NAME

FN_string_t, fn_string_create, fn_string_destroy, fn_string_from_str, fn_string_from_str_n, fn_string_str, fn_string_from_contents, fn_string_code_set, fn_string_charcount, fn_string_bytecount, fn_string_contents, fn_string_copy, fn_string_assign, fn_string_from_strings, fn_string_from_substring, fn_string_is_empty, fn_string_compare, fn_string_compare_substring, fn_string_next_substring, fn_string_prev_substring – a character string

SYNOPSIS

cc [*flag* ...] *file* ... **–lxfn** [*library* ...]

#include <xfn/xfn.h>

FN_string_t *fn_string_create(void);

void fn_string_destroy(FN_string_t **str***);**

FN_string_t *fn_string_from_str(const unsigned char **cstr***);**

FN_string_t *fn_string_from_str_n(const unsigned char **cstr***, size_t** *n***);**

const unsigned char *fn_string_str(const FN_string_t **str***, unsigned int** **status***);**

FN_string_t *fn_string_from_contents(unsigned long *code_set***, const void** **locale_info***, size_t** *locale_info_len***, size_t** *charcount***, size_t** *bytecount***, const void** **contents***, unsigned int** **status***);**

unsigned long fn_string_code_set(const FN_string_t **str***, const void** ***locale_info***, size_t** **locale_info_len***);**

size_t fn_string_charcount(const FN_string_t **str***);**

size_t fn_string_bytecount(const FN_string_t **str***);**

const void *fn_string_contents(const FN_string_t **str***);**

FN_string_t *fn_string_copy(const FN_string_t **str***);**

FN_string_t *fn_string_assign(FN_string_t **dst***, const FN_string_t** **src***);**

FN_string_t *fn_string_from_strings(unsigned int **status***, const FN_string_t** **s1***, const FN_string_t** **s2***, ...**

FN_string_t *fn_string_from_substring(const FN_string_t **str***, int** *first***, int** *last***);**

int fn_string_is_empty(const FN_string_t **str***);**

int fn_string_compare(const FN_string_t **str1***, const FN_string_t** **str2***, unsigned int** *string_case***, unsigned int** **status***);**

int fn_string_compare_substring(const FN_string_t **str1***, int** *first***, int** *last***, const FN_string_t** **str2***, unsigned int** *string_case***, unsigned int** **status***);**

int fn_string_next_substring(const FN_string_t **str***, const FN_string_t** **sub***, int** *index***, unsigned int** *string_case***, unsigned int** **status***);**

int fn_string_prev_substring(const FN_string_t **str***, const FN_string_t** **sub***, int** *index***, unsigned int** *string_case***, unsigned int** **status***);**

MT-LEVEL

Safe.

DESCRIPTION

The **FN_string_t** type is used to represent character strings in the XFN interface. It provides insulation from specific string representations.

The **FN_string_t** supports multiple code sets. It provides creation functions for character strings of the code set of the current locale setting and a generic creation function for arbitrary code sets. The degree of support for the functions that manipulate **FN_string_t** for arbitrary code sets is implementation-dependent. An XFN implementation is required to support the **ISO 646** code set; all other code sets are optional.

fn_string_destroy( ) releases the storage associated with the given string.

fn_string_create( ) creates an empty string.

fn_string_from_str( ) creates an **FN_string_t** object from the given null terminated string based on the code set of the current locale setting. The number of characters in the string is determined by the code set of the current locale setting. **fn_string_from_str_n( )** is like **fn_string_from_str( )** except only *n* characters from the given string are used. **fn_string_str( )** returns the contents of the given string *str* in the form of a null terminated string in the code set and current locale setting.

fn_string_from_contents( ) creates an **FN_string_t** object using the specified code set *code_set*, locale information *locale_info* and data in the given buffer *contents*. *bytecount* specifies the number of bytes in *contents* and *charcount* specifies the number of characters represented by *contents*.

