC32C 104U
  32 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC40 105U
  40 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC64 106U
  64 Bit CRC checksum (no crypto quality)

5.9.1 Detailed Description

Defines integer values for the different hash methods.

Pass this to the hash functions to choose the hash procedure.

5.9.2 Macro Definition Documentation

5.9.2.1 FLCHSH_ALGO_CRC08

```
#define FLCHSH_ALGO_CRC08 100U
```

8 Bit CRC checksum (no crypto quality)

5.9.2.2 FLCHSH_ALGO_CRC16

```
#define FLCHSH_ALGO_CRC16 101U
```

16 Bit CRC checksum (no crypto quality)

5.9.2.3 FLCHSH_ALGO_CRC24

```
#define FLCHSH_ALGO_CRC24 102U
```

24 Bit CRC checksum (no crypto quality)

5.9.2.4 FLCHSH_ALGO_CRC32

```
#define FLCHSH_ALGO_CRC32 103U
```

32 Bit CRC checksum (no crypto quality)

5.9.2.5 FLCHSH_ALGO_CRC32C

```
#define FLCHSH_ALGO_CRC32C 104U
```

32 Bit CRC checksum (no crypto quality)

5.9.2.6 FLCHSH_ALGO_CRC40

```
#define FLCHSH_ALGO_CRC40 105U
```

40 Bit CRC checksum (no crypto quality)

5.9.2.7 FLCHSH_ALGO_CRC64

```
#define FLCHSH_ALGO_CRC64 106U
```

64 Bit CRC checksum (no crypto quality)

5.9.2.8 FLCHSH_ALGO_MD5

```
#define FLCHSH_ALGO_MD5 1U
```

MD5 method.

5.9.2.9 FLCHSH_ALGO_NON

```
#define FLCHSH_ALGO_NON 0U
```

No valid hash method.

5.9.2.10 FLCHSH_ALGO_RMD128

```
#define FLCHSH_ALGO_RMD128 2U
```

RipeMd-128 method.

5.9.2.11 FLCHSH_ALGO_RMD160

```
#define FLCHSH_ALGO_RMD160 3U
```

RipeMd-160 method.

5.9.2.12 FLCHSH_ALGO_SHA1

```
#define FLCHSH_ALGO_SHA1 10U
```

SHA1 method.

5.9.2.13 FLCHSH_ALGO_SHA160

```
#define FLCHSH_ALGO_SHA160 10U
```

SHA1 method.

5.9.2.14 FLCHSH_ALGO_SHA224

```
#define FLCHSH_ALGO_SHA224 20U
```

SHA224 (SHA2 variant) method.

5.9.2.15 FLCHSH_ALGO_SHA256

```
#define FLCHSH_ALGO_SHA256 21U
```

SHA256 (SHA2 variant) method.

5.9.2.16 FLCHSH_ALGO_SHA384

```
#define FLCHSH_ALGO_SHA384 22U
```

SHA384 (SHA2 variant) method.

5.9.2.17 FLCHSH_ALGO_SHA512

```
#define FLCHSH_ALGO_SHA512 23U
```

SHA512 (SHA2 variant) method.

5.10 methods

Defines integer values for the different methods for calculation a message authentication code.

Macros

- #define FLCMAC_ALGO_NON 0U

  No valid hash method.
- #define FLCMAC_ALGO_HMAC 1U

  HMAC conform to RFC 2104.

5.10.1 Detailed Description

Defines integer values for the different methods for calculation a message authentication code.

Pass this to the hash functions to choose the key related procedure.

5.10.2 Macro Definition Documentation

5.10.2.1 FLCMAC_ALGO_HMAC

```
#define FLCMAC_ALGO_HMAC 1U
```

HMAC conform to RFC 2104.

5.10.2.2 FLCMAC_ALGO_NON

```
#define FLCMAC_ALGO_NON 0U
```

No valid hash method.

5.11 values

Defines integer values to define which statistical accounting value is requested by function fcbcount() or FCRCNT().

Macros

- #define FLC_INBOUND_COUNT 1

  Use input values.
- #define FLC_OUTBOUND_COUNT 2

  Use output values.
- #define FLC_FIO_COUNT 1

  Use values from File I/O.
- #define FLC_FMT_COUNT 2

  Use values from formatting.
- #define FLC_BYTE_COUNT 1

  Determine byte counts.
- #define FLC_UNIT_COUNT 2

  Determine unit (block or record) counts.

5.11.1 Detailed Description

Defines integer values to define which statistical accounting value is requested by function fcbcount() or FCRCNT().

5.11.2 Macro Definition Documentation

5.11.2.1 FLC_BYTE_COUNT

```
#define FLC_BYTE_COUNT 1
```

Determine byte counts.

5.11.2.2 FLC_FIO_COUNT

```
#define FLC_FIO_COUNT 1
```

Use values from File I/O.

5.11.2.3 FLC_FMT_COUNT

```
#define FLC_FMT_COUNT 2
```

Use values from formatting.

5.11.2.4 FLC_INBOUND_COUNT

```
#define FLC_INBOUND_COUNT 1
```

Use input values.

5.11.2.5 FLC_OUTBOUND_COUNT

```
#define FLC_OUTBOUND_COUNT 2
```

Use output values.

5.11.2.6 FLC_UNIT_COUNT

```
#define FLC_UNIT_COUNT 2
```

Determine unit (block or record) counts.

5.12 Matrix types

All available matrix type definitions (data formats)

Macros

- #define FLMMAT_TYP_NON 0x00

  Unknown/undefined matrix type.
- #define FLMMAT_TYP_DATBLK 0x80

  Data blocks, 1 element type, no attributes.
- #define FLMMAT_TYP_STDREC 0x81

  Standard record, 1 element type, no attributes.
- #define FLMMAT_TYP_STDASAREC 0x82

  Standard record, 1 element type, with 1 byte ASA control character as attribute.
- #define FLMMAT_TYP_STDMCCREC 0x83

  Standard record, 1 element type, with 1 byte machine control character as attribute.
- #define FLMMAT_TYP_RELREC 0x84

  Relative record, 1 element type, with 8 byte integer (slot number) as attribute.
- #define FLMMAT_TYP_RELASAREC 0x85

  Relative record, 1 element type, with 8 byte slot number and 1 byte ASA as attribute.
- #define FLMMAT_TYP_RELMCCREC 0x86

  Relative record, 1 element type, with 8 byte slot number and 1 byte MCC as attribute.
- #define FLMMAT_TYP_TXTREC 0x87

  Text record, 1 element type, no attributes (like standard record, but text is known)
- #define FLMMAT_TYP_TXTDLM 0x88

  Text record with delimiter at the end, 1 element type, no attributes.
- #define FLMMAT_TYP_TXTRST 0x90

  Text record and rest element (suppressed trailing whitespace and original delimiter), 2 elements, no attributes.
- #define FLMMAT_TYP_XMLELM 0x91

  XML elements, no attributes.
- #define FLMMAT_TYP_TABELM 0x92

  Table elements, no attributes.

5.12.1 Detailed Description

All available matrix type definitions (data formats)

A matrix is a complex data structure used to manage a list of neutral FLAM5 elements. All element lists of a member/file must be built based on the same matrix type.

The matrix type defines a neutral data format which is used to manage the data elements inside of FLAM.

5.12.2 Macro Definition Documentation

5.12.2.1 FLMMAT_TYP_DATBLK

```
#define FLMMAT_TYP_DATBLK 0x80
```

Data blocks, 1 element type, no attributes.

5.12.2.2 FLMMAT_TYP_NON

```
#define FLMMAT_TYP_NON 0x00
```

Unknown/undefined matrix type.

5.12.2.3 FLMMAT_TYP_RELASAREC

```
#define FLMMAT_TYP_RELASAREC 0x85
```

Relative record, 1 element type, with 8 byte slot number and 1 byte ASA as attribute.

5.12.2.4 FLMMAT_TYP_RELMCCREC

```
#define FLMMAT_TYP_RELMCCREC 0x86
```

Relative record, 1 element type, with 8 byte slot number and 1 byte MCC as attribute.

5.12.2.5 FLMMAT_TYP_RELREC

```
#define FLMMAT_TYP_RELREC 0x84
```

Relative record, 1 element type, with 8 byte integer (slot number) as attribute.

5.12.2.6 FLMMAT_TYP_STDASAREC

```
#define FLMMAT_TYP_STDASAREC 0x82
```

Standard record, 1 element type, with 1 byte ASA control character as attribute.

5.12.2.7 FLMMAT_TYP_STDMCCREC

```
#define FLMMAT_TYP_STDMCCREC 0x83
```

Standard record, 1 element type, with 1 byte machine control character as attribute.

5.12.2.8 FLMMAT_TYP_STDREC

```
#define FLMMAT_TYP_STDREC 0x81
```

Standard record, 1 element type, no attributes.

5.12.2.9 FLMMAT_TYP_TABELM

```
#define FLMMAT_TYP_TABELM 0x92
```

Table elements, no attributes.

5.12.2.10 FLMMAT_TYP_TXTDLM

```
#define FLMMAT_TYP_TXTDLM 0x88
```

Text record with delimiter at the end, 1 element type, no attributes.

5.12.2.11 FLMMAT_TYP_TXTREC

```
#define FLMMAT_TYP_TXTREC 0x87
```

Text record, 1 element type, no attributes (like standard record, but text is known)

5.12.2.12 FLMMAT_TYP_TXTRST

```
#define FLMMAT_TYP_TXTRST 0x90
```

Text record and rest element (suppressed trailing whitespace and original delimiter), 2 elements, no attributes.

5.12.2.13 FLMMAT_TYP_XMLELM

```
#define FLMMAT_TYP_XMLELM 0x91
```

XML elements, no attributes.

5.13 Element types

Element type definitions by matrix type.

Modules

- Block

  Block element types (see: FLMMAT_TYP_DATBLK)
- Record

  Record element types (see: FLMMAT_TYP_STDREC etc.)
- Text

  Text element types (see: FLMMAT_TYP_TXTREC etc.)
- XML

  XML element types (see: FLMMAT_TYP_XMLELM)
- TAB

  Table element types (see: FLMMAT_TYP_TABELM)

Macros

- #define FLMELM_NOSKIP -1

  Use this value for an element type if no skip of an element type required at read.

5.13.1 Detailed Description

Element type definitions by matrix type.

Each FLAM 5 matrix can consist of different element types. The interpretation of the different element types or which element types are valid depends on the matrix type. The element type 0 is mainly used for the net data of a matrix type.

5.13.2 Macro Definition Documentation

5.13.2.1 FLMELM_NOSKIP

```
#define FLMELM_NOSKIP -1
```

Use this value for an element type if no skip of an element type required at read.

5.14 Block

Block element types (see: FLMMAT_TYP_DATBLK)

Macros

- #define FLMELM_TYPBLK_STANDARD 0

  Simple data block element type (may contain binary data)

5.14.1 Detailed Description

Block element types (see: FLMMAT_TYP_DATBLK)

A matrix of blocks is a list of chunks containing an arbitrary type of data (probably binary). This matrix type only has the element type zero.

5.14.2 Macro Definition Documentation

5.14.2.1 FLMELM_TYPBLK_STANDARD

```
#define FLMELM_TYPBLK_STANDARD 0
```

Simple data block element type (may contain binary data)

5.15 Record

Record element types (see: FLMMAT_TYP_STDREC etc.)

Macros

- #define FLMELM_TYPREC_STANDARD 0

  Record element type for record matrix types.

5.15.1 Detailed Description

Record element types (see: FLMMAT_TYP_STDREC etc.)

The different kinds of records are separated by different matrix types. Each of these matrix types only have the element type zero.

5.15.2 Macro Definition Documentation

5.15.2.1 FLMELM_TYPREC_STANDARD

```
#define FLMELM_TYPREC_STANDARD 0
```

Record element type for record matrix types.

5.16 Text

Text element types (see: FLMMAT_TYP_TXTREC etc.)

Macros

- #define FLMELM_TYPTXT_RECORD 0

  Text element type of a TXTRST matrix.
- #define FLMELM_TYPTXT_REST 1

  Rest element type of a TXTRST matrix.

5.16.1 Detailed Description

Text element types (see: FLMMAT_TYP_TXTREC etc.)

A text matrix has two element types. The element type zero is the net text record and the element type one is the rest element. The rest element holds the text delimiter and (depending on the text parsing parameters) it could contain trailing whitespace.

5.16.2 Macro Definition Documentation

5.16.2.1 FLMELM_TYPTXT_RECORD

```
#define FLMELM_TYPTXT_RECORD 0
```

Text element type of a TXTRST matrix.

5.16.2.2 FLMELM_TYPTXT_REST

```
#define FLMELM_TYPTXT_REST 1
```

Rest element type of a TXTRST matrix.

