Reference manual

FICO

FICO® Xpress Optimization

XPRNLS command tool and library

Reference manual

Release 1.0

Last update December 2015

XPRNLS

Deliverable Version: A

Last Revised: December 2015

Version 1.0

Contents

CHAPTER 1

Introduction

The *Xpress Natural Language Support* (XPRNLS) component comes as a command line tool and a library.

- The command line tool supports various commands for transcoding text files (such as converting a text file from one encoding to another encoding) and managing message catalogs (for message translations).
- The library offers a system-independent set of routines for converting text buffers and streams from/to some encoding to/from UTF-8. It also implements a mechanism to handle message translation.

1.1 Character encoding conversion

The main feature of the XPRNLS library is character encoding conversion: it provides dedicated routines to ease the writing of programs that work with text data encoded with heterogeneous encodings. The reference encoding used by the XPRNLS library is UTF-8 and the functions it publishes handle conversions between UTF-8 and other encodings.

The functionality of the library is made accessible from the command line via the `xprnls` command tool that enables shell scripts to convert text files from one encoding to another.

Character encodings are identified by *encoding names*. The library supports natively the encodings UTF-8, UTF-16, UTF-32, ISO-8859-1, ISO-8859-15, CP1252 and US-ASCII: these encodings are therefore available on all systems. By default UTF-16 and UTF-32 use the byte order of the architecture of the running system (*e.g.* big endian on a Sparc processor) but the byte order may be selected by appending LE (Little Endian) or BE (Big Endian) to the encoding name (*e.g.* `UTF-16LE`).

The availability and names of other encodings depend on the operating system:

- On Windows the library relies on the win32 API routines *MultiByteToWideChar* and *WideCharToMultiByte*. The encoding names (that are not case sensitive) can be either the code page number prefixed by `CP` (like `CP28605`) or the usual name (*e.g.* `ISO-8859-15`). Except for `GB18030` (that is a variable size encoding), only single and 2-bytes encodings are supported.
- On Posix systems the library is based on the *iconv* function of the standard C library. Depending on the implementation the encoding names may be case sensitive.

An encoding name may also be one of the following aliases: `RAW` (no encoding), `SYS` (default system encoding), `WCHAR` (wide character for the C library), `FNAME` (encoding used for file names), `TTY` (encoding of the output stream of the console), `TTYIN` (encoding of the input stream of the console), `STDIN`, `STDOUT`, `STDERR` (encoding of the default input/output/error stream).

1.2 Message translation with XPRNLS

The creation of message translations typically involves three steps:

1. extraction of the message texts to be translated from a program source (⇒ *Portable Object Template* (POT) file)
2. instantiation with the translations for a particular language (⇒ *Portable Object* (PO) file)
3. compilation of the message translations (⇒ Machine Object (MO) file)

Translations are stored in a set of *message catalog* files: each of these files is specific to a language and a *domain*. A domain is a collection of messages, typically all messages of an application are grouped under a given domain. The `xprnls` command tool supports the necessary operations for building and managing these message catalog files.

Translations are applied in programs via the XPRNLS *gettext* framework for message translation: in a program using this system all strings to be translated are passed to a translation function (`XNLSgettext`). During the execution of the program this function returns a version of the message for the current language or the original English text itself if no translation can be found.

1.2.1 Example

The following example shows a minimal program using the message translation functionality:

```
int main(int argc,char *argv[])
{
 XNLSdomain dom;

 dom=XNLSopenmsgdom("myprg",NULL);    /* Open domain 'myprg' */
 printf(XNLSgettext(dom,"Hello!\n")); /* Display translation of "hello" */
 XNLSclosemsgdom(dom);                /* Close domain */
 return 0;
}
```

This example requires message catalog files for the domain `"myprg"`. The first step in the generation of the message catalogs is to produce a *Portable Object Template* (POT) file for the domain: this text file collects all messages to be translated. For our example the file `myprg.pot` includes only one message (the generation of such a file can be automated using tools like `GNU xgettext`):

```
msgid "Hello!\n"
msgstr ""
```

From this template one *Portable Object* (PO) file per supported language has to be produced. The creation of an initial PO file can be done using the XPRNLS command tool (see Section 2). For instance, to generate the file `myprg.fr.po` (we assume that the operating system is configured for French):

```
xprnls init -o myprg.fr.po myprg.pot
```

