FLICONV-API

1

Generated by Doxygen 1.8.14

Contents

Chapter 1

FLUC ICONV Interface

This module provides a *libiconv*-compatible interface for memory to memory character conversion. All special feature of FLUC character conversion module are provided through the TO and FROM string specification in fliconv_open() function.

Example for C:

```
iconvlist(NULL,NULL);
h=iconv_open("UTF16LE//BOM","1141//ELF2NL//IGNORE//TRANSLIT//REPORT(report.txt)");
r=iconv(h,&inDat,&inLen,&outDat,&outLen);
iconv_close(h);
```

Example in Cobol: (Compile with the dll-option on mainframes)

```
CALL 'fliconv_list'  USING OMITTED, OMITTED.
CALL 'fliconv_open'  USING FLICV-TO, FLICV-FROM
                     RETURNING FLICV-HDL.
CALL  'fliconv'      USING BY VALUE  FLICV-HDL,
                           BY REFERENCE INDAT-PTR,
                           BY REFERENCE INLEN,
                           BY REFERENCE OUTDAT-PTR,
                           BY REFERENCE OUTLEN
                     RETURNING FLICV-RET.
CALL 'fliconv_close' USING BY VALUE  FLICV-HDL.
```

Please have a look at the FLICONV.c sample file which implements the linux like ICONV utility (fliconv) based on this library or the COBOL sample SOFLCICV for mainframe systems.

1.1 CCSID's, encoding strings and defines

The open function supports encoding strings and CCSIDS. The fliconv_list() function provides all supported CCS↩ IDs, CHARSETs and the corresponding encoding strings.

1.2 Compatibility mode

If the compiler switch FICONV is defined, then all ∗ *fliconv*∗ entries are also available as *iconv*∗ functions. I.e. to port your existing code you must simply replace:

```
#include<iconv.h>
```

by

```
#define  FICONV
#include"FLCICV.h"
```

With this replacement you can re-build your source. To use the special features you must append one or more of the encoding string specifications (see fliconv_open()).

1.3 Improvements compared to standard iconv implementations:

- Support of encoding strings and CCSIDs, list supported CCSIDs and encoding strings
- EBCDIC New Line (0x15) to Line Feed (0x0A) management
- Subset support (String.Latin, SEPA, ...) and custom user tables
- Recursive mapping and transliteration ('U:'->'UE')
- Case mapping, comprehensive reporting, byte order change handling
- Reads 5 and 6 byte UTF-8 encoded characters (values with preceding zeros)
- Slightly faster and less memory and CPU consumption

1.4 Differences to the standard iconv library:

- Supports more errno values (mainly for user table parsing, not for conversion itself)
- E2BIG is set if the output buffer is smaller than the input data multiplied by the provided expansion factor –> No data is converted if the output buffer is too small (increases performance significantly)
- Functions to provide about, version, license, statistic, error and other information
- Add a few errno-functions to manage errno in other programming languages (COBOL/PLI)
- Byte order mark is only printed to the output file if the BOM keyword is specified in the TO string
- The list function gets available encoding strings, the CCSID and the corresponding CHARSET (UTF/ASCI↩ I/EBCDIC) information

1.5 Sample programs

A sample program in C with name FLICONV can be found as part of the installation package for mainframe systems in the library SRCLIBC(FLICONV), with the corresponding compile and link step in JOBLIB(SBUILD). For other platforms (Windows, UNIX) the sample program source of FLICONV is located in the 'sample' directory and the compile and link procedures can be found in the Makefile of the same directory.

This sample program implements the Linux like 'iconv' utility with all features of FLAM character conversion module.

1.6 Hints for z/OS

On z/OS you must define the language level with EXTC99, as _POSIX_SOURCE and use of long names to compile this sample.