5.17 XML

XML element types (see: FLMMAT_TYP_XMLELM)

Macros

- #define FLMELM_TYPXML_DATA 0

  XML data element type.
- #define FLMELM_TYPXML_STARTELM 1

  Start of opening XML tag element type.
- #define FLMELM_TYPXML_ENDSTRTELM 2

  End of opening XML tag element type.
- #define FLMELM_TYPXML_ENDELM 3

  Closing XML tag element type.
- #define FLMELM_TYPXML_ATTNAME 4

  Attribute name element type.
- #define FLMELM_TYPXML_ATTRVAL 5

  Attribute value element type.
- #define FLMELM_TYPXML_XML 6

  XML prolog element type.
- #define FLMELM_TYPXML_SKIPENT 7

  General skipped entity element type.
- #define FLMELM_TYPXML_SKIPPARMENT 8

  Skipped parameter entity element type.
- #define FLMELM_TYPXML_STARTDTD 9

  Start of a DTD (Document Type Definition) element type.
- #define FLMELM_TYPXML_ENDDTD 10

  End of a DTD element type.
- #define FLMELM_TYPXML_ELMDECL 11

  DTD element type declaration element type.
- #define FLMELM_TYPXML_ATTLDECL 12

  DTD attribute list declaration element type.
- #define FLMELM_TYPXML_INTENTDECL 13

  Internal entity declaration element type.
- #define FLMELM_TYPXML_SYSENTDECL 14

  External entity declaration element type with system identifier.
- #define FLMELM_TYPXML_PUBENTDECL 15

  External entity declaration element type with public identifier.
- #define FLMELM_TYPXML_NOTDECLS 16

  Notation declaration element type with a system identifier.
- #define FLMELM_TYPXML_NOTDECLP 17

  Notation declaration element type with a public identifier.
- #define FLMELM_TYPXML_NOTDECLPS 18

  Notation declaration element type with a public and system identifier.
- #define FLMELM_TYPXML_PROCINST 19

  Processing instruction element type.
- #define FLMELM_TYPXML_STARTCD 20

  Start of CDATA section element type.
- #define FLMELM_TYPXML_ENDCD 21

  Start of CDATA section element type.
- #define FLMELM_TYPXML_COMMENT 22

  Comment element type.
- #define FLMELM_TYPXML_DEFAULT 23

  Element type for all other data in a document.

5.17.1 Detailed Description

XML element types (see: FLMMAT_TYP_XMLELM)

A XML matrix consists of a number of different element types of which each reflects one part of the XML specification. Some contain single strings, others contain multiple strings separated by.

5.17.2 Macro Definition Documentation

5.17.2.1 FLMELM_TYPXML_ATTLDECL

```
#define FLMELM_TYPXML_ATTLDECL 12
```

DTD attribute list declaration element type.

Attribute lists describe constraints on attributes of XML nodes. This example defines a required attribute "src" for XML nodes named "img". The attribute value must contain character data:

```
<!ATTLIST img src CDATA #REQUIRED>
```

Each attribute list declaration element describes one attribute. The element's payload consists of five attributes separated by a NUL character:

```
<node_name>\0<attribute_name>\0<attribute_type>\0<default>\0<flag>
```

- `node_name` is the name of the XML node the attribute belongs to
- `attribute_name` is the name of the constrained attribute
- `attribute_type` contains a string with the type of the attribute (see XML spec for valid values)
- `default` specifies a default value if `flag` is set to D or F
- `flag` is one byte describing an attribute mode with one of four values:
  - R (attribute is required, i.e. must be specified)
  - I (attribute is implied, i.e it is optional and without default value)
  - F (attribute must always have the default value)
  - D (attribute has a specified default value)

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

5.17.2.2 FLMELM_TYPXML_ATTNAME

```
#define FLMELM_TYPXML_ATTNAME 4
```

Attribute name element type.

The name of an attribute within an opening XML tag. Must be immediately followed by an element of type FLMELM_TYPXML_ATTRVAL.

This element type may only occur between pairs of elements of types FLMELM_TYPXML_STARTELM and FLMELM_TYPXML_ENDELM.

5.17.2.3 FLMELM_TYPXML_ATTRVAL

```
#define FLMELM_TYPXML_ATTRVAL 5
```

Attribute value element type.

The value of an attribute within an opening XML tag. Must be immediately preceded by an element of type FLMELM_TYPXML_ATTNAME.

This element type may only occur between pairs of elements of types FLMELM_TYPXML_STARTELM and FLMELM_TYPXML_ENDELM.

5.17.2.4 FLMELM_TYPXML_COMMENT

```
#define FLMELM_TYPXML_COMMENT 22
```

Comment element type.

XML comment is arbitrary text data enclosed by <!– and –>. This element type contains the comment's data as payload. The comment data may not contain the character sequence –> as this marks the end of a comment.

5.17.2.5 FLMELM_TYPXML_DATA

```
#define FLMELM_TYPXML_DATA 0
```

XML data element type.

Contains the character data of an XML node. More specifically, it holds the character sequence found between an opening and closing tag or within a CDATA section. There may be multiple consecutive XML data elements, possibly interrupted by other elements like children XML nodes, comments and other entities. Data elements returned by FLAM through one of the APIs' read functions are at least 1024 bytes long if the XML node contains that many bytes. For XML nodes containing less than 1024 bytes of character data, the data is guaranteed to be returned in one element (i.e. is not split to multiple elements).

5.17.2.6 FLMELM_TYPXML_DEFAULT

```
#define FLMELM_TYPXML_DEFAULT 23
```

Element type for all other data in a document.

This element type contains the data not covered by one of the other elements. This includes whitespace, tabulators and line breaks that are not part of a nodes content, in other words: ignorable whitespace.

5.17.2.7 FLMELM_TYPXML_ELMDECL

```
#define FLMELM_TYPXML_ELMDECL 11
```

DTD element type declaration element type.

Element type declarations put constraints on XML nodes. For example, this declaration requires an XML node named "html" to have to children "head" and "body":

```
<!ELEMENT html (head, body)>
```

The element's payload consists of two attributes separated by a NUL character:

```
<name>\0<model>
```

- `name` is the XML node name
- `model` is a model string complying to the content specification syntax described in the XML standard

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

5.17.2.8 FLMELM_TYPXML_ENDCD

```
#define FLMELM_TYPXML_ENDCD 21
```

Start of CDATA section element type.

Marks the end of a CDATA section that has previously been started by a FLMELM_TYPXML_STARTCD element.

This element carries no payload, i.e. its data length is zero.

See also

FLMELM_TYPXML_STARTCD

5.17.2.9 FLMELM_TYPXML_ENDDTD

```
#define FLMELM_TYPXML_ENDDTD 10
```

End of a DTD element type.

Marks the end of a document type definition previously started by a FLMELM_TYPXML_STARTDTD element. It must occur before the first XML tag.

5.17.2.10 FLMELM_TYPXML_ENDELM

```
#define FLMELM_TYPXML_ENDELM 3
```

Closing XML tag element type.

Indicates a closing XML tag. It contains the tag name as payload. In a valid XML document, there must be a matching FLMELM_TYPXML_STARTELM element prior to a closing XML tag element.

5.17.2.11 FLMELM_TYPXML_ENDSTRTELM

```
#define FLMELM_TYPXML_ENDSTRTELM 2
```

End of opening XML tag element type.

Indicates the end of an opening XML tag. It must be preceded by a FLMELM_TYPXML_DATA element. This element carries no payload, i.e. its data length is zero.

5.17.2.12 FLMELM_TYPXML_INTENTDECL

```
#define FLMELM_TYPXML_INTENTDECL 13
```

Internal entity declaration element type.

Entity declarations define general or parameter entities. Entities names are assigned replacement texts that are substitutes for any occurrence of the entity. Parameter entities are only valid within the DTD. General entities are valid in the XML document body. This element only describes internal entities, i.e. entities with values directly defined within the DTD. Here is an example for an internal general and a internal parameter entity declaration:

1. `<!ENTITY name "value">`
2. `<!ENTITY % name "value">`

The element's payload consists of three attributes separated by a NUL character:

```
<name>\0<value>\0<is_parameter_entity>
```

- `name` of the entity
- `value` that is substituted for entities with above name
- `is_parameter_entity` is the character 1 (not the byte value!) if a parameter entity is declared, 0 for a general entity

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL, FLMELM_TYPXML_SKIPENT, FLMELM_TYPXML_SKIPPARMENT

5.17.2.13 FLMELM_TYPXML_NOTDECLP

```
#define FLMELM_TYPXML_NOTDECLP 17
```

Notation declaration element type with a public identifier.

Notation declarations define the format of unparsed external entities (referred to by the optional NDATA portion of an entity declaration), the format of elements which bear a notation attribute and more. This element only describes notation declarations with a public identifier of the form:

```
<!NOTATION name PUBLIC "publicId">
```

The element's payload consists of two attributes separated by a NUL character:

```
<name>\0<public_id>
```

- `name` of the entity
- `public_id` contains the public identifier

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_NOTDECLS, FLMELM_TYPXML_NOTDECLPS, FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL

5.17.2.14 FLMELM_TYPXML_NOTDECLPS

```
#define FLMELM_TYPXML_NOTDECLPS 18
```

Notation declaration element type with a public and system identifier.

Notation declarations define the format of unparsed external entities (referred to by the optional NDATA portion of an entity declaration), the format of elements which bear a notation attribute and more. This element only describes notation declarations with a public and system identifier of the form:

```
<!NOTATION name PUBLIC "publicId" "systemId">
```

The element's payload consists of three attributes separated by a NUL character:

```
<name>\0<public_id>\0<system_id>
```

- `name` of the entity
- `public_id` contains the public identifier
- `system_id` contains the system identifier

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_NOTDECLS, FLMELM_TYPXML_NOTDECLP, FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL

5.17.2.15 FLMELM_TYPXML_NOTDECLS

```
#define FLMELM_TYPXML_NOTDECLS 16
```

Notation declaration element type with a system identifier.

Notation declarations define the format of unparsed external entities (referred to by the optional NDATA portion of an entity declaration), the format of elements which bear a notation attribute and more. This element only describes notation declarations with a system identifier of the form:

```
<!NOTATION name SYSTEM "systemId">
```

The element's payload consists of two attributes separated by a NUL character:

```
<name>\0<system_id>
```

- `name` of the entity
- `system_id` contains the system identifier

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_NOTDECLP, FLMELM_TYPXML_NOTDECLPS, FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL

5.17.2.16 FLMELM_TYPXML_PROCINST

```
#define FLMELM_TYPXML_PROCINST 19
```

Processing instruction element type.

Processing instructions carry instructions to the application. They have a target name and application-specific string. A common example is rendering XML documents using an XSLT stylesheet:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

The element's payload consists of two attributes separated by a NUL character:

```
<target>\0<instruction>
```

- `target` names the intended recipient of an instruction
- `instruction` is an application-specific string

5.17.2.17 FLMELM_TYPXML_PUBENTDECL

```
#define FLMELM_TYPXML_PUBENTDECL 15
```

External entity declaration element type with public identifier.

Entity declarations define general or parameter entities. Entities names are assigned replacement texts that are substitutes for any occurrence of the entity. Parameter entities are only valid within the DTD. General entities are valid in the XML document body. This element only describes external entities with a public identifier, i.e. entities with values defined in another file and a public id. The file's content maybe parsed or unparsed (NDATA) data. Here is an example for external general and internal parameter entity declarations with public identifier:

1. `<!ENTITY name PUBLIC "-//private//file//" "file.txt">`
2. `<!ENTITY name PUBLIC "-//private//file//" "file.txt" NDATA notation>`
3. `<!ENTITY % name PUBLIC "-//private//file//" "file.txt">`
4. `<!ENTITY % name PUBLIC "-//private//file//" "file.txt" NDATA notation>`

The element's payload consists of five attributes separated by a NUL character:

```
<name>\0<public_id>\0<system_id>\0<notation_name>\<is_parameter_entity>
```

- `name` of the entity
- `public_id` contains the public identifier
- `system_id` contains the system identifier, typically a file URI
- `notation_name` is a non-empty string if the external resource is unparsed (NDATA) and must match the name of a notation declaration
- `is_parameter_entity` is the character 1 (not the byte value!) if a parameter entity is declared, 0 for a general entity

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SKIPENT, FLMELM_TYPXML_SKIPPARMENT, FLMELM_TYPXML_NOTDECLS, FLMELM_TYPXML_NOTDECLP, FLMELM_TYPXML_NOTDECLPS

5.17.2.18 FLMELM_TYPXML_SKIPENT

```
#define FLMELM_TYPXML_SKIPENT 7
```

General skipped entity element type.

A general entity is a reference to a replacement text that is associated with a name. It is encoded in the XML document body as `&name;` with `name` being the name of the entity. Its value is defined by an entity declaration in the DTD for the XML document. We do not interpret DTDs. Therefore, these entities are not replaced with the actual text, but reported as skipped entities. The payload of a general skipped entity element is the name of the entity.