The contents of the file `myprg.fr.po` generated by this command looks as follows:

```
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
```

```
"POT-Creation-Date: 2015-12-01 16:03+0100\n"
"PO-Revision-Date: 2015-12-01 16:03+0100\n"
"Last-Translator: Your name\n"
"Language: fr\n"
"Content-Type: text/plain; charset=ISO8859-15\n"

msgid "Hello!\n"
msgstr ""
```

In addition to the message to translate the command tool has created a *header record*: this portion is mostly just informative (it is however recommended to complete the missing entries) but 2 entries are exploited by XPRNLS, namely the *language* (of this translation) and *content type* (encoding of the file) must be correct: here, the language is French (fr) and the encoding is ISO-8859-15 — these settings correspond to the configuration of the system on which we have performed the `xprnls` command. It will be necessary to edit these values when preparing translations for a language that is different from the configuration of the operating system.

Completing a PO file consists in entering a translation for each of the messages (*i.e.* make sure that every `msgid` entry is followed by a non-empty `msgstr` record). Our example has only one message, the minimal form of our translation file for French therefore is the following:

```
msgid ""
msgstr ""
"Language: fr\n"
"Content-Type: text/plain; charset=ISO8859-15\n"

msgid "Hello!\n"
msgstr "Bonjour!\n"
```

Once all translations have been prepared, the message catalogs are created by running the following command:

```
xprnls mogen -d locale myprg.*.po
```

This command creates the directory `locale` (if necessary) that contains one subdirectory per language. In each of these language specific subdirectories another directory (`LC_MESSAGES`) is created to store the message catalogs. Each message catalog is named after the domain name with the extension `.mo` (*Machine Object*): this is the binary version of the corresponding PO file. When executing our example program on a system configured for French the function `XNLSgettext` looks for the file `locale/fr/LC_MESSAGES/myprg.mo` to locate the required translations.

1.2.2 PO, POT and MO file formats

Portable Object (PO) and Portable Object Template (POT) files are text files consisting in a list of pairs of entries of the form:

```
msgid  message
msgstr  translation
```

Where *message* is a text identifying the message to translate (it is usually expressed in English) and *translation* its translation in the language associated to the file. A POT file has only empty `msgstr` entries and serves as a basis for the creation of the PO files.

The `'#'` symbol marks the beginning of a comment: any text following after it up to the end of line is ignored.

Both the message and its translation must be expressed in the form of lists of double-quoted strings separated by spaces or newlines (each list is merged into a single text string by the PO

processor). Text strings may contain C escape sequences (like `"\n"`) as well as format markers (*e.g.* `"%s"`). A translation must include the same format markers as the original text and they must appear in the same order (otherwise the translation will be ignored).

Usually PO files include the special message *empty string* (`""`) the translation of which is used to record management information (like name of the author, date of creation etc) instead of an actual translation. The syntax of such a record is a succession of definitions of the form:

property : *value* \n

Although the command `xprnls init` (used to create an initial PO file from a POT file) will generate a certain number of assignments for this header, only two of them are effectively used by XPRNLS: the language associated to the file (*e.g.* `"Language: it\n"`) and its encoding (*e.g.* `"Content-Type: text/plain; charset=ISO8859-1\n"`).

A Machine Object (MO) file is a compiled version of a PO file that is created by the command `xprnls mogen`: this is the format required by the translation routines of XPRNLS. The binary MO format used by XPRNLS is platform independent and compatible with the GNU version of *gettext* (GMO).

CHAPTER 2

XPRNLS Command line tool

The command line tool `xprnls` is typically used with the following syntax from an operating system console:

```
xprnls  command  command_arguments
```

The *command* parameter is one of the following commands and *command_arguments* are the associated arguments (square brackets indicate optional arguments):

`info`
: Display system configuration information relative to native language support: current system language, character encodings of system, console, file system, and actual encoding of the C type `wchar` (wide characters).

`conv` `[-s] [-f frenc] [-dos|-unix|-sys] [-t toenc] [-o dest] [-bom|-nobom] file`
: Convert the text file `file` from the character encoding `frenc` (default: UTF-8) to character encoding `toenc` (default: UTF-8). The resulting file is saved into `dest` (default: console). By default encoding errors are ignored (*e.g.* an incomplete sequence is replaced by some default symbol) but using option '`-s`' makes the conversion fail in case of error. The option '`-nobom`' disables the insertion of a BOM (Byte Order Mark) at the beginning of the file when it is required (a BOM is inserted when producing UTF-16 or UTF-32 documents) while option '`-bom`' forces insertion of a BOM when creating an UTF-8 document (this is usually not required for this encoding).
The options '`-dos`' and '`-unix`' select the appropriate settings regarding BOM and line termination for the specified environment. The option '`-sys`' selects the system encoding for the destination file and enables option '`-dos`' or '`-unix`' depending on the executing environment.

