NAME

FN_compound_name_t, fn_compound_name_from_syntax_attrs, fn_compound_name_get_syntax_attrs, fn_compound_name_destroy, fn_string_from_compound_name, fn_compound_name_copy, fn_compound_name_assign, fn_compound_name_count, fn_compound_name_first, fn_compound_name_next, fn_compound_name_prev, fn_compound_name_last, fn_compound_name_prefix, fn_compound_name_suffix, fn_compound_name_is_empty, fn_compound_name_is_equal, fn_compound_name_is_prefix, fn_compound_name_is_suffix, fn_compound_name_prepend_comp, fn_compound_name_append_comp, fn_compound_name_insert_comp, fn_compound_name_delete_comp, fn_compound_name_delete_all – an XFN compound name

SYNOPSIS

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

#include <xfn/xfn.h>

FN_compound_name_t ∗**fn_compound_name_from_syntax_attrs(**
const FN_attrset_t ∗*aset*, **const FN_string_t** ∗*name*, **FN_status_t** ∗*status***);**

FN_attrset_t ∗**fn_compound_name_get_syntax_attrs(**
const FN_compound_name_t ∗*name***);**

void fn_compound_name_destroy(FN_compound_name_t ∗*name***);**

FN_string_t ∗**fn_string_from_compound_name(const FN_compound_name_t** ∗*name***);**

FN_compound_name_t ∗**fn_compound_name_copy(**
const FN_compound_name_t ∗*name***);**

FN_compound_name_t ∗**fn_compound_name_assign(FN_compound_name_t** ∗*dst*,
const FN_compound_name_t ∗*src***);**

unsigned int fn_compound_name_count(const FN_compound_name_t ∗*name***);**

const FN_string_t ∗**fn_compound_name_first(const FN_compound_name_t** ∗*name*,
void ∗∗*iter_pos***);**

const FN_string_t ∗**fn_compound_name_next(const FN_compound_name_t** ∗*name*,
void ∗∗*iter_pos***);**

const FN_string_t ∗**fn_compound_name_prev(const FN_compound_name_t** ∗*name*,
void ∗∗*iter_pos***);**

const FN_string_t ∗**fn_compound_name_last(const FN_compound_name_t** ∗*name*,
void ∗∗*iter_pos***);**

FN_compound_name_t ∗**fn_compound_name_prefix(**
const FN_compound_name_t ∗*name*, **const void** ∗*iter_pos***);**

FN_compound_name_t ∗**fn_compound_name_suffix(**
const FN_compound_name_t ∗*name*, **const void** ∗*iter_pos***);**

int fn_compound_name_is_empty(const FN_compound_name_t ∗*name***);**

int fn_compound_name_is_equal(" const FN_compound_name_t **name1*,
const FN_compound_name_t **name2*, **unsigned int** **status*);

int fn_compound_name_is_prefix(const FN_compound_name_t **name*,
const FN_compound_name_t **pre*, **void** ***iter_pos*, **unsigned int** **status*);

int fn_compound_name_is_suffix(const FN_compound_name_t **name*,
const FN_compound_name_t **suffix*, **void** ***iter_pos*, **unsigned int** **status*);

int fn_compound_name_prepend_comp(FN_compound_name_t **name*,
const FN_string_t **atomic_comp*, **unsigned int** **status*);

int fn_compound_name_append_comp(FN_compound_name_t **name*,
const FN_string_t **atomic_comp*, **unsigned int** **status*);

int fn_compound_name_insert_comp(FN_compound_name_t **name*, **void** ***iter_pos*,
const FN_string_t **atomic_comp*, **unsigned int** **status*);

int fn_compound_name_delete_comp(FN_compound_name_t **name*,
void ***iter_pos*);

int fn_compound_name_delete_all(FN_compound_name_t **name*);

MT-LEVEL

Safe.

DESCRIPTION

Most applications treat names as opaque data and hence, the majority of clients of the XFN interface will not need to parse names. Some applications, however, such as browsers, need to parse names. For these applications, XFN provides support in the form of the **FN_compound_name_t** object.

Each naming system in an XFN federation potentially has its own naming conventions. The **FN_compound_name_t** object has associated operations for applications to process compound names that conform to the XFN model of expressing compound name syntax. The XFN syntax model for compound names covers a large number of specific name syntaxes and is expressed in terms of syntax properties of the naming convention. See XFN_compound_syntax.