DEFINE(_POSIX_SOURCE),LANGLVL(EXTC99),LO

Chapter 2

Module Index

2.1 API definitions

Here is a list of the API parts:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 Error codes

Macros

- #define FLICONV_EINVAL 1
- #define FLICONV_ENOMEM 2
- #define FLICONV_EILSEQ 3
- #define FLICONV_E2BIG 4
- #define FLICONV_REOPEN 5
- #define FLICONV_BOMCHG 6
- #define FLICONV_UTFAIL 7
- #define FLICONV_MAXEXP 8
- #define FLICONV_LICVIL 9

4.1.1 Detailed Description

Use this predefined values in function ∗fliconv_chkerrno().

4.1.2 Macro Definition Documentation

4.1.2.1 FLICONV_EINVAL

```
#define FLICONV_EINVAL 1
```

Incomplete character

4.1.2.2 FLICONV_ENOMEM

```
#define FLICONV_ENOMEM 2
```

Not enought memory

4.1.2.3 FLICONV_EILSEQ

```
#define FLICONV_EILSEQ 3
```

Invalid character

4.1.2.4 FLICONV_E2BIG

```
#define FLICONV_E2BIG 4
```

Need more space

4.1.2.5 FLICONV_REOPEN

```
#define FLICONV_REOPEN 5
```

Reopen failed (internal error)

4.1.2.6 FLICONV_BOMCHG

```
#define FLICONV_BOMCHG 6
```

Byte order change

4.1.2.7 FLICONV_UTFAIL

```
#define FLICONV_UTFAIL 7
```

Parse of user table failed

4.1.2.8 FLICONV_MAXEXP

```
#define FLICONV_MAXEXP 8
```

Maximal expansion reached

4.1.2.9 FLICONV_LICVIL

```
#define FLICONV_LICVIL 9
```

License violation

4.2 Miscellaneous stuff

Macros

- #define FLICONV_LISTBYTES 8192

  Internal buffer sizes for listings.

Typedefs

- typedef int(* TpfDoOneList) (unsigned int, const char *const *, void *)

  Typedef of function pointer for do_one call back function.

4.2.1 Detailed Description

4.2.2 Macro Definition Documentation

4.2.2.1 FLICONV_LISTBYTES

```
#define FLICONV_LISTBYTES 8192
```

Internal buffer sizes for listings.

Use this value to allocate the data buffer written to by the default formatting function.

Example for C:

```
char data[FLICONV_LISTBYTES]
fliconv_list(NULL,data)
```

4.2.3 Typedef Documentation

4.2.3.1 TpfDoOneList

```
typedef int(* TpfDoOneList) (unsigned int, const char *const *, void *)
```

Typedef of function pointer for *do_one* call back function.

A call back function that is used in function fliconv_list() to format the output of the supported CCSIDs, CHARSETs and encoding strings.

Parameters

in	*namescount*	3 + amount of aliases for this CCSID
in	*names*	array of pointer to the CCSID, CHARSET, encoding string and it's aliases
in,out	*data*	pointer to the data buffer (handle for the call back function)

Returns

return 0 if success anything else for an error

4.3 Functions

Functions

- const char * fliconv_version (void)
  Retrieves version information.
- const char * fliconv_about (void)
  Retrieves about information.
- const char * fliconv_license (void)
  Retrieves the license text.
- void fliconv_list (TpfDoOneList do_one, void *data)
- void * fliconv_open (const char *pcTo_Code, const char *pcFrmCode)
  Open character conversion module.
- int fliconv_close (void *hdl)
  Close character conversion module.
- size_t fliconv (void *cd, char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft)
  Convert a data block.
- void fliconv_seterrno (const int err)
  Set error number.
- int fliconv_geterrno (void)
  Get error number.
- int fliconv_chkerrno (const int err, const int val)
  Check error number.
- const char * fliconv_strerror (int err)
  Get error message.
- const char * fliconv_error_trace (void)
  Get error trace.
- int fliconv_expansion (void *cd)
  Get expansion factor.
- int64_t fliconv_position (void *cd)
  Get position.