`poconv` `[-s] [-dos|-unix|-sys] [-t toenc] [-o dest] [-bom|-nobom] pofile`
: This command is a special version of the `conv` command described above specifically designed for PO (Portable Object) files (see Section 1.2): it retrieves the source encoding from the file header and updates the output header according to the destination encoding.

`init` `[-o dest] potfile`
: Create an initial PO file from the provided POT (Portable Objecy Template) file `potfile`. The resulting data is sent to the console or saved into `dest`.

`mogen` `[-d dir] [-o dest] pofile [[-o dest] pofile [...]]`
: Compile a PO file `pofile` into an MO (Machine Object) file. If no destination name `dest` is provided the resulting file has the same name as the source file with the file extension '`.mo`' instead of '`.po`'. If a locale directory `dir` is specified then the resulting file is saved under '`dir`/*lang*/`LC_MESSAGES`' where *lang* is the language specified in the PO file (missing directories are automatically created). The destination file name is of the form '*domain*`.mo`' assuming the source file name is of the form '*domain.lang*`.po`'. Several PO files may be specified for a single operation.

update `[-f] [-m] pofile [pofile ...]  potfile`
: Update a group of PO files with the POT file `potfile`: messages of the POT file that are missing from the PO files are added and messages not included in the POT file are turned into comments. If the option '`-m`' is used then the comment "`#, missing`" is put before missing translations (to ease their localisation). The PO file is not changed if it already contains all messages of the POT file unless option '`-f`' is selected.

merge `[-o dest] [-c] pofile [pofile ...]`
: Merge a collection of PO (or POT) files, the resulting data is saved into `dest` (default: console). If the option '`-c`' is used then the first PO file is completed with the other files (*i.e.* missing translations of the first file are searched for in the other files but no additional message is added).

CHAPTER 3

XPRNLS library

3.1 General

The XPRNLS library must be initialized with a call to `XNLSinit` before being used. This initialization function may be called several times and each call must be matched with a call to `XNLSfinish` after terminating the use of the library.

All encoding conversion routines expect an *encoding ID* to identify an encoding. This numerical identifier is obtained from an encoding name via `XNLSgetencid`.

XNLSinit

Purpose

Initialize the library.

Synopsis

```
int XNLSinit(void);
```

Return value

0 if executed successfully, any other value indicates a failure.

Further information

1. This function initializes the library. It must be called before any other function described in this document may be executed.
2. This initialisation procedure may be called more than once. In this case the termination routine `XNLSfinish` must be used the same number of times in order for all resources to be cleared.

Related topics

`XNLSfinish`.

XNLSfinish

Purpose

Release resources used by the library.

Synopsis

```
void XNLSfinish(void);
```

Further information

This function should be called when the library is not longer required. It releases all resources allocated by the library since its initialization.

Related topics

XNLSinit.

XNLSgetencid

Purpose

Get the ID associated with an encoding name.

Synopsis

```
int XNLSgetencid(char *enc);
```

Argument

`enc` Encoding name

Return value

Encoding ID or a negative value in case of failure.

Further information

1. All encoding conversion routines require an encoding ID to identify the encoding to use: this routine returns the ID associated with a given encoding name.
2. XPRNLS supports natively `UTF-8`, `UTF-16LE` (Little Endian), `UTF-16BE` (big endian), `UTF-32LE`, `UTF-32BE`, `US-ASCII`, `ISO-8859-1`, `ISO-8859-15`, `CP1252` and `RAW` (no encoding). For these encodings the same IDs are always returned and a call to this function can be replaced by the corresponding constants: `XNLS_ENC_UTF8`, `XNLS_ENC_UTF16LE`, `XNLS_ENC_UTF16BE`, `XNLS_ENC_UTF32LE`, `XNLS_ENC_UTF32BE`, `XNLS_ENC_ASCII`, `XNLS_ENC_88591`, `XNLS_ENC_885915`, `XNLS_ENC_CP1252` and `XNLS_ENC_RAW`. With other encoding names the function may return a different ID for a given encoding after the library has been reset or the program restarted.
3. The availability of not natively supported encodings (for instance `"iso-8859-3"`) depends on the operating system.
4. In addition to proper encoding names the function also accepts the following aliases: `"SYS"` (default system encoding), `"WCHAR"` (wide char representation for the system), `"FNAME"` (file names, on most systems this is the same as "SYS"), `"TTY"` (encoding of the console output stream), `"TTYIN"` (encoding of the input stream of the console, this is usually the same as "TTY"), `"STDIN"` (encoding of the default input stream), `"STDOUT"` (encoding of the default output stream), `"STDERR"` (encoding of the default error stream), `"UTF-16"` (UTF-16LE or UTF-16BE depending on the architecture), `"UTF-32"` (UTF-32LE or UTF-32BE depending on the architecture).