An **FN_compound_name_t** object is constructed by the operation **fn_compound_name_from_syntax_attrs ,** using a string name and an attribute set containing the "fn_syntax_type" (FN_ID_STRING syntax) attribute identifying the namespace syntax of the string name. The value "standard" (FN_ID_STRING syntax) in the "fn_syntax_type" specifies a syntax model that is by default supported by the **FN_compound_name_t** object. An implementation may support other syntax types instead of the XFN standard syntax model, in which case, the value of the "fn_syntax_type" attribute would be set to an implementation specific string. **fn_compound_name_get_syntax_attrs( )** returns an attribute set containing the syntax attributes that describes the given compound name. **fn_compound_name_destroy( )** releases the storage associated with the given compound name.

fn_string_from_compound_name( ) returns the string form of the given compound name. **fn_compound_name_copy( )** returns a copy of the given compound name. **fn_compound_name_assign( )** makes a copy of the compound name *src* and assigns it to *dst*, releasing any old contents of *dst*. A pointer to the object pointed to by *dst* is returned.

fn_compound_name_count( ) returns the number of atomic components in the given compound name.

The function **fn_compound_name_first( )** returns a handle to the **FN_string_t** that is the first atomic component in the compound 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_compound_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_compound_name_next( )** returns **0**. Similarly, **fn_compound_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_compound_name_prev( )** returns **0**. The function **fn_compound_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_compound_name_prev( )** can be used to step through trailing components of the name).

The **fn_compound_name_suffix( )** function returns a compound name consisting of a copy of those components following the supplied iteration marker. The function **fn_compound_name_prefix( )** returns a compound name consisting of those components that precede the iteration marker. Using these functions with an iteration marker that was not initialized using **fn_compound_name_first( ), fn_compound_name_last( ), fn_compound_name_is_prefix( ),** or **fn_compound_name_is_suffix( )** yields undefined and generally undesirable behavior.

The function **fn_compound_name_is_equal( ), fn_compound_name_is_prefix( ), fn_compound_name_is_suffix( ),** tests for equality between compound names or between parts of compound names. For these functions, equality is defined as name equivalence. A name's syntactic property, such as case-insensitivity, is taken into account by these functions.

The function **fn_compound_name_is_prefix( )** tests if one compound 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_compound_name_suffix( )** will return the remainder of the name.) Otherwise it returns **0** and value of the iteration marker is undefined. The function **fn_compound_name_is_suffix( )** is similar. It tests if a one compound name is a suffix of another. If so it returns **1** and sets the iteration marker immediately preceding the suffix.

fn_compound_name_prepend_comp( ) and **fn_compound_name_append_comp( )** prepends and appends a single atomic component to the given compound name, respectively. These operations invalidates any interation marker the client holds for that object. **fn_compound_name_insert_comp( )** inserts an atomic component before *iter_pos* to the given compound name and sets *iter_pos* to be immediately after the component just inserted. **fn_compound_name_delete_comp( )** deletes the atomic component located before *iter_pos* from the given compound name and sets *iter_pos* back one component. **fn_compound_name_delete_all( )** deletes all the atomic components from *name.*

RETURN VALUE

The functions **fn_compound_name_is_empty( ), fn_compound_name_is_equal( ), fn_compound_name_is_suffix( )** and **fn_compound_name_is_prefix( )** return **1** if the test indicated is true; **0** otherwise.

The update functions **fn_compound_name_prepend_comp( ), fn_compound_name_append_comp( ), fn_compound_name_insert_comp( ), fn_compound_name_delete_comp( )** and **fn_compound_name_delete_all( )** 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

When the function **fn_compound_name_from_syntax_attrs( )** fails, it returns in *status* a status code. The possible status codes are:

FN_E_ILLEGAL_NAME
The name supplied to the operation was not a well- formed XFN compound name, or one of the component names was not well-formed according to the syntax of the naming system(s) involved in its resolution.

FN_E_INCOMPATIBLE_CODE_SETS
The code set of the given string is incompatible with that supported by the compound name.

FN_E_INVALID_SYNTAX_ATTRS
The syntax attributes supplied are invalid or insufficient to fully specify the syntax.

FN_E_SYNTAX_NOT_SUPPORTED
The syntax type specified is not supported.

fn_compound_name_is_equal( ), fn_compound_name_is_suffix( ), fn_compound_name_is_prefix( ), fn_compound_name_prepend_comp( ), fn_compound_name_append_comp( ) and **fn_compound_name_insert_comp ( )** may **FN_E_INCOMPATIBLE_CODE_SETS** return in *status* when the code set of the given string is incompatible with that of the compound name.

SEE ALSO

FN_attribute_t(3N), **FN_attrset_t**(3N), **FN_composite_name_t**(3N), **FN_status_t**(3N), **FN_string_t**(3N), **fn_ctx_get_syntax_attrs**(3N), **xfn_compound_names**(3N), **xfn**(3N)