See also

FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL

5.17.2.19 FLMELM_TYPXML_SKIPPARMENT

```
#define FLMELM_TYPXML_SKIPPARMENT 8
```

Skipped parameter entity element type.

A parameter entity is a reference to a replacement text that is associated with a name. Unlike general entities, parameter entities are only replaced if used inside the DTD and are encoded as `name;` with `name` being the name of the entity. Its value is defined by a parameter entity declaration in the DTD of the XML document. We do not interpret DTDs. Therefore, these entities are not replaced with the actual text, but reported as skipped parameter entities. The payload of a skipped parameter entity element is the name of the entity.

See also

FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_SYSENTDECL, FLMELM_TYPXML_PUBENTDECL

5.17.2.20 FLMELM_TYPXML_STARTCD

```
#define FLMELM_TYPXML_STARTCD 20
```

Start of CDATA section element type.

A CDATA section contains arbitrary character data which is not parsed by an XML data. So, CDATA sections can contain whole XML documents, which are treated like any other character data. The data in a CDATA section may not contain the character sequence]]> as this marks the end of a CDATA section.

This element carries no payload, i.e. its data length is zero.

Any number of elements of type FLMELM_TYPXML_DATA may follow, followed by a FLMELM_TYPXML_ENDCD element.

See also

FLMELM_TYPXML_ENDCD

5.17.2.21 FLMELM_TYPXML_STARTDTD

```
#define FLMELM_TYPXML_STARTDTD 9
```

Start of a DTD (Document Type Definition) element type.

A document type definition (DTD) defines the structure of an XML document. It can be specified inline within the XML document itself or as a reference to some external resource. An FLMELM_TYPXML_ENDDTD element must follow before the first XML tag occurs to terminate the DTD. The start of a DTD has one of these following forms:

1. `<!DOCTYPE name SYSTEM "sysid"`
2. `<!DOCTYPE name SYSTEM "sysid" [`

3. <!DOCTYPE name PUBLIC "pubid" "sysid"
4. <!DOCTYPE name PUBLIC "pubid" "sysid" [
5. <!DOCTYPE name [

The element's payload consists of five attributes separated by a NUL character:

```
<type>\0<name>\0<public_id>\0<system_id>\0<has_internal_subset>
```

- The `type` flag is one byte that indicates the DTD reference type and has one of three values ** - (case 5, no external reference, DTD is inline, requires `has_internal_subset` set to 1) ** S (case 1+2, the SYSTEM identifier is present) ** P (case 3+4, the PUBLIC identifier is present)
- `name` must be same as the root element in the document
- `public_id` contains the public identifier if type=P
- `system_id` contains the system identifier if type=S or P
- `has_internal_subset` is a single byte field set to the character 1 (not the byte value!) if there is an internal subset (inline DTD) portion. Otherwise, it's set to the character 0.

5.17.2.22 FLMELM_TYPXML_STARTELM

```
#define FLMELM_TYPXML_STARTELM 1
```

Start of opening XML tag element type.

Indicates the start of an opening XML tag. It contains the tag name as payload. This element type is immediately followed by pairs of elements of types FLMELM_TYPXML_ATTNAME and FLMELM_TYPXML_ATTRVAL for each of the tag's attributes (if any). The end of an opening XML tag is indicated by an element of type FLMELM_TYPXML_ENDSTRTELM (mandatory).

5.17.2.23 FLMELM_TYPXML_SYSENTDECL

```
#define FLMELM_TYPXML_SYSENTDECL 14
```

External entity declaration element type with system identifier.

Entity declarations define general or parameter entities. Entities names are assigned replacement texts that are substitutes for any occurrence of the entity. Parameter entities are only valid within the DTD. General entities are valid in the XML document body. This element only describes external entities with a system identifier, i.e. entities with values defined in another file. The file's content maybe parsed or unparsed (NDATA) data. Here is an example for external general and internal parameter entity declarations with system identifier:

1. <!ENTITY name SYSTEM "file.txt">
2. <!ENTITY name SYSTEM "file.txt" NDATA notation>
3. <!ENTITY % name SYSTEM "file.txt">
4. <!ENTITY % name SYSTEM "file.txt" NDATA notation>

The element's payload consists of four attributes separated by a NUL character:

```
<name>\0<system_id>\0<notation_name>\0<is_parameter_entity>
```

- `name` of the entity
- `system_id` contains the system identifier, typically a file URI
- `notation_name` is a non-empty string if the external resource is unparsed (NDATA) and must match the name of a notation declaration
- `is_parameter_entity` is the character 1 (not the byte value!) if a parameter entity is declared, 0 for a general entity

This element type may only occur within a DTD, i.e. between pairs of elements of types FLMELM_TYPXML_STARTDTD and FLMELM_TYPXML_ENDDTD.

See also

FLMELM_TYPXML_INTENTDECL, FLMELM_TYPXML_PUBENTDECL, FLMELM_TYPXML_SKIPENT, FLMELM_TYPXML_SKIPPARMENT, FLMELM_TYPXML_NOTDECLS, FLMELM_TYPXML_NOTDECLP, FLMELM_TYPXML_NOTDECLPS

5.17.2.24 FLMELM_TYPXML_XML

```
#define FLMELM_TYPXML_XML 6
```

XML prolog element type.

Element for the prolog of an XML document, e.g. `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`. The XML prolog is always the first element and must not be preceded by any other element. The prolog is optional, but recommended by the standard. The element's payload are the three attributes separated by a NUL character, each:

```
<version>\0<encoding>\0<standalone>
```

- The `version` is mandatory according to the XML standard.
- The `encoding` string is optional and may be empty. It contains the character encoding of the document.
- The `standalone` field is a one byte field with one of three possible values:
  - - (no standalone field present)
  - Y (standalone="yes")
  - N (standalone="no")

5.18 TAB

Table element types (see: FLMMAT_TYP_TABELM)

Macros

- #define FLMELM_TYPTAB_NONE 0

  No element type.
- #define FLMELM_TYPTAB_FLAG_NULLID 0x00010000

  Flag for null indication.
- #define FLMELM_TYPTAB_FLAG_EMPTID 0x00020000

  Flag for empty indication.
- #define FLMELM_TYPTAB_BINARY 0x00000001

  Binary element type.
- #define FLMELM_TYPTAB_STRING 0x00000002

  String element type.
- #define FLMELM_TYPTAB_INTEGER 0x00000003

  Integer element type.
- #define FLMELM_TYPTAB_FLOAT 0x00000004

  Float element type.
- #define FLMELM_TYPTAB_HEADER 0x0000000A

  Column name element type.

5.18.1 Detailed Description

Table element types (see: FLMMAT_TYP_TABELM)

A table matrix consists of a number of different element types of which each reflects the data type of the column. Additional bits in the second byte are used for different kind of attributes (e.g. tag of an TLV). The attribute length are encoded in the high oder 2 bytes\0.

5.18.2 Macro Definition Documentation

5.18.2.1 FLMELM_TYPTAB_BINARY

```
#define FLMELM_TYPTAB_BINARY 0x00000001
```

Binary element type.

A binary data type is a black box for each kind of data in a certain length. The external representation can be different encodings. Internally representation if always the binary form.

5.18.2.2 FLMELM_TYPTAB_FLAG_EMPTID

```
#define FLMELM_TYPTAB_FLAG_EMPTID 0x00020000
```

Flag for empty indication.

This flag are add to each data type to indicate a default value for an empty data item (e.g. a integer string of length 0 replaced by a default value (e.g. 0)).

5.18.2.3 FLMELM_TYPTAB_FLAG_NULLID

```
#define FLMELM_TYPTAB_FLAG_NULLID 0x00010000
```

Flag for null indication.

This flag are add to each data type to indicate a default value for an optional item that was not part of the data.

5.18.2.4 FLMELM_TYPTAB_FLOAT

```
#define FLMELM_TYPTAB_FLOAT 0x00000004
```

Float element type.

A float data type is a byte array in a certain length. The external representation can be in different encodings (IEEE, BCD, string). The internal representation is a hidden complex data structure (black box).

5.18.2.5 FLMELM_TYPTAB_HEADER

```
#define FLMELM_TYPTAB_HEADER 0x0000000A
```

Column name element type.

This is a special element type to detect a table header in a element list of tables. Each table starts with a CLP string containing the table name and a list of column names. The names can have optional different kind of path variants preceded to support XML and other hierarchical data formats. And the name can be appended optional the DTL for this column in parenthesis.

5.18.2.6 FLMELM_TYPTAB_INTEGER

```
#define FLMELM_TYPTAB_INTEGER 0x00000003
```

Integer element type.

A integer data type is a byte array in a certain length. The external representation can be in different encodings (two's complement, BCD, string). The internal representation is a hidden complex data structure (black box).

5.18.2.7 FLMELM_TYPTAB_NONE

```
#define FLMELM_TYPTAB_NONE 0
```

No element type.

This element type contains elements which are only parsed from a row. There is no data type defined, it is simply a part (blob) from a row. This element type is handled like a binary element at write.

Attention: The net data of table element list are only consists of such elements.

5.18.2.8 FLMELM_TYPTAB_STRING

```
#define FLMELM_TYPTAB_STRING 0x00000002
```

String element type.

A string data type is a character array in a certain length. The external representation can be in different encodings (character sets). The internal representation if always Unicode in UTF-8.

5.19 Structures

Definition of structures for FLAM's APIs.

Data Structures

- struct FlmElmRec0

  FLAM 5 serialized element structure (version 0)

Macros

- #define FLMELMREC0_LENGTH(element)

  Macro that calculates the length of an element of type FlmElmRec0.
- #define FLMELMREC0_ATTRPTR(element)

  Convenience macro that calculates the correct starting address of the attribute data within FlmElmRec0::buffer.
- #define FLMELMREC0_DATAPTR(element)

  Convenience macro that calculates the correct starting address of the element's data payload within FlmElmRec0::buffer.
- #define FLMELMREC0_HASHPTR(element)

  Convenience macro that calculates the correct starting address of the hash data within FlmElmRec0::buffer.

5.19.1 Detailed Description

Definition of structures for FLAM's APIs.

5.19.2 Macro Definition Documentation

5.19.2.1 FLMELMREC0_ATTRPTR

```
#define FLMELMREC0_ATTRPTR(
              element )
```

Convenience macro that calculates the correct starting address of the attribute data within FlmElmRec0::buffer.

5.19.2.2 FLMELMREC0_DATAPTR

```
#define FLMELMREC0_DATAPTR(
              element )
```

Convenience macro that calculates the correct starting address of the element's data payload within FlmElmRec0::buffer.

5.19.2.3 FLMELMREC0_HASHPTR

```
#define FLMELMREC0_HASHPTR(
              element )
```

Convenience macro that calculates the correct starting address of the hash data within FlmElmRec0::buffer.

5.19.2.4 FLMELMREC0_LENGTH

```
#define FLMELMREC0_LENGTH(
              element )
```

Macro that calculates the length of an element of type FlmElmRec0.

This macro calculates the correct length in bytes of a serialized element structure of type FlmElmRec0 by passing a pointer the element.

The macro may be used to conveniently calculate the correct buffer size e.g. to pass to fcbwrite() or FCRPUT() when writing elements.

5.20 Return codes

FLAM Return/Reason codes.

Macros

- #define FLMRTC_OK 0

  Success.
- #define FLMRTC_FAT 1

  Fatal error (not expected)
- #define FLMRTC_SYS 2

  System error (sizeof(U32)!=4)
- #define FLMRTC_HDL 3

  Handle error.
- #define FLMRTC_ITF 4

  Call/parameter/interface error.
- #define FLMRTC_NOT 5

  Action not possible.
- #define FLMRTC_ALC 6

  Memory allocation failed.
- #define FLMRTC_LEN 7

  Length not valid.
- #define FLMRTC_CTR 8

  Control code not valid.
- #define FLMRTC_VSN 9

  Version not supported.
- #define FLMRTC_FUC 10

  Function not supported.
- #define FLMRTC_MOD 11

  Mode not supported.
- #define FLMRTC_SUT 12

  Suite not supported.
- #define FLMRTC_CNF 13

  Confidential mode not supported.
- #define FLMRTC_INT 14

  Integrity mode not supported.
- #define FLMRTC_VRI 15

  Verification mode not supported.
- #define FLMRTC_HCS 16

  Header check sum not valid.
- #define FLMRTC_PCS 17

  Pot check sum not valid.
- #define FLMRTC_DCS 18

  Data check sum not valid.
- #define FLMRTC_CHK 19

  Check sum not valid.
- #define FLMRTC_MAC 20

  MAC not valid.
- #define FLMRTC_KTV 21

KTV not valid.