Related topics

`XNLSgetencname`.

XNLSgetencname

Purpose

Get the name corresponding to an encoding ID.

Synopsis

```
const char *XNLSgetencname(int eid);
```

Argument

`eid` Encoding ID

Return value

Encoding name or an empty string if the ID is not valid.

Related topics

`XNLSgetencid`.

3.2 Handling of program parameters

In addition to transcoding routines the library also provides an operating system independent method for retrieving program parameters (namely `argc` and `argv` of the `main` function of the C program) encoded as UTF-8. The procedure requires the main program to be named `XNLS_MAIN` and the `argv` parameter to be of type `XNLSargv`. The function `XNLSconvargv` can then be called to get the converted arguments. The following example shows how to organise the main function of a program using this feature:

```
int XNLS_MAIN(int argc,XNLSargv sargv[])
{
 char **argv;

 argv=XNLSconvargv(&argc,sargv);   /* 'argv' is encoded in UTF-8 */

 /* Insert here the body of the function using 'argc' and 'argv' as usual */

 XNLSfreeargv(argv);
 return 0;
}
```

XNLSconvargv	Generate a UTF-8 version of the program parameters.	p. 13
XNLSfreeargv	Release the datastructures allocated by XNLSconvargv.	p. 14
XNLSprogpath	Get the full path to the running program.	p. 15

XNLSconvargv

Purpose

Generate a UTF-8 version of the program parameters.

Synopsis

```
char **XNLSconvargv(int *argc, XNLSargv argv[]);
```

Arguments

`argc`	Number of program parameters
`argv`	Array of program parameters

Return value

A version of `argv` encoded in UTF-8.

Further information

1. This function calls `XNLSinit`, it is therefore not necessary to initialise the library when using this routine.
2. Under Windows the arguments including wildcard characters are expanded (on Posix systems this is done by the shell). As a consequence the number of arguments `argc` may be increased by the call.

Related topics

`XNLSfreeargv`.

XNLSfreeargv

Purpose

Release the datastructures allocated by `XNLSconvargv`.

Synopsis

```
void XNLSfreeargv(char **argv);
```

Argument

`argv` Array of program parameters returned by a previous call to `XNLSconvargv`

Further information

This function calls `XNLSfinish`.

Related topics

`XNLSconvargv`.

XNLSprogpath

Purpose

Get the full path to the running program.

Synopsis

```
const char *XNLSprogpath(const char *name);
```

Argument

`name` Program name or path (*e.g.* `argv[0]`) encoded in UTF-8.

Return value

Full path to running program or `name` in case of error.

Further information

1. On Posix systems the argument `name` should be `argv[0]`: it is expanded in order to get a full path. Under Windows it is not used (unless an error occurs).
2. The string returned by this function is the result of a call to `XNLSconvstrfrom`.

Related topics

`XNLSfreeargv`.

3.3 Buffer and string encoding conversion

The functions in this section serve for converting character strings from and to UTF-8 encoding. The functions `XNLSconvstrfrom` and `XNLSconvbuffrom` return a statically allocated buffer (that is re-used each time any of these functions is called) while `XNLSconvbuffrom` and `XNLSconvbufto` require a destination buffer.

XNLSconvstrfrom

Purpose

Convert a text string to UTF-8.

Synopsis

```
const char *XNLSconvstrfrom(int eid, const char *src, int srclen, int *dstlen);
```

Arguments

`eid`	Encoding ID of the source
`src`	Text buffer to transcode
`srclen`	Length in bytes of the source text buffer (or -1 for a null terminated string)
`dstlen`	A location where to return the length (in bytes) of the generated string

Return value

The converted null terminated string or `NULL` in case of memory allocation error.

Further information

1. This function returns a thread-specific statically allocated buffer that it shares with `XNLSconvstrto`: each call to any of these functions will overwrite the result of the previous call.
2. Any invalid character is replaced by a default symbol (*middle dot*) in the converted string.
3. Mosel uses this function extensively: it is therefore recommended to duplicate any string returned by this routine if you need to pass it to a Mosel API function. Alternatively, use the `XNLSconvbuffrom` form of this function where you can specify your own buffer.