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 fliconv_version()

```
const char* fliconv_version (
            void  )
```

Retrieves version information.

Returns a string with version information for each component used. This should be used in a support case.

Example for C:

```
printf("Version: %s\n",fliconv_version();
```

Example in Cobol:

```
CALL  'fliconv_version' RETURNING RET-PTR.
SET ADDRESS OF MSGAREA  TO RET-PTR.
UNSTRING MSGAREA  DELIMITED BY X'00'
                  INTO DISPAREA COUNT MSGLEN.
DISPLAY 'Version Message:'.
DISPLAY  DISPAREA(1:MSGLEN).
```

Returns

a pointer to a static area containing the zero-terminated version string

4.3.2.2 fliconv_about()

```
const char* fliconv_about (
            void  )
```

Retrieves about information.

Returns a string with about information for this library on multiple lines and license information if external libraries are used.

Example for C:

```
printf("About:\n%s\n",fliconv_about();
```

Example in Cobol:

```
CALL  'fliconv_about' RETURNING RET-PTR.
SET ADDRESS OF MSGAREA  TO RET-PTR.
UNSTRING MSGAREA  DELIMITED BY X'00'
                  INTO DISPAREA COUNT MSGLEN.
DISPLAY 'About Message:'.
DISPLAY  DISPAREA(1:MSGLEN).
```

Returns

a pointer to a static area containing the zero-terminated about string

4.3.2.3 fliconv_license()

```
const char* fliconv_license (
            void  )
```

Retrieves the license text.

This function can be used to get the current license text on multiple lines. The license text defines the permissible use of this library.

Example for C:

```
printf("License:\n%s\n",fliconv_license());
```

Example in Cobol:

```
    CALL  'fliconv_license' RETURNING RET-PTR.
    SET ADDRESS OF MSGAREA  TO RET-PTR.
DISP-MSGS.
    MOVE ZERO  TO MSGLEN.
    UNSTRING MSGAREA  DELIMITED BY X'15' OR X'00'
                      INTO DISPAREA
                      DELIMITER DISP-DLIM COUNT MSGLEN
                      WITH POINTER DISP-PTR.
    IF  DISP-DLIM = X'15'
       THEN
          DISPLAY DISPAREA(1:MSGLEN)
          GO TO DISP-MSGS
    END-IF.
```

Returns

a pointer to a static area with a zero-terminated string containing the current license text.

4.3.2.4 fliconv_list()

```
void fliconv_list (
            TpfDoOneList do_one,
            void * data )
```

This function corresponds to the *iconvlist()* function of *libiconv* and can be used to get a list of supported CCSIDs, CHARSETs and encoding strings. The formatting can be defined by passing a function pointer of type TpfDoOne↩ List. If this function pointer is NULL, a default formatting function is used. This default formatting function writes to STDERR if the data pointer is also NULL. If the data pointer is non-NULL and the function pointer is NULL, the data pointer must point to a buffer that is at least FLICONV_LISTBYTES bytes long. The default formatting function writes the list this buffer. If the buffer is shorter than FLICONV_LISTBYTES bytes, behavior is undefined (risk of segmentation fault).

For example:

- `fliconv_list(NULL,NULL)` writes the CCSID list to stderr

- `fliconv_list(NULL,data)` writes the CCSID list to the data buffer

When specifying a custom callback function (`fliconv_list(myone,mydata)`) the data pointer is passed to the function on each call. In contrast to other *fliconv_list()* implementations, the *namescount* is at least 3 and the index has the meaning below:

- 0 CCSID
- 1 CHARSET (not valid as input for fliconv_open())
- 2 Main encoding string
- >2 Alias encoding strings (currently not supported)

The default formatting function writes lines (delimiter='
') of the following format to STDERR or the specified buffer. If using the buffer version, the resulting string is null-terminated.

```
'CCSID' CHRSET (encoding string)
```