- #define FLMRTC_SYN 22

  Syntax not valid.
- #define FLMRTC_CNT 23

  Amount not valid.
- #define FLMRTC_EOL 24

  End of list (informational)
- #define FLMRTC_DEL 25

  Delete not successful.
- #define FLMRTC_TYP 26

  Type not valid.
- #define FLMRTC_RBD 27

  Rebuild required (informational)
- #define FLMRTC_NCG 28

  No change happen (informational)
- #define FLMRTC_STA 29

  State not valid.
- #define FLMRTC_ORI 30

  Orientation not supported.
- #define FLMRTC_SPL 31

  Split method not supported.
- #define FLMRTC_DLM 32

  Maximum of records achieved.
- #define FLMRTC_CMP 33

  Error at compression.
- #define FLMRTC_DCO 34

  Error at decompression.
- #define FLMRTC_HLN 35

  Maximal header length achieved.
- #define FLMRTC_DLN 36

  Maximal data length achieved.
- #define FLMRTC_FFO 37

  Failed to open a FLAMFILE.
- #define FLMRTC_RCF 38

  Record format not supported.
- #define FLMRTC_FWR 39

  Write failed.
- #define FLMRTC_FRD 40

  Read failed.
- #define FLMRTC_FGP 41

  Determine position failed.
- #define FLMRTC_FSP 42

  Set position failed.
- #define FLMRTC_EOF 43

  End of file (informational)
- #define FLMRTC_FFR 44

  Rename of FLAMFILE failed.
- #define FLMRTC_FFD 45

  Delete of FLAMFILE failed.
- #define FLMRTC_FFC 46

  Close of FLAMFILE failed.

- #define FLMRTC_FSK 47

  Seek failed.
- #define FLMRTC_FSB 48

  Set buffer size failed.
- #define FLMRTC_FFF 49

  FLAMFILE format not supported.
- #define FLMRTC_NFF 50

  No FLAMFILE.
- #define FLMRTC_FND 51

  Not find (informational)
- #define FLMRTC_EPY 52

  Empty construct.
- #define FLMRTC_CON 53

  Connection error.
- #define FLMRTC_FID 54

  Unsupported FLAM-ID.
- #define FLMRTC_RPC 55

  Remote procedure call failed.
- #define FLMRTC_MTD 56

  Method not supported.
- #define FLMRTC_OFO 57

  Failed to open an original file.
- #define FLMRTC_GZP 58

  GZIP error.
- #define FLMRTC_CHS 59

  Char set not supported.
- #define FLMRTC_AIR 60

  Space too small.
- #define FLMRTC_LOG 61

  Logging facility does not work.
- #define FLMRTC_ICV 62

  Error from character conversion module.
- #define FLMRTC_HFO 63

  Failed to open the hash output file.
- #define FLMRTC_OCL 64

  Close of original file failed.
- #define FLMRTC_KME 65

  Error in FKME module.
- #define FLMRTC_CHR 66

  Character conversion error.
- #define FLMRTC_INF 67

  Info not supported.
- #define FLMRTC_BND 68

  Band not supported.
- #define FLMRTC_RFO 69

  Open report file failed.
- #define FLMRTC_SIZ 70

  Size invalid.
- #define FLMRTC_IVC 71

  Invalid char.
- #define FLMRTC_ICC 72

Incomplete char.

- #define FLMRTC_BOM 73

  Error with byte order mark.
- #define FLMRTC_FMT 74

  Format wrong or not supported.
- #define FLMRTC_ACV 75

  Auto conversion not possible.
- #define FLMRTC_BZ2 76

  BZIP2 error.
- #define FLMRTC_ENV 77

  Error with environment variable.
- #define FLMRTC_DYL 78

  Dynamic load failed.
- #define FLMRTC_ACS 79

  Error from access module.
- #define FLMRTC_LIM 80

  Special condition code, compression limit not reached.
- #define FLMRTC_LXZ 81

  XZ error.
- #define FLMRTC_EXP 82

  Expat error.
- #define FLMRTC_PAD 83

  Whole first block is padding.
- #define FLMRTC_CLP 84

  Command line parser failed.
- #define FLMRTC_MAP 85

  Mapping failed.
- #define FLMRTC_TMP 86

  Failed to open a temporary file.
- #define FLMRTC_MBE 87

  Wrong magic bytes (magic bytes error)
- #define FLMRTC_CFF 88

  Corrupted or no FLAMFILE.
- #define FLMRTC_INI 89

  Initialization error.
- #define FLMRTC_PAR 90

  Parameter not formatted correctly.
- #define FLMRTC_BIN 91

  Binary data detected.
- #define FLMRTC_EQU 92

  From and to are equal.
- #define FLMRTC_ENC 93

  Data are encrypted (key required)
- #define FLMRTC_KEY 94

  Key wrong.
- #define FLMRTC_REA 95

  Reallocation required.
- #define FLMRTC_USR 96

  Invalid user name.
- #define FLMRTC_AUH 97

  Authentication failed.

- #define FLMRTC_HSH 98

  Hash calculation failed.
- #define FLMRTC_DIR 99

  Processing of one or more files failed.
- #define FLMRTC_HFE 100

  Hash file empty.
- #define FLMRTC_HFR 101

  Read of hash/checksum file failed.
- #define FLMRTC_HFN 102

  Hash/checksum file name not valid.
- #define FLMRTC_HFF 103

  Hash/checksum file not correct formated.
- #define FLMRTC_CTO 104

  Failed to open the code table.
- #define FLMRTC_CTF 105

  Code table not correct formated.
- #define FLMRTC_ATR 106

  Attribute settings not supported.
- #define FLMRTC_OPT 107

  Option not supported.
- #define FLMRTC_IPC 108

  Invalid parameter combination.
- #define FLMRTC_IFO 109

  Failed to open the info file.
- #define FLMRTC_PRT 110

  Protocol not supported.
- #define FLMRTC_DIF 111

  Differences found.
- #define FLMRTC_TAS 112

  Special condition code, try again with ASCII.
- #define FLMRTC_INC 113

  Data incomplete.
- #define FLMRTC_PRC 114

  Procedure not supported.
- #define FLMRTC_KNF 115

  Key not found.
- #define FLMRTC_PGP 116

  OpenPGP error.
- #define FLMRTC_ALG 117

  Algorithm not supported.
- #define FLMRTC_KIA 118

  Key inactive.
- #define FLMRTC_KIV 119

  Key invalid.
- #define FLMRTC_KXP 120

  Key expired.
- #define FLMRTC_POS 121

  Positioning failed.
- #define FLMRTC_SEQ 122

  Sequence error, incorrect order.
- #define FLMRTC_UPD 123

Update not possible.

- #define FLMRTC_REG 124

  Regular expression error.
- #define FLMRTC_RNG 125

  Value out of range.
- #define FLMRTC_USG 126

  Usage measurement not possible.
- #define FLMRTC_DMY 127

  Dummy processing requested, but not possible.
- #define FLMRTC_VFO 128

  Failed to open the inverse command file.
- #define FLMRTC_ITN 129

  Internal error (may not happen)
- #define FLMRTC_MFS 130

  Micro Focus support failed.
- #define FLMRTC_STR 131

  Dynamic string library error.
- #define FLMRTC_OUL 132

  Open of usage log failed.
- #define FLMRTC_SER 133

  Serialization failed.
- #define FLMRTC_DSR 134

  De-serialization failed.
- #define FLMRTC_PCR 135

  Program call (PC) failed.
- #define FLMRTC_ROW 136

  Row specification wrong.
- #define FLMRTC_RCD 137

  Record delimiter not supported.
- #define FLMRTC_EOT 138

  End of table (informational)
- #define FLMRTC_EOM 139

  End of member (informational)
- #define FLMRTC_HMD 140

  Have more data (informational)
- #define FLMRTC_ALN 141

  Alignment error.
- #define FLMRTC_SPC 142

  Not enough space.
- #define FLMRTC_PLS 143

  Plausibility error.
- #define FLMRTC_RST 144

  Unexpected remaining rest.
- #define FLMRTC_LNG 145

  Length value too small.
- #define FLMRTC_VRS 146

  Virus found.
- #define FLMRTC_CWC 147

  Connection was closed.
- #define FLMRTC_AVS 148

  Error from anti virus scanner.
- #define FLMRTC_CFO 171

  Failed to open the command file (not used)

5.20.1 Detailed Description

FLAM Return/Reason codes.

This chapter contains all FLAM return codes defined by the FL5 infrastructure. This internal reason code will be a 32 bit integer value containing a decimal number which is related to a certain error situation.

The reason code are useful for program control. Additional the FL5 infrastructure provides functionalities to map such a return code in an error message and to obtain a error trace for more information. It depends of the interface you are using how the mapping function and error trace will be provided.

A FLAM return code provide the main reason for an error or warning. On high level interfaces (subprogram, command line) the internal FLAM return code is provided as reason code and a additional condition code (0, 4, 8, ...) is set by the application or subprogram. Low level interfaces (byte, record and element API) use the values defined in this chapter as return code.

5.20.2 Macro Definition Documentation

5.20.2.1 FLMRTC_ACS

```
#define FLMRTC_ACS 79
```

Error from access module.

5.20.2.2 FLMRTC_ACV

```
#define FLMRTC_ACV 75
```

Auto conversion not possible.

5.20.2.3 FLMRTC_AIR

```
#define FLMRTC_AIR 60
```

Space too small.

5.20.2.4 FLMRTC_ALC

```
#define FLMRTC_ALC 6
```

Memory allocation failed.

5.20.2.5 FLMRTC_ALG

```
#define FLMRTC_ALG 117
```

Algorithm not supported.

5.20.2.6 FLMRTC_ALN

```
#define FLMRTC_ALN 141
```

Alignment error.

5.20.2.7 FLMRTC_ATR

```
#define FLMRTC_ATR 106
```

Attribute settings not supported.

5.20.2.8 FLMRTC_AUH

```
#define FLMRTC_AUH 97
```

Authentication failed.

5.20.2.9 FLMRTC_AVS

```
#define FLMRTC_AVS 148
```

Error from anti virus scanner.

5.20.2.10 FLMRTC_BIN

```
#define FLMRTC_BIN 91
```

Binary data detected.

5.20.2.11 FLMRTC_BND

```
#define FLMRTC_BND 68
```

Band not supported.

5.20.2.12 FLMRTC_BOM

```
#define FLMRTC_BOM 73
```

Error with byte order mark.

5.20.2.13 FLMRTC_BZ2

```
#define FLMRTC_BZ2 76
```

BZIP2 error.

5.20.2.14 FLMRTC_CFF

```
#define FLMRTC_CFF 88
```

Corrupted or no FLAMFILE.

5.20.2.15 FLMRTC_CFO

```
#define FLMRTC_CFO 171
```

Failed to open the command file (not used)

5.20.2.16 FLMRTC_CHK

```
#define FLMRTC_CHK 19
```

Check sum not valid.

5.20.2.17 FLMRTC_CHR

```
#define FLMRTC_CHR 66
```

Character conversion error.

5.20.2.18 FLMRTC_CHS

```
#define FLMRTC_CHS 59
```

Char set not supported.

5.20.2.19 FLMRTC_CLP

```
#define FLMRTC_CLP 84
```

Command line parser failed.

5.20.2.20 FLMRTC_CMP

```
#define FLMRTC_CMP 33
```

Error at compression.

5.20.2.21 FLMRTC_CNF

```
#define FLMRTC_CNF 13
```

Confidential mode not supported.

5.20.2.22 FLMRTC_CNT

```
#define FLMRTC_CNT 23
```

Amount not valid.

5.20.2.23 FLMRTC_CON

```
#define FLMRTC_CON 53
```

Connection error.

5.20.2.24 FLMRTC_CTF

```
#define FLMRTC_CTF 105
```

Code table not correct formated.

5.20.2.25 FLMRTC_CTO

```
#define FLMRTC_CTO 104
```

Failed to open the code table.

5.20.2.26 FLMRTC_CTR

```
#define FLMRTC_CTR 8
```

Control code not valid.

5.20.2.27 FLMRTC_CWC

```
#define FLMRTC_CWC 147
```

Connection was closed.

5.20.2.28 FLMRTC_DCO

```
#define FLMRTC_DCO 34
```

Error at decompression.

5.20.2.29 FLMRTC_DCS

```
#define FLMRTC_DCS 18
```

Data check sum not valid.

5.20.2.30 FLMRTC_DEL

```
#define FLMRTC_DEL 25
```

Delete not successful.

5.20.2.31 FLMRTC_DIF

```
#define FLMRTC_DIF 111
```

Differences found.

5.20.2.32 FLMRTC_DIR

```
#define FLMRTC_DIR 99
```

Processing of one or more files failed.

5.20.2.33 FLMRTC_DLM

```
#define FLMRTC_DLM 32
```

Maximum of records achieved.

5.20.2.34 FLMRTC_DLN

```
#define FLMRTC_DLN 36
```

Maximal data length achieved.