Related topics

`XNLSconvbuffrom`, `XNLSconvstrto`.

XNLSconvstrto

Purpose

Convert a UTF-8 text string to a given encoding.

Synopsis

```
const char *XNLSconvstrto(int eid, const char *src, int srclen, int *dstlen);
```

Arguments

`eid`	Encoding ID of the destination
`src`	Text buffer to transcode
`srclen`	Length in bytes of the source text buffer (or -1 for a null terminated string)
`dstlen`	A location where to return the length (in bytes) of the generated string (can be `NULL`)

Return value

The converted null terminated string or `NULL` in case of memory allocation error.

Further information

1. This function returns a thread-specific statically allocated buffer that it shares with `XNLSconvstrfrom`: each call to any of these functions will overwrite the result of the previous call.
2. Any invalid character is replaced by a default symbol (specific to the destination encoding) in the converted string.
3. Mosel uses this function extensively: it is therefore recommended to duplicate any string returned by this routine if you need to pass it to a Mosel API function. Alternatively, use the `XNLSconvbufto` form of this function where you can specify your own buffer.

Related topics

`XNLSconvbufto`, `XNLSconvstrfrom`.

XNLSconvbuffrom

Purpose

Convert a text buffer to UTF-8.

Synopsis

```
int XNLSconvbuffrom(int eid, char **srcstart, char *srcend, char **dststart, char
       *dstend, int flags);
```

Arguments

`eid`	Encoding ID of the source
`srcstart`	Reference to a pointer at the beginning of the input buffer
`srcend`	Pointer to the character after the end of the input buffer
`dststart`	Reference to a pointer at the beginning of the output buffer
`dstend`	Pointer to the character after the end of the output buffer
`flags`	Conversion flags (can be combined): `XNLS_UTF_FLAG_STRICT` Fail in case of invalid sequence (otherwise skip it) `XNLS_UTF_FLAG_PARTIAL` Stop and return `XNLS_CONV_PARTIAL` when buffer ends on an incomplete sequence (instead of failing)

Return value

`XNLS_CONV_OK` Function executed sucessfully

`XNLS_CONV_PARTIAL` Input buffer terminates on an incomplete sequence (process stopped at the beginning of the sequence)

`XNLS_CONV_DSTOUT` Output buffer not large enough

`XNLS_CONV_FAIL` Process stopped at an invalid sequence

Further information

The arguments `srcstart` and `dststart` are updated such that they point to the byte following the decoded sequence (input buffer) or the first unused byte (output buffer) after the function returns.

Related topics

`XNLSconvbufto`, `XNLSconvstrfrom`.

XNLSconvbufto

Purpose

Convert a text buffer from UTF-8 to a given encoding.

Synopsis

```
int XNLSconvbufto(int eid, char **srcstart, char *srcend, char **dststart, char
       *dstend, int flags);
```

Arguments

`eid`	Encoding ID of the destination
`srcstart`	Reference to a pointer at the beginning of the input buffer
`srcend`	Pointer to the character after the end of the input buffer
`dststart`	Reference to a pointer at the beginning of the output buffer
`dstend`	Pointer to the character after the end of the output buffer
`flags`	Conversion flags (can be combined): `XNLS_UTF_FLAG_STRICT` Fail in case of invalid sequence (otherwise skip it) `XNLS_UTF_FLAG_PARTIAL` Stop and return `XNLS_CONV_PARTIAL` buffer ends on an incomplete sequence (instead of failing)

Return value

`XNLS_CONV_OK` Function executed sucessfully

`XNLS_CONV_PARTIAL` Input buffer terminates on an incomplete sequence (process stopped at the beginning of the sequence)

`XNLS_CONV_DSTOUT` Output buffer not large enough

`XNLS_CONV_FAIL` Process stopped at an invalid sequence

Further information

The arguments `srcstart` and `dststart` are updated such that they point to the byte following the decoded sequence (input buffer) or the first unused byte (output buffer) after the function returns.

Related topics

`XNLSconvbuffrom`, `XNLSconvstrto`.

XNLSutf8tocode

Purpose

Get the corresponding code point of a UTF-8 sequence.