The character sets below are currently supported:

- ASCII all single byte ASCII code pages (CP1252)
- EBCDIC all single byte EBCDIC code pages (IBM1141)
- UTF8 for UTF-8
- UTF16BE for UTF-16BE
- UTF16LE for UTF-16LE
- UTF32BE for UTF-32BE
- UTF32LE for UTF-32LE
- UCS1 for UCS-1
- UCS2BE for UCS-2BE
- UCS2LE for UCS-2LE
- UCS4BE for UCS-4BE
- UCS4LE for UCS-4LE

NOTE: UCS(*)- subset of UTF(*) with first 65536 codepoints, mostly enough for usual applications

The function may set the errno values below:

- EINVAL error in *do_one* function

Below you can find an example of a *do_one* function:

```
static int oneFwrite(unsigned int         namescount,
                     const char* const*   names,
                     void*                data){
   int i;
   if(namescount<3){ return -1; }


   fprintf((FILE*)data,"'%s' %7s (%s",names[0],names[1],names[2]);
   for(i=3;i<namescount;i++){
      fprintf((FILE*)data,"/%s",names[i]);
   }
   fprintf((FILE*)data,")\n");


   return 0;
}
```

Example in Cobol:

```
     CALL 'fliconv_list' USING OMITTED,
                               DISPAREA.
     DISPLAY 'Supported CCSIDs:'.
 DISP-CCSIDS.
     MOVE ZERO  TO MSGLEN.
     UNSTRING DISPAREA DELIMITED BY X'15' OR X'00'
                       INTO DISP-CCSID
                       DELIMITER DISP-DLIM COUNT MSGLEN
                       WITH POINTER DISP-PTR.
     IF  DISP-DLIM = X'15'
       THEN
          DISPLAY DISP-CCSID(1:MSGLEN)
          GO TO DISP-CCSIDS
     END-IF.
```

Parameters

in	*do_one*	NULL for default formatting or function pointer to an own formatting function
in	*data*	Pointer passed to the callback function specified by *do_one*

4.3.2.5 fliconv_open()

```
void* fliconv_open (
            const char * pcTo_Code,
            const char * pcFrmCode )
```

Open character conversion module.

Opens the character conversion module. A target and a source encoding string is required. The target encoding string defines to which character set the data is converted. The source encoding string defines from which character set the input data is converted.

Both strings can be enhanced to use the features below:

Input encoding string enhancements:

- //BOM Manage byte order change
- //ENL2LF Convert EBCDIC new line (0x15) to line feed (0x0A) Output encoding string enhancements:
- //BOM Write byte order mark
- //ELF2NL Convert EBCDIC line feed (0x0A) to new line (0x15)
- //TOUPPER Upper case mapping
- //TOLOWER Lower case mapping
- //TOSUPPER Special upper case mapping
- //TOSLOWER Special lower case mapping
- //TOFOLD Special case folding
- //TOUSER User module/table defined case mapping
- //IGNORE Ignore invalid characters
- //TRANSLIT['('[systab]')'] Transliterate invalid characters [ICONV]
- //SUBSTITUTE['('[codepoint_list]')'] Substitute invalid characters [0x1A]
- //USRMOD'('module_name')' Use a predefined user table module
- //USRTAB'('file_name')' Use a custom user table text file
- //REPORT['('[file_name]')'] Write a report file [STDERR]
- //NFD Normalisation Form D (Canonical Decomposition)
- //NFC Normalisation Form D (Canonical Decomposition, followed by Canonical Composition)
- //COMBINED Character conversion with combined character support

Currently supported system transliteration tables:

- ICONV Transliteration table of libiconv

Currently supported user table module names:

- CCUTNPAS UCS subset for String-Latin (XOEV/NPA, with best fit mapping and case folding)
- CCUTSPEA UCS subset for SEPA (all valid UTF-8 character $<$ 128, with transliteration)
- CCUTDELA UCS subset of IBM1141, CP1252 and ISO8859-15 (only CP check)
- CCUTDLAX UCS Subset of IBM1141, CP1252, ISO8859-15 and XOEF (only CP check)

For more information about custom user table text files please refer to the FLCL user manual.