5.20.2.35 FLMRTC_DMY

```
#define FLMRTC_DMY 127
```

Dummy processing requested, but not possible.

5.20.2.36 FLMRTC_DSR

```
#define FLMRTC_DSR 134
```

De-serialization failed.

5.20.2.37 FLMRTC_DYL

```
#define FLMRTC_DYL 78
```

Dynamic load failed.

5.20.2.38 FLMRTC_ENC

```
#define FLMRTC_ENC 93
```

Data are encrypted (key required)

5.20.2.39 FLMRTC_ENV

```
#define FLMRTC_ENV 77
```

Error with environment variable.

5.20.2.40 FLMRTC_EOF

```
#define FLMRTC_EOF 43
```

End of file (informational)

5.20.2.41 FLMRTC_EOL

```
#define FLMRTC_EOL 24
```

End of list (informational)

5.20.2.42 FLMRTC_EOM

```
#define FLMRTC_EOM 139
```

End of member (informational)

5.20.2.43 FLMRTC_EOT

```
#define FLMRTC_EOT 138
```

End of table (informational)

5.20.2.44 FLMRTC_EPY

```
#define FLMRTC_EPY 52
```

Empty construct.

5.20.2.45 FLMRTC_EQU

```
#define FLMRTC_EQU 92
```

From and to are equal.

5.20.2.46 FLMRTC_EXP

```
#define FLMRTC_EXP 82
```

Expat error.

5.20.2.47 FLMRTC_FAT

```
#define FLMRTC_FAT 1
```

Fatal error (not expected)

5.20.2.48 FLMRTC_FFC

```
#define FLMRTC_FFC 46
```

Close of FLAMFILE failed.

5.20.2.49 FLMRTC_FFD

```
#define FLMRTC_FFD 45
```

Delete of FLAMFILE failed.

5.20.2.50 FLMRTC_FFF

```
#define FLMRTC_FFF 49
```

FLAMFILE format not supported.

5.20.2.51 FLMRTC_FFO

```
#define FLMRTC_FFO 37
```

Failed to open a FLAMFILE.

5.20.2.52 FLMRTC_FFR

```
#define FLMRTC_FFR 44
```

Rename of FLAMFILE failed.

5.20.2.53 FLMRTC_FGP

```
#define FLMRTC_FGP 41
```

Determine position failed.

5.20.2.54 FLMRTC_FID

```
#define FLMRTC_FID 54
```

Unsupported FLAM-ID.

5.20.2.55 FLMRTC_FMT

```
#define FLMRTC_FMT 74
```

Format wrong or not supported.

5.20.2.56 FLMRTC_FND

```
#define FLMRTC_FND 51
```

Not find (informational)

5.20.2.57 FLMRTC_FRD

```
#define FLMRTC_FRD 40
```

Read failed.

5.20.2.58 FLMRTC_FSB

```
#define FLMRTC_FSB 48
```

Set buffer size failed.

5.20.2.59 FLMRTC_FSK

```
#define FLMRTC_FSK 47
```

Seek failed.

5.20.2.60 FLMRTC_FSP

```
#define FLMRTC_FSP 42
```

Set position failed.

5.20.2.61 FLMRTC_FUC

```
#define FLMRTC_FUC 10
```

Function not supported.

5.20.2.62 FLMRTC_FWR

```
#define FLMRTC_FWR 39
```

Write failed.

5.20.2.63 FLMRTC_GZP

```
#define FLMRTC_GZP 58
```

GZIP error.

5.20.2.64 FLMRTC_HCS

```
#define FLMRTC_HCS 16
```

Header check sum not valid.

5.20.2.65 FLMRTC_HDL

```
#define FLMRTC_HDL 3
```

Handle error.

5.20.2.66 FLMRTC_HFE

```
#define FLMRTC_HFE 100
```

Hash file empty.

5.20.2.67 FLMRTC_HFF

```
#define FLMRTC_HFF 103
```

Hash/checksum file not correct formated.

5.20.2.68 FLMRTC_HFN

```
#define FLMRTC_HFN 102
```

Hash/checksum file name not valid.

5.20.2.69 FLMRTC_HFO

```
#define FLMRTC_HFO 63
```

Failed to open the hash output file.

5.20.2.70 FLMRTC_HFR

```
#define FLMRTC_HFR 101
```

Read of hash/checksum file failed.

5.20.2.71 FLMRTC_HLN

```
#define FLMRTC_HLN 35
```

Maximal header length achieved.

5.20.2.72 FLMRTC_HMD

```
#define FLMRTC_HMD 140
```

Have more data (informational)

5.20.2.73 FLMRTC_HSH

```
#define FLMRTC_HSH 98
```

Hash calculation failed.

5.20.2.74 FLMRTC_ICC

```
#define FLMRTC_ICC 72
```

Incomplete char.

5.20.2.75 FLMRTC_ICV

```
#define FLMRTC_ICV 62
```

Error from character conversion module.

5.20.2.76 FLMRTC_IFO

```
#define FLMRTC_IFO 109
```

Failed to open the info file.

5.20.2.77 FLMRTC_INC

```
#define FLMRTC_INC 113
```

Data incomplete.

5.20.2.78 FLMRTC_INF

```
#define FLMRTC_INF 67
```

Info not supported.

5.20.2.79 FLMRTC_INI

```
#define FLMRTC_INI 89
```

Initialization error.

5.20.2.80 FLMRTC_INT

```
#define FLMRTC_INT 14
```

Integrity mode not supported.

5.20.2.81 FLMRTC_IPC

```
#define FLMRTC_IPC 108
```

Invalid parameter combination.

5.20.2.82 FLMRTC_ITF

```
#define FLMRTC_ITF 4
```

Call/parameter/interface error.

5.20.2.83 FLMRTC_ITN

```
#define FLMRTC_ITN 129
```

Internal error (may not happen)

5.20.2.84 FLMRTC_IVC

```
#define FLMRTC_IVC 71
```

Invalid char.

5.20.2.85 FLMRTC_KEY

```
#define FLMRTC_KEY 94
```

Key wrong.

5.20.2.86 FLMRTC_KIA

```
#define FLMRTC_KIA 118
```

Key inactive.

5.20.2.87 FLMRTC_KIV

```
#define FLMRTC_KIV 119
```

Key invalid.

5.20.2.88 FLMRTC_KME

```
#define FLMRTC_KME 65
```

Error in FKME module.

5.20.2.89 FLMRTC_KNF

```
#define FLMRTC_KNF 115
```

Key not found.

5.20.2.90 FLMRTC_KTV

```
#define FLMRTC_KTV 21
```

KTV not valid.

5.20.2.91 FLMRTC_KXP

```
#define FLMRTC_KXP 120
```

Key expired.

5.20.2.92 FLMRTC_LEN

```
#define FLMRTC_LEN 7
```

Length not valid.

5.20.2.93 FLMRTC_LIM

```
#define FLMRTC_LIM 80
```

Special condition code, compression limit not reached.

5.20.2.94 FLMRTC_LNG

```
#define FLMRTC_LNG 145
```

Length value too small.

5.20.2.95 FLMRTC_LOG

```
#define FLMRTC_LOG 61
```

Logging facility does not work.

5.20.2.96 FLMRTC_LXZ

```
#define FLMRTC_LXZ 81
```

XZ error.

5.20.2.97 FLMRTC_MAC

```
#define FLMRTC_MAC 20
```

MAC not valid.

5.20.2.98 FLMRTC_MAP

```
#define FLMRTC_MAP 85
```

Mapping failed.

5.20.2.99 FLMRTC_MBE

```
#define FLMRTC_MBE 87
```

Wrong magic bytes (magic bytes error)

5.20.2.100 FLMRTC_MFS

```
#define FLMRTC_MFS 130
```

Micro Focus support failed.

5.20.2.101 FLMRTC_MOD

```
#define FLMRTC_MOD 11
```

Mode not supported.

5.20.2.102 FLMRTC_MTD

```
#define FLMRTC_MTD 56
```

Method not supported.

5.20.2.103 FLMRTC_NCG

```
#define FLMRTC_NCG 28
```

No change happen (informational)

5.20.2.104 FLMRTC_NFF

```
#define FLMRTC_NFF 50
```

No FLAMFILE.

5.20.2.105 FLMRTC_NOT

```
#define FLMRTC_NOT 5
```

Action not possible.

5.20.2.106 FLMRTC_OCL

```
#define FLMRTC_OCL 64
```

Close of original file failed.

5.20.2.107 FLMRTC_OFO

```
#define FLMRTC_OFO 57
```

Failed to open an original file.

5.20.2.108 FLMRTC_OK

```
#define FLMRTC_OK 0
```

Success.

5.20.2.109 FLMRTC_OPT

```
#define FLMRTC_OPT 107
```

Option not supported.

5.20.2.110 FLMRTC_ORI

```
#define FLMRTC_ORI 30
```

Orientation not supported.

5.20.2.111 FLMRTC_OUL

```
#define FLMRTC_OUL 132
```

Open of usage log failed.

5.20.2.112 FLMRTC_PAD

```
#define FLMRTC_PAD 83
```

Whole first block is padding.

5.20.2.113 FLMRTC_PAR

```
#define FLMRTC_PAR 90
```

Parameter not formatted correctly.

5.20.2.114 FLMRTC_PCR

```
#define FLMRTC_PCR 135
```

Program call (PC) failed.

5.20.2.115 FLMRTC_PCS

```
#define FLMRTC_PCS 17
```

Pot check sum not valid.

5.20.2.116 FLMRTC_PGP

```
#define FLMRTC_PGP 116
```

OpenPGP error.

5.20.2.117 FLMRTC_PLS

```
#define FLMRTC_PLS 143
```

Plausibility error.

5.20.2.118 FLMRTC_POS

```
#define FLMRTC_POS 121
```

Positioning failed.

5.20.2.119 FLMRTC_PRC

```
#define FLMRTC_PRC 114
```

Procedure not supported.

5.20.2.120 FLMRTC_PRT

```
#define FLMRTC_PRT 110
```

Protocol not supported.

5.20.2.121 FLMRTC_RBD

```
#define FLMRTC_RBD 27
```

Rebuild required (informational)

5.20.2.122 FLMRTC_RCD

```
#define FLMRTC_RCD 137
```

Record delimiter not supported.

5.20.2.123 FLMRTC_RCF

```
#define FLMRTC_RCF 38
```

Record format not supported.

5.20.2.124 FLMRTC_REA

```
#define FLMRTC_REA 95
```

Reallocation required.

5.20.2.125 FLMRTC_REG

```
#define FLMRTC_REG 124
```

Regular expression error.

5.20.2.126 FLMRTC_RFO

```
#define FLMRTC_RFO 69
```

Open report file failed.

5.20.2.127 FLMRTC_RNG

```
#define FLMRTC_RNG 125
```

Value out of range.

5.20.2.128 FLMRTC_ROW

```
#define FLMRTC_ROW 136
```

Row specification wrong.

5.20.2.129 FLMRTC_RPC

```
#define FLMRTC_RPC 55
```

Remote procedure call failed.

5.20.2.130 FLMRTC_RST

```
#define FLMRTC_RST 144
```

Unexpected remaining rest.

5.20.2.131 FLMRTC_SEQ

```
#define FLMRTC_SEQ 122
```

Sequence error, incorrect order.

5.20.2.132 FLMRTC_SER

```
#define FLMRTC_SER 133
```

Serialization failed.

5.20.2.133 FLMRTC_SIZ

```
#define FLMRTC_SIZ 70
```

Size invalid.

5.20.2.134 FLMRTC_SPC

```
#define FLMRTC_SPC 142
```

Not enough space.

5.20.2.135 FLMRTC_SPL

```
#define FLMRTC_SPL 31
```

Split method not supported.

5.20.2.136 FLMRTC_STA

```
#define FLMRTC_STA 29
```

State not valid.

5.20.2.137 FLMRTC_STR

```
#define FLMRTC_STR 131
```

Dynamic string library error.

5.20.2.138 FLMRTC_SUT

```
#define FLMRTC_SUT 12
```

Suite not supported.

5.20.2.139 FLMRTC_SYN

```
#define FLMRTC_SYN 22
```

Syntax not valid.

5.20.2.140 FLMRTC_SYS

```
#define FLMRTC_SYS 2
```

System error (sizeof(U32)!=4)

5.20.2.141 FLMRTC_TAS

```
#define FLMRTC_TAS 112
```

Special condition code, try again with ASCII.

5.20.2.142 FLMRTC_TMP

```
#define FLMRTC_TMP 86
```

Failed to open a temporary file.

5.20.2.143 FLMRTC_TYP

```
#define FLMRTC_TYP 26
```

Type not valid.

5.20.2.144 FLMRTC_UPD

```
#define FLMRTC_UPD 123
```

Update not possible.

5.20.2.145 FLMRTC_USG

```
#define FLMRTC_USG 126
```

Usage measurement not possible.