Synopsis

```
int XNLSutf8tocode(const char **src, const char *srcend);
```

Arguments

`src`	Reference to a pointer at the beginning of the sequence
`srcend`	Pointer to the character after the end of the buffer

Return value

Unicode code point or `-1` if the sequence is not valid or incomplete

Further information

The argument `src` is updated such that it points to the byte following the decoded sequence after the function returns.

Related topics

`XNLScodetoutf8`.

XNLScodetoutf8

Purpose

Generate a UTF-8 sequence from a Unicode code point.

Synopsis

```
int XNLScodetoutf8(unsigned int ucp, char *dst);
```

Arguments

`ucp`	Code point
`dst`	Destination buffer (must be of at least 4 bytes length)

Return value

Number of bytes written to the ouput buffer (between 1 and 4)

Related topics

`XNLSutf8tocode`.

3.4 Stream encoding conversion

These functions are designed for the transcoding of text streams. The creation of a stream with XNLSopenconv requires an additional function used to read (or write for an output stream) a block of data: it is called whenever the internal buffer of the transcoder is empty (or full if writing). When the stream is open for reading the user calls iteratively the reader routine XNLSconvread to get the data of the stream encoded in UTF-8. If the stream is open for writing the data sent to XNLSconvwrite is expected to be encoded in UTF-8 and it is transcoded to the encoding specified at the creation of the stream.

XNLSopenconv

Purpose

Open a transcoder stream.

Synopsis

```
XNLSstream XNLSopenconv(int eid, int flags, long (*fsync)(void *vctx,void *fd,void
       *buf,unsigned long bufsize), void *fd);
```

Arguments

`eid`	Encoding ID of the destination	
`flags`	Conversion flags:	
	`XNLS_OPT_READ`	Convert to UTF-8
	`XNLS_OPT_WRITE`	Convert from UTF-8
	`XNLS_OPT_STRICT`	Fail in case of invalid sequence (otherwise skip it)
	`XNLS_OPT_NOBOM`	Do not put a BOM when writing (by default a BOM is emitted for UTF-16 and UTF-32) and do not look for a BOM when reading
	`XNLS_OPT_BOM`	Force BOM (by default no BOM is emitted for UTF-8), option ignored for a reader stream
`fsync`	Data reader or writer function	
`fd`	Data reader/writer file descriptor	

Return value

A stream context or `NULL` in case of error.

Further information

1. When a stream is created for reading (*i.e.* option *flags* is `XNLS_OPT_READ`), input data is expected to be encoded in the specified encoding `eid`. The converter gets the stream of data to process from a callback function: the provided *read* function `fsync` takes as parameters some general context `vctx` (provided to `XNLSconvread`), the file descriptor for this stream (specified with this call as `fd`) and a buffer `buf` where it puts up to `bufsize` bytes of data. This function must return the amount of data transfered, `0` when the end of file has been reached or a negative value to indicate an error condition.

2. When a stream is created for writing (*i.e.* option *flags* is `XNLS_OPT_WRITE`), output data is encoded in the specified encoding `eid`. The converter outputs the stream of data it has processed using a callback function: the provided *write* function `fsync` takes as parameters some general context `vctx` (provided to `XNLSconvwrite`), the file descriptor for this stream (specified with this call as `fd`) and a buffer `buf` of `bufsize` bytes of data to save. This function is expected to return a non-zero value on success (*i.e.* all data has been saved) and `0` on failure.

3. The option `XNLS_OPT_NOBOM` tells the routine not to try to find a BOM (Byte Order Mark) when opening a stream for reading. This BOM consists in a sequence of bytes at the beginning of the stream to identify the Unicode encoding used for the stream. Unless the option `XNLS_OPT_NOBOM` is used, this BOM detection is effective when the selected encoding is Unicode (either UTF-8, UTF-16 or UTF-32). When applied to a stream open for writing, this option disables the insertion of a BOM when creating a stream encoded in UT-F16 or UTF-32 (it is gnored for all other encodings).

4. The option `XNLS_OPT_BOM` forces the insertion of the BOM when openning a stream encoded in UTF-8 for writing (this encoding usually does not require a BOM).

Related topics

`XNLScloseconv`.

XNLSconvwrite

Purpose

Write to an output transcoder stream.

Synopsis

```
long XNLSconvwrite(void *vctx,XNLSstream stream,const void *buf,unsigned long size);
```

Arguments

`vctx`	Runtime context for data writer
`stream`	Stream open for writing
`buf`	Buffer to output encoded in UTF-8
`size`	size of `buf`

Return value

`1` if successful, `-1` in case of conversion error and `0` for other errors.

Further information

The parameter `vctx` is passed as the first argument to the `sync` function defined when opening the stream.