Example for C:

```
h=fliconv_open("UTF16LE//BOM","1141//ELF2NL//IGNORE//TRANSLIT//REPORT(report.txt)");
if (h==NULL) return(fliconv_geterrno());
```

Example in Cobol:

```
CALL 'fliconv_open'  USING FLICV-TO, FLICV-FROM
                     RETURNING FLICV-HDL.
IF  FLICV-HDL = ZERO THEN error-handling ...
```

Example for error handling:

```
CALL 'fliconv_geterrno' RETURNING FLICV-ERRNO.
CALL 'fliconv_strerror' USING BY VALUE FLICV-ERRNO
                        RETURNING RET-PTR.
SET ADDRESS OF MSGAREA  TO RET-PTR.
UNSTRING MSGAREA  DELIMITED BY X'00'
                  INTO DISPAREA COUNT MSGLEN.
DISPLAY 'Error Message:'.
DISPLAY  DISPAREA(1:MSGLEN).
```

The function may set the errno values below:

- EINVAL parameter wrong or not supported
- ENOMEM allocation failed, not enough space left
- UTFAIL loading the user table failed
- MAXEXP maximum expansion reached
- LICVIL license violation

You can use fliconv_strerror() to get a corresponding error message and fliconv_error_trace() to get more error information.

Parameters

in	*pcTo_Code*	Target encoding string
in	*pcFrmCode*	Source encoding string

Returns

a pointer to a handle to manage character conversion, must be passed to subsequent functions, NULL in case of an error.

4.3.2.6 fliconv_close()

```
int fliconv_close (
            void * hdl )
```

Close character conversion module.

The module must be closed through this function after conversion is finished. All allocated resources are released.

You can use fliconv_strerror() to get an corresponding error message and fliconv_error_trace() to get more error information.

Example for C:

```
fliconv_close(h);
```

Example in Cobol:

```
CALL 'fliconv_close' USING BY VALUE FLICV-HDL.
```

Parameters

in	*hdl*	Handle from fliconv_open()

Returns

Upon successful completion 0 is returned. Otherwise, -1 is returned.

4.3.2.7 fliconv()

```
size_t fliconv (
             void * cd,
             char ** inbuf,
             size_t * inbytesleft,
             char ** outbuf,
             size_t * outbytesleft )
```

Convert a data block.

Converts at most ∗*inbytesleft* bytes starting at ∗*inbuf*, writing at most ∗*outbytesleft* bytes starting at ∗*outbuf*. The function decrements ∗*inbytesleft* and increments ∗*inbuf* by the same amount. On the other side, it also decrements ∗*outbytesleft* and increments ∗*outbuf* by the same amount.

If ∗*inbytesleft* multiplied by the maximum expansion factor is greater than ∗*outbytesleft*, no data is converted, errno is set to E2BIG and -1 is returned. You have to reallocate ∗*outbuf* in order to provide enough space for conversion.

If ∗*inbytesleft* is not 0, an incomplete multi-byte character might have been found at the end of the input data block (return code = -1, errno = EINVAL). If the input block was the last block to be processed, the input data is incomplete. Otherwise, the rest (pointed to by ∗*inbuf* must be at the beginning of the next input block.

To save memory for Unicode character sets the pre-calculated tables are first limit to to 64k entries. If a multibyte character encountered which requires more than 16 bit, then a reopen is done to pre-calculate bigger (1.1M) tables. If not enought memory available the reopen can fail.