5.20.2.146 FLMRTC_USR

```
#define FLMRTC_USR 96
```

Invalid user name.

5.20.2.147 FLMRTC_VFO

```
#define FLMRTC_VFO 128
```

Failed to open the inverse command file.

5.20.2.148 FLMRTC_VRI

```
#define FLMRTC_VRI 15
```

Verification mode not supported.

5.20.2.149 FLMRTC_VRS

```
#define FLMRTC_VRS 146
```

Virus found.

5.20.2.150 FLMRTC_VSN

```
#define FLMRTC_VSN 9
```

Version not supported.

Chapter 6

Data Structure Documentation

6.1 FcpeCls Struct Reference

FCPE Close structure.

`#include <FCPE.h>`

Data Fields

- unsigned int uiMsg

  [OUTPUT] Length of error message (32 bit word)
- unsigned int uiDy1

  [DUMMY] 32 bit dummy field for 64 bit alignment
- char acMsg [1024]

  [OUTPUT] Maximal 1024 byte long error string

6.1.1 Detailed Description

FCPE Close structure.

This structure is used when the exit function is called with the function code FCPE_FUNC_CLS. It is always called after data processing is finished, no matter if there was error or not.

The only member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

6.1.2 Field Documentation

6.1.2.1 acMsg

```
char FcpeCls::acMsg[1024]
```

[OUTPUT] Maximal 1024 byte long error string

6.1.2.2 uiDy1

```
unsigned int FcpeCls::uiDy1
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.1.2.3 uiMsg

```
unsigned int FcpeCls::uiMsg
```

[OUTPUT] Length of error message (32 bit word)

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

- FCPE.h

6.2 FcpeOpn Struct Reference

FCPE parameter structure for function code FCPE_FUNC_OPN.

```
#include <FCPE.h>
```

Data Fields

- unsigned int uiFlg
  [INOUT] Flag word (32 bit) with several bit settings
- unsigned int uiCcs
  [INOUT] CCSID in first low order 24 bit and the and combined character form in the high order 8 bit
- unsigned int uiTab
  [INPUT] Length of the table name (32 bit word)
- unsigned int uiDy1
  [DUMMY] 32 bit dummy field for 64 bit alignment
- char * pcTab
  [INPUT] Pointer to the table name (32 or 64 bit)
- unsigned int uiCol
  [INPUT] Length of the column name (32 bit word)
- unsigned int uiDy2
  [DUMMY] 32 bit dummy field for 64 bit alignment

- char * pcCol

  [INPUT] Pointer to the column name (32 or 64 bit)
- unsigned int uiPar

  [INPUT] Length of the parameter string (32 bit word)
- unsigned int uiDy3

  [DUMMY] 32 bit dummy field for 64 bit alignment
- char * pcPar

  [INPUT] Pointer to the parameter string (32 or 64 bit)
- unsigned int uiIvr

  [OUTPUT] Length of the inverse parameter string (32 bit word)
- unsigned int uiDy4

  [DUMMY] 32 bit dummy field for 64 bit alignment
- char * pcIvr

  [OUTPUT] Pointer to the inverse parameter string (32 or 64 bit)
- unsigned int uiMsg

  [OUTPUT] Length of error message (32 bit word)
- unsigned int uiDy5

  [DUMMY] 32 bit dummy field for 64 bit alignment
- char acMsg [1024]

  [OUTPUT] Maximal 1024 byte long error string

6.2.1 Detailed Description

FCPE parameter structure for function code FCPE_FUNC_OPN.

This structure is used when the exit function is called with the function code FCPE_FUNC_OPN.

The first member contains flags that can be set in both directions. If the exit is opened for converting output data, the flag bit FCPE_FLAG_WRITE is set. If it is not set, the exit is opened for converting input data. The exit function may set the FCPE_FLAG_NOCPY flag if it does not write to the output buffer in the TsFcpeRun structure (i.e. it only validates or changes the data in-place).

The second member contains the CCSID in the low order 24 bits. The high order 8 bits contain the combined character form. The member is set by the exit driver to a value corresponding to the input data and must be changed by the exit function only if its output is in a different character set. A CCSID of 0 indicates binary data. For example, if the exit function implements encryption of strings, the input text may be in UTF-8 (CCSID=1208). Since the encryption produces binary output, the CCSID must be set to 0.

The next three member pairs contain the length and a pointer to the corresponding string with the table name, the column name and the parameter string. The parameter string must be interpreted by the service provider.

The following member pair may be set by the exit function if the operation is invertible. The exit function must the pointer to the inverse parameter string and is responsible for allocating and freeing memory for this string. If the length is zero or the pointer is NULL, the exit is considered not invertible.

The last member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

6.2.2 Field Documentation

6.2.2.1 acMsg

```
char FcpeOpn::acMsg[1024]
```

[OUTPUT] Maximal 1024 byte long error string

6.2.2.2 pcCol

```
char* FcpeOpn::pcCol
```

[INPUT] Pointer to the column name (32 or 64 bit)

6.2.2.3 pcIvr

```
char* FcpeOpn::pcIvr
```

[OUTPUT] Pointer to the inverse parameter string (32 or 64 bit)

6.2.2.4 pcPar

```
char* FcpeOpn::pcPar
```

[INPUT] Pointer to the parameter string (32 or 64 bit)

6.2.2.5 pcTab

```
char* FcpeOpn::pcTab
```

[INPUT] Pointer to the table name (32 or 64 bit)

6.2.2.6 uiCcs

```
unsigned int FcpeOpn::uiCcs
```

[INOUT] CCSID in first low order 24 bit and the and combined character form in the high order 8 bit

6.2.2.7 uiCol

```
unsigned int FcpeOpn::uiCol
```

[INPUT] Length of the column name (32 bit word)

6.2.2.8 uiDy1

```
unsigned int FcpeOpn::uiDy1
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.2.2.9 uiDy2

```
unsigned int FcpeOpn::uiDy2
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.2.2.10 uiDy3

```
unsigned int FcpeOpn::uiDy3
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.2.2.11 uiDy4

```
unsigned int FcpeOpn::uiDy4
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.2.2.12 uiDy5

```
unsigned int FcpeOpn::uiDy5
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.2.2.13 uiFlg

```
unsigned int FcpeOpn::uiFlg
```

[INOUT] Flag word (32 bit) with several bit settings

6.2.2.14 uiIvr

```
unsigned int FcpeOpn::uiIvr
```

[OUTPUT] Length of the inverse parameter string (32 bit word)

6.2.2.15 uiMsg

```
unsigned int FcpeOpn::uiMsg
```

[OUTPUT] Length of error message (32 bit word)

6.2.2.16 uiPar

```
unsigned int FcpeOpn::uiPar
```

[INPUT] Length of the parameter string (32 bit word)

6.2.2.17 uiTab

```
unsigned int FcpeOpn::uiTab
```

[INPUT] Length of the table name (32 bit word)

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

- FCPE.h

6.3 FcpePar Union Reference

FCPE Union of function structures.

```
#include <FCPE.h>
```

Data Fields

- TsFcpeOpn stOpn

  [INOUT] Structure for function code FCPE_FUNC_OPN
- TsFcpeRun stRun

  [INOUT] Structure for function code FCPE_FUNC_RUN
- TsFcpeCls stCls

  [INOUT] Structure for function code FCPE_FUNC_CLS

6.3.1 Detailed Description

FCPE Union of function structures.

This union must be used to access structure fields depending on the function code that was passed to exit function. Only the member that corresponds to the function code may be accessed. Failing to do so results in undefined behavior!

6.3.2 Field Documentation

6.3.2.1 stCls

```
TsFcpeCls FcpePar::stCls
```

[INOUT] Structure for function code FCPE_FUNC_CLS

6.3.2.2 stOpn

```
TsFcpeOpn FcpePar::stOpn
```

[INOUT] Structure for function code FCPE_FUNC_OPN

6.3.2.3 stRun

```
TsFcpeRun FcpePar::stRun
```

[INOUT] Structure for function code FCPE_FUNC_RUN

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

- FCPE.h

6.4 FcpeRun Struct Reference

FCPE Run structure.

`#include <FCPE.h>`

Data Fields

- unsigned int uiInp
  [INPUT] Length of the input data (32 bit word)
- unsigned int uiDy1
  [DUMMY] 32 bit dummy field for 64 bit alignment
- unsigned char * pcInp
  [INPUT] Pointer to the input data (32 or 64 bit)
- unsigned int uiOut
  [INOUT] Output space at input and output length at output (32 bit word)
- unsigned int uiDy2
  [DUMMY] 32 bit dummy field for 64 bit alignment
- unsigned char * pcOut
  [OUTPUT] Pointer to the output data (32 or 64 bit)
- unsigned int uiFlg
  [INOUT] Flag word for null, empty and other indications (32 bit word) (output at read, input at write)
- unsigned int uiMsg
  [OUTPUT] Length of error message (32 bit word)
- char acMsg [1024]
  [OUTPUT] Maximal 1024 byte long error string

6.4.1 Detailed Description

FCPE Run structure.

This structure is used when the exit function is called with the function code FCPE_FUNC_RUN. It is called for each data item of this column, allowing the exit to manipulate the data as needed.

The exit drivers passes the length and a pointer to the input data of the current column as the first pair of members of the structure.

The second member pair is the size and a pointer to an already allocated buffer where the exit function may write its output to and set the output length accordingly. The input and output buffers are managed by the exit driver. If the output buffer is too small, then FLMRTC_LEN must be set as return code and the output length must contain the minimum required buffer length. The exit driver then repeats the call with a buffer of at least the desired size. This is repeated until the exit function no longer sets the FLMRTC_LEN error code. If the FCPE_FLAG_NOCPY flag is set, the output pointer and length is set to NULL/0 by the exit driver and the exit function must set the output length to the input length and the output pointer to the input pointer for plausibility checks by the exit driver.

If the FCPE_FLAG_NOCPY flag has been set during the FCPE_FUNC_RUN function code call, the input buffer may be modified in-place. However, the output pointer must be set to the input pointer. This prevents unnessary copy operation if the exit function only perform minor changes or none at all.

The flag word can be used to set indication flags bits when reading and to use these flags when writing. Currently, the bit definitions below are defined:

FLMELM_TYPTAB_FLAG_NULLID FLMELM_TYPTAB_FLAG_EMPTID

At normal pre- or post-processing the flag word is not touched.

The last member pair is a 1024 bytes buffer for an error message. The corresponding length must be set to the length of the error message.

6.4.2 Field Documentation

6.4.2.1 acMsg

```
char FcpeRun::acMsg[1024]
```

[OUTPUT] Maximal 1024 byte long error string

6.4.2.2 pcInp

```
unsigned char* FcpeRun::pcInp
```

[INPUT] Pointer to the input data (32 or 64 bit)

6.4.2.3 pcOut

```
unsigned char* FcpeRun::pcOut
```

[OUTPUT] Pointer to the output data (32 or 64 bit)

6.4.2.4 uiDy1

```
unsigned int FcpeRun::uiDy1
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.4.2.5 uiDy2

```
unsigned int FcpeRun::uiDy2
```

[DUMMY] 32 bit dummy field for 64 bit alignment

6.4.2.6 uiFlg

```
unsigned int FcpeRun::uiFlg
```

[INOUT] Flag word for null, empty and other indications (32 bit word) (output at read, input at write)

6.4.2.7 uiInp

```
unsigned int FcpeRun::uiInp
```

[INPUT] Length of the input data (32 bit word)

6.4.2.8 uiMsg

```
unsigned int FcpeRun::uiMsg
```

[OUTPUT] Length of error message (32 bit word)

6.4.2.9 uiOut

```
unsigned int FcpeRun::uiOut
```

[INOUT] Output space at input and output length at output (32 bit word)

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

- FCPE.h

6.5 FlmElmRec0 Struct Reference

FLAM 5 serialized element structure (version 0)

```
#include <FLMDEF.h>
```

Data Fields

- int version

  Element structure version (must be 0)
- int matTyp

  Matrix type (type of data format)
- int elmTyp

  Element type.
- int atrLen

  Attribute data length (at offset 32)
- int datLen

  Element data length (at offset 32 + atrLen)
- int hshLen

  Hash data length (at offset 32 + atrLen + datLen)
- int reserved1
- int reserved2
- char buffer [4]

  Unsigned data buffer (variable length)

6.5.1 Detailed Description

FLAM 5 serialized element structure (version 0)

This structure defines a FLAM 5 element in a serialized form. It may be used to cast a buffer that contains data written by `format.element()`. A serialized element basically consists of a set of 32 bit integers followed by element attributes, data and a data hash (is applicable).

Users MUST check that the version field matches the version of the used struct definition. In future versions, further structures with different fields may be added which will be assigned a new version. The first 32 bits can always be used to identify the correct structure.

Each FLAM 5 matrix can consist of different element types. The interpretation of the different element types depends on the matrix type.

See also

Matrix types, Element types

6.5.2 Field Documentation

6.5.2.1 atrLen