Related topics

`XNLSopenconv`.

XNLSconvread

Purpose

Read from an input transcoder stream.

Synopsis

```
long XNLSconvread(void *vctx,XNLSstream stream,void *buf,unsigned long size);
```

Arguments

`vctx`	Runtime context for data reader
`stream`	Stream open for reading
`buf`	Destination buffer for the UTF-8 encoded string
`size`	size of `buf`

Return value

The number of bytes read if successful, `-1` in case of conversion error, `-2` for other errors and `0` when the end of file has been reached.

Further information

The parameter `vctx` is passed as the first argument to the `sync` function defined when opening the stream.

Related topics

`XNLSopenconv`.

XNLSgetfd

Purpose

Get reader/writer file descriptor of a transcoder stream.

Synopsis

```
void *XNLSgetfd(XNLSstream stream);
```

Argument

`stream` A stream

Return value

Data reader/writer file descriptor as passed to `XNLSopenconv`.

Related topics

`XNLSgetenc`, `XNLSopenconv`.

XNLSgetenc

Purpose

Get the encoding ID and status of a transcoder stream.

Synopsis

```
int XNLSgetenc(XNLSstream stream, int *status);
```

Arguments

`stream` A stream

`status` An area where to return the current status of the stream (may be `NULL`)

`0`	initial state
`1`	normal processing
`2`	a BOM has been found
`3`	end of file
`4`	error

Return value

Current encoding ID of the stream.

Related topics

`XNLSgetfd`, `XNLSopenconv`.

XNLSgetoffset

Purpose

Get the current offset in a transcoder stream.

Synopsis

```
int XNLSgetoffset(XNLSstream stream);
```

Argument

`stream` A stream

Return value

Current offset in the stream.

Further information

The *offset* corresponds to the total amount of data written (for an output stream) or read (for an input stream).

Related topics

`XNLSgetfd`.

XNLScloseconv

Purpose

Close a transcoder stream.

Synopsis

```
int XNLScloseconv(void *vctx, XNLSstream stream);
```

Arguments

`vctx`	Runtime context for data reader/writer
`stream`	Stream to close

Return value

`0` if successful, any other value indicates an error.

Related topics

`XNLSopenconv`.

3.5 Translation

The library publishes an implementation of the so-called *gettext* system for automatic translation of text strings in a program. This system relies on *message catalog* files containing for each message of the application both the English version and its translation in a particular language (there is therefore one message catalog file per language and application). The message catalog files are organised in a dedicated directory hierarchy: the root directory of the localisation data (usually named `locale`) contains one sub-directory per language named after its ISO 639 code (for instance `de` for German, `it` for Italian etc). The message catalogs for a given language are stored under the `LC_MESSAGES` sub-directory of this language specific directory. Each message catalog file has the name of the *domain* it provides translations for (typically the name of the application) with the extension `.mo` (Machine Object). An `mo` file is generated using the command `xprnls mogen` (see Section 2) from a `po` file (Portable Object).

A program using the XPRNLS translation framework must first open the domain for which it needs translations using `XNLSopenmsgdom` in order to get a domain descriptor. Then each message translation can be obtained by applying `XNLSgettext` to the English version of the text: the translated message is returned if the message catalog file for the current language has been found and includes the translation. Otherwise the original English text is used as the return value.

`XNLSclosemsgdom`	Close a message domain.	p. 34
`XNLSfindmsgdom`	Find an open message domain based on its name.	p. 35
`XNLSgettext`	Get the translation of a text string.	p. 37
`XNLSopenmsgdom`	Open a message domain.	p. 33
`XNLSsetlang`	Set or get the active language for message translation.	p. 32
`XNLSsetlocaledir`	Define the default or domain specific locale directory.	p. 36

XNLSsetlang

Purpose

Set or get the active language for message translation.

Synopsis

```
const char *XNLSsetlang(const char *lang);
```

Argument

`lang` An ISO 639 language code (*e.g.* en, de, ja, zh...), or `NULL` or an empty string `""`

Return value

The new language code.

Further information

1. The language code is used to select the locale directory in which catalog files are searched from. During the library initialisation the language is set according to the current system settings.
2. If `NULL` is used as the language code the function returns the current language. If an empty string is given then the language is set to the system default setting (as after library initialisation).
3. Special language names `"C"`, `"POSIX"` as well as `"en"` disable translation (since initial messages are written in English).
4. All loaded message catalogs are unloaded when a call to this routine modifies the current language.

