NAME

FN_composite_name_t, fn_composite_name_create, fn_composite_name_destroy, fn_composite_name_from_string, fn_string_from_composite_name, fn_composite_name_copy, fn_composite_name_assign, fn_composite_name_is_empty, fn_composite_name_count, fn_composite_name_first, fn_composite_name_next, fn_composite_name_prev, fn_composite_name_last, fn_composite_name_prefix, fn_composite_name_suffix, fn_composite_name_is_equal, fn_composite_name_is_prefix, fn_composite_name_is_suffix, fn_composite_name_prepend_comp, fn_composite_name_append_comp, fn_composite_name_insert_comp, fn_composite_name_delete_comp, fn_composite_name_prepend_name, fn_composite_name_append_name, fn_composite_name_insert_name – a sequence of component names spanning multiple naming systems

SYNOPSIS

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

#include <xfn/xfn.h>

FN_composite_name_t *fn_composite_name_create(void);

void fn_composite_name_destroy(FN_composite_name_t ***name*****);**

FN_composite_name_t *fn_composite_name_from_string(const FN_string_t ***str*****);**

FN_string_t *fn_string_from_composite_name(const FN_composite_name_t ***name*****,**
unsigned int ***status*****);**

FN_composite_name_t *fn_composite_name_copy(
const FN_composite_name_t ***name*****);**

FN_composite_name_t *fn_composite_name_assign(FN_composite_name_t ***dst*****,**
const FN_composite_name_t ***src*****);**

int fn_composite_name_is_empty(const FN_composite_name_t ***name*****);**

unsigned int fn_composite_name_count(const FN_composite_name_t ***name*****);**

const FN_string_t *fn_composite_name_first(const FN_composite_name_t ***name*****,**
void ****iter_pos*****);**

const FN_string_t *fn_composite_name_next(const FN_composite_name_t ***name***
void ****iter_pos*****);**

const FN_string_t *fn_composite_name_prev(const FN_composite_name_t ***name*****,**
void ****iter_pos*****);**

const FN_string_t *fn_composite_name_last(const FN_composite_name_t ***name*****,**
void ****iter_pos*****);**

FN_composite_name_t *fn_composite_name_prefix(
const FN_composite_name_t ***name*****, const void** ***iter_pos*****);**

FN_composite_name_t *fn_composite_name_suffix(
const FN_composite_name_t ***name*****, const void** ***iter_pos*****);**

int fn_composite_name_is_equal(const FN_composite_name_t ***name*****,**
const FN_composite_name_t ***name2*****, unsigned int** ***status*****);**

int fn_composite_name_is_prefix(const FN_composite_name_t **name*,
const FN_composite_name_t **prefix*, **void** ***iter_pos*, **unsigned int** **status***);**

int fn_composite_name_is_suffix(const FN_composite_name_t **name*,
const FN_composite_name_t **suffix*, **void** ***iter_pos*, **unsigned int** **status***);**

int fn_composite_name_prepend_comp(FN_composite_name_t **name*,
const FN_string_t **newcomp***);**

int fn_composite_name_append_comp(FN_composite_name_t **name*,
const FN_string_t **newcomp***);**

int fn_composite_name_insert_comp(FN_composite_name_t **name*,
void ***iter_pos*, **const FN_string_t** **newcomp***);**

int fn_composite_name_delete_comp(FN_composite_name_t **name*, **void** ***iter_pos***);**

int fn_composite_name_prepend_name(FN_composite_name_t **name*,
const FN_composite_name_t **newcomps***);**

int fn_composite_name_append_name(FN_composite_name_t **name*,
const FN_composite_name_t **newcomps***);**

int fn_composite_name_insert_name(FN_composite_name_t **name*,
void ***iter_pos*, **const FN_composite_name_t** **newcomps***);**

MT-LEVEL

Safe.

DESCRIPTION

A composite name is represented by an object of type **FN_composite_name_t**. Each component is a string name, of type **FN_string_t**, from the namespace of a single naming system. It may be an atomic name or a compound name in that namespace.

fn_composite_name_create creates an **FN_composite_name_t** object with zero components. Components may be subsequently added to the composite name using the modify operations described below. **fn_composite_name_destroy** releases any storage associated with the given **FN_composite_name_t** handle.

fn_composite_name_from_string( ) creates an **FN_composite_name_t** from the string *str* using the XFN composite name syntax. **fn_string_from_composite_name( )** returns the standard string form of the given composite name, by concatenating the components of the composite name in a left to right order, each separated by the XFN component separator.

fn_composite_name_copy( ) returns a copy of the given composite name object. **fn_composite_name_assign( )** makes a copy of the composite name object pointed to by *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the same object as *dst* is returned.

fn_composite_name_is_empty( ) returns **1** if the given composite name is an empty composite name (i.e. consists of a single, empty component name); otherwise, it returns **0**. **fn_composite_name_count( )** returns the number of components in the given composite name.