The function may set the errno values below:

- EILSEQ invalid byte sequence
- EINVAL incomplete byte sequence
- REOPEN reopen failed
- BOMCHG byte order changed
- E2BIG out buffer smaller than input_length∗expansion

- E2BIG after reopen, expansion factor changed
- MAXEXP in reopen, max expansion reached

You can use fliconv_strerror() to get a corresponding error message and fliconv_error_trace() to get more error information.

Example for C:

```
r=iconv(h,&inDat,&inLen,&outDat,&outLen);
```

Example in Cobol:

```
CALL  'fliconv'      USING BY VALUE  FLICV-HDL,
                           BY REFERENCE INDAT-PTR,
                           BY REFERENCE INLEN,
                           BY REFERENCE OUTDAT-PTR,
                           BY REFERENCE OUTLEN
                     RETURNING FLICV-RET.
```

Parameters

in	*cd*	Handle from fliconv_open()
in,out	*inbuf*	Pointer to the input data buffer
in,out	*inbytesleft*	Input data length
in,out	*outbuf*	Pointer to the output data buffer
in,out	*outbytesleft*	Output buffer size

Returns

The function returns the number of characters converted in a non-reversible way (ignored, substituted or transliterated) during this call; reversible conversions are not counted. If the return code $>$ 0, then entries in the report file can be found. A positive return code is a warning that an internal error handling was done. In case of error, it sets errno and returns (size_t) -1.

4.3.2.8 fliconv_seterrno()

```
void fliconv_seterrno (
            const int err )
```

Set error number.

Use this function in non C programming languages to set the global variable *errno* of the c runtime library. For error handling you must set *errno* in front of a call to 0, to check after the call the value.

Example for C:

```
fliconv_seterrno(0);
r=fliconv(...);
if (r==-1) {
   if (fliconv_chkerrno(fliconv_geterrno(),FLICONV_EINVAL)) {
    ...
   }
}
```

Example in Cobol:

```
CALL  'fliconv_seterrno'   USING BY VALUE ERRNO.
```

Parameters

in	*err*	Error number (mainly only 0 make sense)

4.3.2.9 fliconv_geterrno()

```
int fliconv_geterrno (
            void  )
```

Get error number.

Use this function in non C programming languages to get the global variable *errno* of the c runtime library. For error handling you need access to the *errno* after a call. Attention: This integer contains platform dependent values. You can use the function *fliconv_chkerrno()* to verify the *errno* against the predefined platform independent values above.

Example for C:

```
if (fliconv_chkerrno(fliconv_geterrno(),FLICONV_E2BIG)) ...
```

Example in Cobol:

```
CALL  'fliconv_geterrno' RETURNING FLICV-ERRNO.
```

Returns

Error number

4.3.2.10 fliconv_chkerrno()

```
int fliconv_chkerrno (
            const int err,
            const int val )
```

Check error number.

Use this function in non C programming languages to check the global variable *errno* of the C runtime environment. The value *err* is provided by function *fliconv_geterrno()* and the value *val* is one of the predefined constants above.

Example for C:

```
if (fliconv_chkerrno(fliconv_geterrno(),FLICONV_E2BIG)) ...
```

Parameters

err	Error number
val	Error value (see defined values above)

Returns

1 if match, 0 don't match or unknown

4.3.2.11 fliconv_strerror()

```
const char* fliconv_strerror (
            int err )
```

Get error message.

This function can be used to retrieve a human-readable error message using the errno set by *fliconv* functions.

Example for C:

```
printf("Error message: %s\n",fliconv_strerror(fliconv_geterrno());
```

Example in Cobol:

```
CALL  'fliconv_strerror'  USING BY VALUE FLICV-ERRNO,
                          RETURNING RET-PTR.
SET ADDRESS OF MSGAREA  TO RET-PTR.
UNSTRING MSGAREA  DELIMITED BY X'00'
                  INTO DISPAREA COUNT MSGLEN.
DISPLAY 'Error Message:'.
DISPLAY  DISPAREA(1:MSGLEN).
```

Parameters

in	*err*	A valid error code

Returns

A pointer to a static area with a zero-terminated string containing a FLAM error message matching the passed error code as one line. (no LF)

4.3.2.12 fliconv_error_trace()

```
const char* fliconv_error_trace (
            void  )
```

Get error trace.