Related topics

`XNLSgettext`, `XNLSsetlocaledir`.

XNLSopenmsgdom

Purpose

Open a message domain.

Synopsis

```
XNLSdomain XNLSopenmsgdom(const char *name, const char *localedir);
```

Arguments

`name`	Domain name
`localedir`	Locale directory for this domain (can be `NULL`)

Return value

A context for the domain or `NULL` in case of error.

Further information

1. The default locale directory is used if the `localedir` parameter is `NULL`. The current working directory will be used if the resulting locale directory is `NULL`.
2. The domain name `name` is used to build file names of message catalogs. For instance the message catalog for domain `"mosel"`, language `"es"` in the locale directory `"/usr/share/locale"` is `"/usr/share/locale/es/LC_MESSAGES/mosel.mo"`
3. Message catalogs are opened when the first `XNLSgettext` call is issued, not at the time of opening the domain.
4. The same domain context will be returned if a given domain name is open several times (the `localedir` parameter is ignored when the domain is already open). The system keeps track of the number of times each domain context has been returned and requires an equal number of calls to `XNLSclosemsgdom` to release properly the resources associated with the domain.

Related topics

`XNLSgettext`, `XNLSsetlocaledir`, `XNLSclosemsgdom`.

XNLSclosemsgdom

Purpose

Close a message domain.

Synopsis

```
void XNLSclosemsgdom(XNLSdomain domain);
```

Argument

`domain` Domain to close

Further information

Every call to `XNLSopenmsgdom` should be completed by a matching call to this procedure once the domain is no longer required in order to release the resources associated with the domain.

Related topics

`XNLSopenmsgdom`.

XNLSfindmsgdom

Purpose

Find an open message domain based on its name.

Synopsis

```
XNLSdomain XNLSfindmsgdom(const char *name);
```

Argument

`name` Name of the doamin

Return value

A context for the domain or `NULL` if no domain of this name has been open so far.

Further information

This function does not modify the reference count of the domain.

Related topics

`XNLSopenmsgdom`.

XNLSsetlocaledir

Purpose

Define the default or domain specific locale directory.

Synopsis

```
const char *XNLSsetlocaledir(XNLSdomain domain, const char *localedir);
```

Arguments

`domain`	Domain (`NULL` to change the global default setting)
`localedir`	Locale directory (can be `NULL`)

Return value

The new locale directory.

Further information

1. Modifying the locale directory of a domain has no effect on already loaded message catalogs (*i.e.* they are not reloaded from the new location).
2. Changing the default locale directory does not affect the domains that are already open.

Related topics

`XNLSopenmsgdom`.

XNLSgettext

Purpose

Get the translation of a text string.

Synopsis

```
const char *XNLSgettext(XNLSdomain domain, const char *txt);
```

Arguments

`domain`	Domain
`txt`	Message to translate

Return value

Translated message or `txt` if no translation could be found.

Further information

The function returns its argument `txt` if the current language is English or no translation is found or the text to translate is `NULL` or an empty string.

Related topics

`XNLSopenmsgdom`.

APPENDIX A

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

FICO offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have purchased a FICO product and have an active support or maintenance contract. You can find support contact information on the Product Support home page (www.fico.com/support).

On the Product Support home page, you can also register for credentials to log on to FICO Online Support, our web-based support tool to access Product Support 24x7 from anywhere in the world. Using FICO Online Support, you can enter cases online, track them through resolution, find articles in the FICO Knowledge Base, and query known issues.

Please include *'Xpress'* in the subject line of your support queries.

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support. For additional information, visit the Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide. If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com.

Sales and maintenance

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 207 940 8718
Fax: +44 870 420 3601

Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Optimization Modeler to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

About FICO

FICO (NYSE:FICO) delivers superior predictive analytics solutions that drive smarter decisions. The company's groundbreaking use of mathematics to predict consumer behavior has transformed entire industries and revolutionized the way risk is managed and products are marketed. FICO's innovative solutions include the FICO® Score—the standard measure of consumer credit risk in the United States—along with industry-leading solutions for managing credit accounts, identifying and minimizing the impact of fraud, and customizing consumer offers with pinpoint accuracy. Most of the world's top banks, as well as leading insurers, retailers, pharmaceutical companies, and government agencies, rely on FICO solutions to accelerate growth, control risk, boost profits, and meet regulatory and competitive demands. FICO also helps millions of individuals manage their personal credit health through www.myfico.com. Learn more at www.fico.com. FICO: Make every decision count™.

Index