```
int FlmElmRec0::atrLen
```

Attribute data length (at offset 32)

6.5.2.2 buffer

```
char FlmElmRec0::buffer[4]
```

Unsigned data buffer (variable length)

Contains `atrLen` bytes of attribute data, followed by `datLen` bytes of element data, followed by `hshLen` bytes of hash data.

The presence of attribute and hash data when reading depends on the parameter given to format.element(). When writing, attributes and hash data is only passed to FLAM if this is requested in the format.element() clause.

Hash values are only relevant with the FLAM element interface in order to find data elements in compressed and encrypted data. The FLUC element interface (used for data formats like PGP, GZIP, ...) does not support hash values.

6.5.2.3 datLen

```
int FlmElmRec0::datLen
```

Element data length (at offset 32 + atrLen)

6.5.2.4 elmTyp

```
int FlmElmRec0::elmTyp
```

Element type.

The interpretation of the different element types depends on the matrix type. The available element types can be found here.

6.5.2.5 hshLen

```
int FlmElmRec0::hshLen
```

Hash data length (at offset 32 + atrLen + datLen)

6.5.2.6 matTyp

```
int FlmElmRec0::matTyp
```

Matrix type (type of data format)

The matrix type indicates the type of data and defines the set of valid element types that each FLAM 5 matrix can consist of. The list of available matrix types can be found here.

6.5.2.7 reserved1

```
int FlmElmRec0::reserved1
```

Reserved for future use (should be 0)

6.5.2.8 reserved2

```
int FlmElmRec0::reserved2
```

Reserved for future use (should be 0)

6.5.2.9 version

```
int FlmElmRec0::version
```

Element structure version (must be 0)

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

- FLMDEF.h

Chapter 7

File Documentation

7.1 FCPE.h File Reference

FCPE - FLAM column processor exit.

```
#include "FLMRTC.h"
#include "FLMDEF.h"
```

Data Structures

- struct FcpeOpn

  FCPE parameter structure for function code FCPE_FUNC_OPN.
- struct FcpeRun

  FCPE Run structure.
- struct FcpeCls

  FCPE Close structure.
- union FcpePar

  FCPE Union of function structures.

Macros

- #define FCPE_FUNC_OPN 1

  Function code for opening (used in the first call of the exit function)
- #define FCPE_FUNC_RUN 2

  Function code for processing data.
- #define FCPE_FUNC_CLS 3

  Function code for closing (used in the last call of the exit function)
- #define FCPE_DEFLIB_NAME "libfcpe"

  Default library name.
- #define FCPE_DEFFUC_NAME "FLAMCPE"

  Default function name.
- #define FCPE_CMBFRM_NON 0U

  No combined character form.
- #define FCPE_CMBFRM_NFD 1U

Normalization Form Canonical Decomposition.

- #define FCPE_CMBFRM_NFC 2U

  Normalization Form Canonical Composition.

- #define FCPE_FLAG_WRITE 0x00000001U

  [INPUT] If set by exit driver, this signals that the exit is called as post-processing on the writing side

- #define FCPE_FLAG_NOCPY 0x00000002U

  [OUTPUT] Must be set by the exit function if it does not write data to the output buffer (validation or in-place manipulation of input)

Typedefs

- typedef struct FcpeOpn TsFcpeOpn

  FCPE parameter structure for function code FCPE_FUNC_OPN.

- typedef struct FcpeRun TsFcpeRun

  FCPE Run structure.

- typedef struct FcpeCls TsFcpeCls

  FCPE Close structure.

- typedef union FcpePar TuFcpePar

  FCPE Union of function structures.

- typedef void(∗ TpfFcpExit) (void ∗∗ppHdl, int ∗piRetco, const int ∗piFunc, TuFcpePar ∗puPara)

  FCPE function.

7.1.1 Detailed Description

FCPE - FLAM column processor exit.

Author

limes datentechnik gmbh

Date

07.08.2018

Copyright

(c) 2018 limes datentechnik gmbh

7.2 FLMDEF.h File Reference

FLMDEF - External FLAM definitions.

Data Structures

- struct FlmElmRec0

  FLAM 5 serialized element structure (version 0)

Macros

- #define FLMOUT_FORMAT_NON 0

  Default format (FLMOUT_FORMAT_LST)
- #define FLMOUT_FORMAT_LST 1

  Simple list of lines with each line separated by '\n' and the whole string terminated by 0x00.
- #define FLMOUT_FORMAT_XML 2

  As XML-formatted, null-terminated string (in local charset) with each line separated by '\n'.
- #define FLC_READ_FILE 1

  File definition strings in read mode.
- #define FLC_READ_FORMAT 2

  Format data strings in read mode.
- #define FLC_WRITE_FILE 3

  File definition strings in write mode.
- #define FLC_WRITE_FORMAT 4

  Format data strings in write mode.
- #define FLC_CONV_READ 5

  Data conversion strings in read mode.
- #define FLC_CONV_WRITE 6

  Data conversion strings in write mode.
- #define FLC_CONV_FROM_TO 7

  Data conversion string in from-to mode.
- #define FLC_INPUT_FILE 8

  File definition strings in input mode.
- #define FLC_OUTPUT_FILE 9

  File definition strings in output mode.
- #define FLC_INFO 10

  Definition of the info string.
- #define FLC_STATE 11

  Definition of the state string.
- #define FLC_LOG 12

  Definition of the memory log string.
- #define FLCHSH_ALGO_NON 0U

  No valid hash method.
- #define FLCHSH_ALGO_MD5 1U

  MD5 method.
- #define FLCHSH_ALGO_RMD128 2U

  RipeMd-128 method.
- #define FLCHSH_ALGO_RMD160 3U

  RipeMd-160 method.
- #define FLCHSH_ALGO_SHA1 10U

  SHA1 method.
- #define FLCHSH_ALGO_SHA160 10U

  SHA1 method.
- #define FLCHSH_ALGO_SHA224 20U

  SHA224 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA256 21U

  SHA256 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA384 22U

  SHA384 (SHA2 variant) method.
- #define FLCHSH_ALGO_SHA512 23U

SHA512 (SHA2 variant) method.

- #define FLCHSH_ALGO_CRC08 100U

  8 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC16 101U

  16 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC24 102U

  24 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC32 103U

  32 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC32C 104U

  32 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC40 105U

  40 Bit CRC checksum (no crypto quality)
- #define FLCHSH_ALGO_CRC64 106U

  64 Bit CRC checksum (no crypto quality)
- #define FLCMAC_ALGO_NON 0U

  No valid hash method.
- #define FLCMAC_ALGO_HMAC 1U

  HMAC conform to RFC 2104.
- #define FLC_INBOUND_COUNT 1

  Use input values.
- #define FLC_OUTBOUND_COUNT 2

  Use output values.
- #define FLC_FIO_COUNT 1

  Use values from File I/O.
- #define FLC_FMT_COUNT 2

  Use values from formatting.
- #define FLC_BYTE_COUNT 1

  Determine byte counts.
- #define FLC_UNIT_COUNT 2

  Determine unit (block or record) counts.
- #define FLMMAT_TYP_NON 0x00

  Unknown/undefined matrix type.
- #define FLMMAT_TYP_DATBLK 0x80

  Data blocks, 1 element type, no attributes.
- #define FLMMAT_TYP_STDREC 0x81

  Standard record, 1 element type, no attributes.
- #define FLMMAT_TYP_STDASAREC 0x82

  Standard record, 1 element type, with 1 byte ASA control character as attribute.
- #define FLMMAT_TYP_STDMCCREC 0x83

  Standard record, 1 element type, with 1 byte machine control character as attribute.
- #define FLMMAT_TYP_RELREC 0x84

  Relative record, 1 element type, with 8 byte integer (slot number) as attribute.
- #define FLMMAT_TYP_RELASAREC 0x85

  Relative record, 1 element type, with 8 byte slot number and 1 byte ASA as attribute.
- #define FLMMAT_TYP_RELMCCREC 0x86

  Relative record, 1 element type, with 8 byte slot number and 1 byte MCC as attribute.
- #define FLMMAT_TYP_TXTREC 0x87

  Text record, 1 element type, no attributes (like standard record, but text is known)
- #define FLMMAT_TYP_TXTDLM 0x88

  Text record with delimiter at the end, 1 element type, no attributes.

- #define FLMMAT_TYP_TXTRST 0x90

  Text record and rest element (suppressed trailing whitespace and original delimiter), 2 elements, no attributes.
- #define FLMMAT_TYP_XMLELM 0x91

  XML elements, no attributes.
- #define FLMMAT_TYP_TABELM 0x92

  Table elements, no attributes.
- #define FLMELM_NOSKIP -1

  Use this value for an element type if no skip of an element type required at read.
- #define FLMELM_TYPBLK_STANDARD 0

  Simple data block element type (may contain binary data)
- #define FLMELM_TYPREC_STANDARD 0

  Record element type for record matrix types.
- #define FLMELM_TYPTXT_RECORD 0

  Text element type of a TXTRST matrix.
- #define FLMELM_TYPTXT_REST 1

  Rest element type of a TXTRST matrix.
- #define FLMELM_TYPXML_DATA 0

  XML data element type.
- #define FLMELM_TYPXML_STARTELM 1

  Start of opening XML tag element type.
- #define FLMELM_TYPXML_ENDSTRTELM 2

  End of opening XML tag element type.
- #define FLMELM_TYPXML_ENDELM 3

  Closing XML tag element type.
- #define FLMELM_TYPXML_ATTNAME 4

  Attribute name element type.
- #define FLMELM_TYPXML_ATTRVAL 5

  Attribute value element type.
- #define FLMELM_TYPXML_XML 6

  XML prolog element type.
- #define FLMELM_TYPXML_SKIPENT 7

  General skipped entity element type.
- #define FLMELM_TYPXML_SKIPPARMENT 8

  Skipped parameter entity element type.
- #define FLMELM_TYPXML_STARTDTD 9

  Start of a DTD (Document Type Definition) element type.
- #define FLMELM_TYPXML_ENDDTD 10

  End of a DTD element type.
- #define FLMELM_TYPXML_ELMDECL 11

  DTD element type declaration element type.
- #define FLMELM_TYPXML_ATTLDECL 12

  DTD attribute list declaration element type.
- #define FLMELM_TYPXML_INTENTDECL 13

  Internal entity declaration element type.
- #define FLMELM_TYPXML_SYSENTDECL 14

  External entity declaration element type with system identifier.
- #define FLMELM_TYPXML_PUBENTDECL 15

  External entity declaration element type with public identifier.
- #define FLMELM_TYPXML_NOTDECLS 16

  Notation declaration element type with a system identifier.
- #define FLMELM_TYPXML_NOTDECLP 17

Notation declaration element type with a public identifier.

- #define FLMELM_TYPXML_NOTDECLPS 18

  Notation declaration element type with a public and system identifier.
- #define FLMELM_TYPXML_PROCINST 19

  Processing instruction element type.
- #define FLMELM_TYPXML_STARTCD 20

  Start of CDATA section element type.
- #define FLMELM_TYPXML_ENDCD 21

  Start of CDATA section element type.
- #define FLMELM_TYPXML_COMMENT 22

  Comment element type.
- #define FLMELM_TYPXML_DEFAULT 23

  Element type for all other data in a document.
- #define FLMELM_TYPTAB_NONE 0

  No element type.
- #define FLMELM_TYPTAB_FLAG_NULLID 0x00010000

  Flag for null indication.
- #define FLMELM_TYPTAB_FLAG_EMPTID 0x00020000

  Flag for empty indication.
- #define FLMELM_TYPTAB_BINARY 0x00000001

  Binary element type.
- #define FLMELM_TYPTAB_BINARY_NI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_BINARY_EI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_EMPTID)
- #define FLMELM_TYPTAB_BINARY_EINI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_STRING 0x00000002

  String element type.
- #define FLMELM_TYPTAB_STRING_NI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_STRING_EI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_EMPTID)
- #define FLMELM_TYPTAB_STRING_EINI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_INTEGER 0x00000003

  Integer element type.
- #define FLMELM_TYPTAB_INTEGER_NI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_INTEGER_EI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_EMPTID)
- #define FLMELM_TYPTAB_INTEGER_EINI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_FLOAT 0x00000004

  Float element type.
- #define FLMELM_TYPTAB_FLOAT_NI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_FLOAT_EI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_EMPTID)
- #define FLMELM_TYPTAB_FLOAT_EINI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID)
- #define FLMELM_TYPTAB_HEADER 0x0000000A

  Column name element type.
- #define FLMELMREC0_LENGTH(element)

  Macro that calculates the length of an element of type FlmElmRec0.
- #define FLMELMREC0_ATTRPTR(element)

  Convenience macro that calculates the correct starting address of the attribute data within FlmElmRec0::buffer.
- #define FLMELMREC0_DATAPTR(element)

  Convenience macro that calculates the correct starting address of the element's data payload within FlmElmRec0::buffer.
- #define FLMELMREC0_HASHPTR(element)

  Convenience macro that calculates the correct starting address of the hash data within FlmElmRec0::buffer.

7.2.1 Detailed Description

FLMDEF - External FLAM definitions.

Author

limes datentechnik gmbh

Date

11.01.2016

Copyright

This header contains different global defines for FLAM application programming interfaces.

7.2.2 Macro Definition Documentation

7.2.2.1 FLMELM_TYPTAB_BINARY_EI