fn_string_code_set( ) returns the code set associated with the given string object, and if present, the locale information in *locale_info*. **fn_string_charcount( )** returns the number of characters in the given string object. **fn_string_bytecount( )** returns the number of bytes used to represent the given string object. **fn_string_contents( )** returns a pointer to the contents of the given string object.

fn_string_copy( ) returns a copy of the given string object. **fn_string_assign( )** makes a copy of the string object *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned. **fn_string_from_strings( )** is a function that takes a variable number of arguments (minimum of 2), the last of which must be **NULL** (**0**); it returns a new string object composed of the left to right concatenation of the given strings, in the given order. The support for strings with different code sets and/or locales as arguments to a single invocation of **fn_string_from_strings( )** is implementation-dependent. **fn_string_from_substring( )** returns a new string object consisting of the characters located between *first* and *last* inclusive from *str*. Indexing begins with **0**. If *last* is **FN_STRING_INDEX_LAST** or exceeds the length of the string, the index of the last character of the string is used.

fn_string_is_empty( ) returns whether *str* is an empty string.

Comparison of two strings must take into account code set and locale information. If strings are in the same code set and same locale, case sensitivity is applied according to the case sensitivity rules applicable for the code set and locale; case sensitivity may not necessarily be relevant for all string encodings. If *string_case* is non-zero, case is significant and equality for strings of the same code set is defined as equality between byte-wise encoded values of the strings. If *string_case* is zero, case is ignored and equality for strings of the same code set is defined using the definition of case-insensitive equality for the specific code set. Support for comparison between strings of different code sets, or lack thereof, is implementation-dependent.

fn_string_compare( ) compares strings *str1* and *str2* and returns **0** if they are equal, non-zero if they are not equal. If two strings are not equal, **fn_string_compare( )** returns a positive value if the difference of *str2* precedes that of *str1* in terms of byte-wise encoded value (with case-sensitivity taken into account when *string_case* is non-zero), and a negative value if the difference of *str1* precedes that of *str2*, in terms of byte-wise encoded value (with case-sensitivity taken into account when *string_case* is non-zero). Such information (positive versus negative return value) may be used by applications that use strings of code sets in which ordering is meaningful; this information is not of general use in internationalized environments. **fn_string_compare_substring( )** is similar to **fn_string_compare( )** except **fn_string_compare_substring( )** compares characters between *first* and *last* inclusive of *str2* with *str1*. Comparison of strings with incompatible code sets returns a negative or positive value (never **0**) depending on the implementation.

fn_string_next_substring( ) returns the index of the next occurrence of *sub* at or after *index* in the string *str*. **FN_STRING_INDEX_NONE** is returned if *sub* does not occur. **fn_string_prev_substring( )** returns the index of the previous occurrence of *sub* at or before *index* in the string *str*. **FN_STRING_INDEX_NONE** is returned if *sub* does not occur. In both of these functions, *string_case* specifies whether the search should take case-sensitivity into account.

ERRORS

fn_string_str( ) returns **0** and sets *status* to **FN_E_INCOMPATIBLE_CODE_SETS** if the given string's representation cannot be converted into the code set of the current locale setting. It is implementation-dependent which code sets can be converted into the code set of the current locale.

Code set mismatches that occur during concatenation, searches or comparisons are resolved in an implementation-dependent way. When an implementation discovers that arguments to substring searches and comparison operations have incompatible code sets, it sets *status* to **FN_E_INCOMPATIBLE_CODE_SETS.** In such cases, **fn_string_from_strings( )** returns **0**. The returned value for comparison operations when there is code set or locale incompatibility is either negative or positive (greater than **0**); it is never **0**;

fn_string_from_contents( ) returns **0** and *status* is set to **FN_E_INCOMPATIBLE_CODE_SETS** if the supplied code set and/or locale information are not supported by the XFN implementation.

SEE ALSO

xfn(3N)