The iteration scheme is based on the exchange of an opaque **void** * argument, *iter_pos,* that serves to record the position of the iteration in the sequence. Conceptually, *iter_pos* records a position between two successive components (or at one of the extreme ends of the sequence).

The function **fn_composite_name_first( )** returns a handle to the **FN_string_t** that is the first component in the name, and sets *iter_pos* to indicate the position immediately following the first component. It returns **0** if the name has no components. Thereafter, successive calls of the **fn_composite_name_next( )** function return pointers to the component following the iteration marker, and advance the iteration marker. If the iteration marker is at the end of the sequence, **fn_composite_name_next( )** returns **0**. Similarly, **fn_composite_name_prev( )** returns the component preceding the iteration pointer and moves the marker back one component. If the marker is already at the beginning of the sequence, **fn_composite_name_prev( )** returns **0**. The function **fn_composite_name_last( )** returns a pointer to the last component of the name and sets the iteration marker immediately preceding this component (so that subsequent calls to **fn_composite_name_prev( )** can be used to step through leading components of the name).

The **fn_composite_name_suffix( )** function returns a composite name consisting of a copy of those components following the supplied iteration marker. The method **fn_composite_name_prefix( )** returns a composite name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized using **fn_composite_name_first( ), fn_composite_name_last( ), fn_composite_name_is_prefix( ),** or **fn_composite_name_is_suffix( )** yields undefined and generally undesirable behavior.

The functions **fn_composite_name_is_equal( ), fn_composite_name_is_prefix( ), fn_composite_name_is_suffix( ),** test for equality between composite names or between parts of composite names. For these functions, equality is defined as exact string equality, not name equivalence. A name's syntactic property, such as case-insensitivity, is not taken into account by these functions.

The function **fn_composite_name_is_prefix( )** tests if one composite name is a prefix of another. If so, it returns **1** and sets the iteration marker immediately following the prefix. (So for example, a subsequent call to **fn_composite_name_suffix( )** will return the remainder of the name.) Otherwise it returns **0** and value of the iteration marker is undefined. The function **fn_composite_name_is_suffix( )** is similar. It tests if a one composite name is a suffix of another. If so it returns **1** and sets the iteration marker immediately preceding the suffix.

fn_composite_name_prepend_comp( ) and **fn_composite_name_append_comp( )** prepends and appends a single component to the given composite name, respectively. These operations invalidate any iteration marker the client holds for that object. **fn_composite_name_insert_comp( )** inserts a single component before *iter_pos* to the given composite name and sets *iter_pos* to be immediately after the component just inserted. **fn_composite_name_delete_comp( )** deletes the component located before *iter_pos* from the given composite name and sets *iter_pos* back one component.

The functions **fn_composite_name_prepend_name( ), fn_composite_name_append_name( )** and **fn_composite_name_insert_name( )** perform the same update functions as their *_comp* counterparts, respectively, except that multiple components are being added, rather than single components. For **fn_composite_name_insert_name( )** sets *iter_pos* to be immediately after the name just added.

RETURN VALUE

The functions **fn_composite_name_is_empty( ), fn_composite_name_is_equal( ), fn_composite_name_is_suffix( )** and **fn_composite_name_is_prefix( )** return **1** if the test indicated is true; **0** otherwise.

The update functions **fn_composite_name_prepend_comp( ), fn_composite_name_append_comp( ), fn_composite_name_insert_comp( ), fn_composite_name_delete_comp( )** and their *_name* counterparts return **1** if the update was successful; **0** otherwise.

If a function is expected to return a pointer to an object, a null pointer (**0**) is returned if the function fails.

ERRORS

Code set mismatches that occur during the composition of the string form or during comparisons of composite names are resolved in an implementation-dependent way. **fn_string_from_composite_name( ), fn_composite_name_is_equal( ), fn_composite_name_is_suffix( )** and **fn_composite_name_is_prefix( )** sets *status* to **FN_E_INCOMPATIBLE_CODE_SETS** for composite names whose components have code sets that are determined by the implementation to be incompatible.

SEE ALSO

FN_string_t(3N), **xfn**(3N)