```
#define FLMELM_TYPTAB_BINARY_EI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_EMPTID)
```

7.2.2.2 FLMELM_TYPTAB_BINARY_EINI

```
#define FLMELM_TYPTAB_BINARY_EINI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID
```

7.2.2.3 FLMELM_TYPTAB_BINARY_NI

```
#define FLMELM_TYPTAB_BINARY_NI (FLMELM_TYPTAB_BINARY | FLMELM_TYPTAB_FLAG_NULLID)
```

7.2.2.4 FLMELM_TYPTAB_FLOAT_EI

```
#define FLMELM_TYPTAB_FLOAT_EI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_EMPTID)
```

7.2.2.5 FLMELM_TYPTAB_FLOAT_EINI

```
#define FLMELM_TYPTAB_FLOAT_EINI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID)
```

7.2.2.6 FLMELM_TYPTAB_FLOAT_NI

```
#define FLMELM_TYPTAB_FLOAT_NI (FLMELM_TYPTAB_FLOAT | FLMELM_TYPTAB_FLAG_NULLID)
```

7.2.2.7 FLMELM_TYPTAB_INTEGER_EI

```
#define FLMELM_TYPTAB_INTEGER_EI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_EMPTID)
```

7.2.2.8 FLMELM_TYPTAB_INTEGER_EINI

```
#define FLMELM_TYPTAB_INTEGER_EINI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLI
```

7.2.2.9 FLMELM_TYPTAB_INTEGER_NI

```
#define FLMELM_TYPTAB_INTEGER_NI (FLMELM_TYPTAB_INTEGER | FLMELM_TYPTAB_FLAG_NULLID)
```

7.2.2.10 FLMELM_TYPTAB_STRING_EI

```
#define FLMELM_TYPTAB_STRING_EI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_EMPTID)
```

7.2.2.11 FLMELM_TYPTAB_STRING_EINI

```
#define FLMELM_TYPTAB_STRING_EINI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_EMPTID | FLMELM_TYPTAB_FLAG_NULLID
```

7.2.2.12 FLMELM_TYPTAB_STRING_NI

```
#define FLMELM_TYPTAB_STRING_NI (FLMELM_TYPTAB_STRING | FLMELM_TYPTAB_FLAG_NULLID)
```

7.3 FLMRTC.h File Reference

FLMRTC - FLAM-Return-Codes.

Macros

- #define FLMRTC_OK 0
  Success.
- #define FLMRTC_FAT 1
  Fatal error (not expected)
- #define FLMRTC_SYS 2
  System error (sizeof(U32)!=4)
- #define FLMRTC_HDL 3
  Handle error.
- #define FLMRTC_ITF 4
  Call/parameter/interface error.
- #define FLMRTC_NOT 5
  Action not possible.
- #define FLMRTC_ALC 6
  Memory allocation failed.
- #define FLMRTC_LEN 7
  Length not valid.
- #define FLMRTC_CTR 8
  Control code not valid.
- #define FLMRTC_VSN 9
  Version not supported.
- #define FLMRTC_FUC 10
  Function not supported.
- #define FLMRTC_MOD 11
  Mode not supported.
- #define FLMRTC_SUT 12
  Suite not supported.
- #define FLMRTC_CNF 13
  Confidential mode not supported.
- #define FLMRTC_INT 14
  Integrity mode not supported.
- #define FLMRTC_VRI 15
  Verification mode not supported.
- #define FLMRTC_HCS 16
  Header check sum not valid.
- #define FLMRTC_PCS 17
  Pot check sum not valid.
- #define FLMRTC_DCS 18
  Data check sum not valid.
- #define FLMRTC_CHK 19
  Check sum not valid.
- #define FLMRTC_MAC 20
  MAC not valid.
- #define FLMRTC_KTV 21

KTV not valid.

- #define FLMRTC_SYN 22

  Syntax not valid.
- #define FLMRTC_CNT 23

  Amount not valid.
- #define FLMRTC_EOL 24

  End of list (informational)
- #define FLMRTC_DEL 25

  Delete not successful.
- #define FLMRTC_TYP 26

  Type not valid.
- #define FLMRTC_RBD 27

  Rebuild required (informational)
- #define FLMRTC_NCG 28

  No change happen (informational)
- #define FLMRTC_STA 29

  State not valid.
- #define FLMRTC_ORI 30

  Orientation not supported.
- #define FLMRTC_SPL 31

  Split method not supported.
- #define FLMRTC_DLM 32

  Maximum of records achieved.
- #define FLMRTC_CMP 33

  Error at compression.
- #define FLMRTC_DCO 34

  Error at decompression.
- #define FLMRTC_HLN 35

  Maximal header length achieved.
- #define FLMRTC_DLN 36

  Maximal data length achieved.
- #define FLMRTC_FFO 37

  Failed to open a FLAMFILE.
- #define FLMRTC_RCF 38

  Record format not supported.
- #define FLMRTC_FWR 39

  Write failed.
- #define FLMRTC_FRD 40

  Read failed.
- #define FLMRTC_FGP 41

  Determine position failed.
- #define FLMRTC_FSP 42

  Set position failed.
- #define FLMRTC_EOF 43

  End of file (informational)
- #define FLMRTC_FFR 44

  Rename of FLAMFILE failed.
- #define FLMRTC_FFD 45

  Delete of FLAMFILE failed.
- #define FLMRTC_FFC 46

  Close of FLAMFILE failed.

- #define FLMRTC_FSK 47
  Seek failed.
- #define FLMRTC_FSB 48
  Set buffer size failed.
- #define FLMRTC_FFF 49
  FLAMFILE format not supported.
- #define FLMRTC_NFF 50
  No FLAMFILE.
- #define FLMRTC_FND 51
  Not find (informational)
- #define FLMRTC_EPY 52
  Empty construct.
- #define FLMRTC_CON 53
  Connection error.
- #define FLMRTC_FID 54
  Unsupported FLAM-ID.
- #define FLMRTC_RPC 55
  Remote procedure call failed.
- #define FLMRTC_MTD 56
  Method not supported.
- #define FLMRTC_OFO 57
  Failed to open an original file.
- #define FLMRTC_GZP 58
  GZIP error.
- #define FLMRTC_CHS 59
  Char set not supported.
- #define FLMRTC_AIR 60
  Space too small.
- #define FLMRTC_LOG 61
  Logging facility does not work.
- #define FLMRTC_ICV 62
  Error from character conversion module.
- #define FLMRTC_HFO 63
  Failed to open the hash output file.
- #define FLMRTC_OCL 64
  Close of original file failed.
- #define FLMRTC_KME 65
  Error in FKME module.
- #define FLMRTC_CHR 66
  Character conversion error.
- #define FLMRTC_INF 67
  Info not supported.
- #define FLMRTC_BND 68
  Band not supported.
- #define FLMRTC_RFO 69
  Open report file failed.
- #define FLMRTC_SIZ 70
  Size invalid.
- #define FLMRTC_IVC 71
  Invalid char.
- #define FLMRTC_ICC 72

Incomplete char.

- #define FLMRTC_BOM 73

  Error with byte order mark.
- #define FLMRTC_FMT 74

  Format wrong or not supported.
- #define FLMRTC_ACV 75

  Auto conversion not possible.
- #define FLMRTC_BZ2 76

  BZIP2 error.
- #define FLMRTC_ENV 77

  Error with environment variable.
- #define FLMRTC_DYL 78

  Dynamic load failed.
- #define FLMRTC_ACS 79

  Error from access module.
- #define FLMRTC_LIM 80

  Special condition code, compression limit not reached.
- #define FLMRTC_LXZ 81

  XZ error.
- #define FLMRTC_EXP 82

  Expat error.
- #define FLMRTC_PAD 83

  Whole first block is padding.
- #define FLMRTC_CLP 84

  Command line parser failed.
- #define FLMRTC_MAP 85

  Mapping failed.
- #define FLMRTC_TMP 86

  Failed to open a temporary file.
- #define FLMRTC_MBE 87

  Wrong magic bytes (magic bytes error)
- #define FLMRTC_CFF 88

  Corrupted or no FLAMFILE.
- #define FLMRTC_INI 89

  Initialization error.
- #define FLMRTC_PAR 90

  Parameter not formatted correctly.
- #define FLMRTC_BIN 91

  Binary data detected.
- #define FLMRTC_EQU 92

  From and to are equal.
- #define FLMRTC_ENC 93

  Data are encrypted (key required)
- #define FLMRTC_KEY 94

  Key wrong.
- #define FLMRTC_REA 95

  Reallocation required.
- #define FLMRTC_USR 96

  Invalid user name.
- #define FLMRTC_AUH 97

  Authentication failed.

- #define FLMRTC_HSH 98

  Hash calculation failed.
- #define FLMRTC_DIR 99

  Processing of one or more files failed.
- #define FLMRTC_HFE 100

  Hash file empty.
- #define FLMRTC_HFR 101

  Read of hash/checksum file failed.
- #define FLMRTC_HFN 102

  Hash/checksum file name not valid.
- #define FLMRTC_HFF 103

  Hash/checksum file not correct formated.
- #define FLMRTC_CTO 104

  Failed to open the code table.
- #define FLMRTC_CTF 105

  Code table not correct formated.
- #define FLMRTC_ATR 106

  Attribute settings not supported.
- #define FLMRTC_OPT 107

  Option not supported.
- #define FLMRTC_IPC 108

  Invalid parameter combination.
- #define FLMRTC_IFO 109

  Failed to open the info file.
- #define FLMRTC_PRT 110

  Protocol not supported.
- #define FLMRTC_DIF 111

  Differences found.
- #define FLMRTC_TAS 112

  Special condition code, try again with ASCII.
- #define FLMRTC_INC 113

  Data incomplete.
- #define FLMRTC_PRC 114

  Procedure not supported.
- #define FLMRTC_KNF 115

  Key not found.
- #define FLMRTC_PGP 116

  OpenPGP error.
- #define FLMRTC_ALG 117

  Algorithm not supported.
- #define FLMRTC_KIA 118

  Key inactive.
- #define FLMRTC_KIV 119

  Key invalid.
- #define FLMRTC_KXP 120

  Key expired.
- #define FLMRTC_POS 121

  Positioning failed.
- #define FLMRTC_SEQ 122

  Sequence error, incorrect order.
- #define FLMRTC_UPD 123

Update not possible.

- #define FLMRTC_REG 124

  Regular expression error.
- #define FLMRTC_RNG 125

  Value out of range.
- #define FLMRTC_USG 126

  Usage measurement not possible.
- #define FLMRTC_DMY 127

  Dummy processing requested, but not possible.
- #define FLMRTC_VFO 128

  Failed to open the inverse command file.
- #define FLMRTC_ITN 129

  Internal error (may not happen)
- #define FLMRTC_MFS 130

  Micro Focus support failed.
- #define FLMRTC_STR 131

  Dynamic string library error.
- #define FLMRTC_OUL 132

  Open of usage log failed.
- #define FLMRTC_SER 133

  Serialization failed.
- #define FLMRTC_DSR 134

  De-serialization failed.
- #define FLMRTC_PCR 135

  Program call (PC) failed.
- #define FLMRTC_ROW 136

  Row specification wrong.
- #define FLMRTC_RCD 137

  Record delimiter not supported.
- #define FLMRTC_EOT 138

  End of table (informational)
- #define FLMRTC_EOM 139

  End of member (informational)
- #define FLMRTC_HMD 140

  Have more data (informational)
- #define FLMRTC_ALN 141

  Alignment error.
- #define FLMRTC_SPC 142

  Not enough space.
- #define FLMRTC_PLS 143

  Plausibility error.
- #define FLMRTC_RST 144

  Unexpected remaining rest.
- #define FLMRTC_LNG 145

  Length value too small.
- #define FLMRTC_VRS 146

  Virus found.
- #define FLMRTC_CWC 147

  Connection was closed.
- #define FLMRTC_AVS 148

  Error from anti virus scanner.
- #define FLMRTC_CFO 171

  Failed to open the command file (not used)

7.3.1 Detailed Description

FLMRTC - FLAM-Return-Codes.

Author

limes datentechnik gmbh

Date

14.01.2016

Index