This function can be used to get an error trace after a call to fliconv_open(), fliconv() or fliconv_close(). The error trace contains the FLAM error stack.

Example for C:

```
printf("Error message: %s\n",fliconv_strerror(fliconv_geterrno());
printf("Error trace:\n%s\n" ,fliconv_error_trace());
```

Returns

a pointer to a static area with a zero-terminated string containing the current FLAM error trace on several lines

4.3.2.13 fliconv_expansion()

```
int fliconv_expansion (
            void * cd )
```

Get expansion factor.

Returns the maximum expansion factor. This should be used to allocate enough memory for the output buffer. You must pass a valid descriptor obtained from fliconv_open(). This function must be used after fliconv_open() and if you get *errno==E2BIG*.

Example for C:

```
outlen=inlen*fliconv_expansion(h);
outbuf=(unsigned char*)realloc(outbuf,outlen);
```

Example in Cobol:

```
CALL 'fliconv_expansion' USING BY VALUE FLICV-HDL
                         RETURNING BUFFEXP.
DISPLAY 'Expansion factor: ' BUFFEXP.
```

Parameters

in	*cd*	Conversion descriptor

Returns

Expansion factor (0 in case of an error)

4.3.2.14 fliconv_position()

```
int64_t fliconv_position (
            void * cd )
```

Get position.

Return the current position as byte offset. This is mainly for error handling or statistic (before close). The position is the amount of processed bytes. To point to the wrong byte in the case of an error an addition of 1 is required.

Example for C:

```
printf("Error at byte %"PRIi64"\n",fliconv_position()+1);
printf("Statistic: Processed %"PRIi64" bytes\n",fliconv_position());
```

Parameters

in	*cd*	Conversion descriptor

Returns

Current position or 0 if no position available

Chapter 5

File Documentation

5.1 FLCICV.h File Reference

Specification for the FLUC iconv interface.

Macros

- #define FLICONV_EINVAL 1
- #define FLICONV_ENOMEM 2
- #define FLICONV_EILSEQ 3
- #define FLICONV_E2BIG 4
- #define FLICONV_REOPEN 5
- #define FLICONV_BOMCHG 6
- #define FLICONV_UTFAIL 7
- #define FLICONV_MAXEXP 8
- #define FLICONV_LICVIL 9
- #define FLICONV_LISTBYTES 8192

  Internal buffer sizes for listings.

Typedefs

- typedef int(∗ TpfDoOneList) (unsigned int, const char ∗const ∗, void ∗)

  Typedef of function pointer for do_one call back function.

Functions

- const char ∗ fliconv_version (void)

  Retrieves version information.
- const char ∗ fliconv_about (void)

  Retrieves about information.
- const char ∗ fliconv_license (void)

  Retrieves the license text.
- void fliconv_list (TpfDoOneList do_one, void ∗data)
- void ∗ fliconv_open (const char ∗pcTo_Code, const char ∗pcFrmCode)

Open character conversion module.

- int fliconv_close (void *hdl)

  Close character conversion module.

- size_t fliconv (void *cd, char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft)

  Convert a data block.

- void fliconv_seterrno (const int err)

  Set error number.

- int fliconv_geterrno (void)

  Get error number.

- int fliconv_chkerrno (const int err, const int val)

  Check error number.

- const char * fliconv_strerror (int err)

  Get error message.

- const char * fliconv_error_trace (void)

  Get error trace.

- int fliconv_expansion (void *cd)

  Get expansion factor.

- int64_t fliconv_position (void *cd)

  Get position.

5.1.1 Detailed Description

Specification for the FLUC iconv interface.

Author

limes datentechnik gmbh

Date

21.11.2014

Copyright

Index