NAME

FN_ref_t, fn_ref_create, fn_ref_destroy, fn_ref_copy, fn_ref_assign, fn_ref_type, fn_ref_addrcount, fn_ref_first, fn_ref_next, fn_ref_prepend_addr, fn_ref_append_addr, fn_ref_insert_addr, fn_ref_delete_addr, fn_ref_delete_all, fn_ref_create_link, fn_ref_is_link, fn_ref_link_name, fn_ref_description – an XFN reference

SYNOPSIS

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

#include <xfn/xfn.h>

FN_ref_t ∗**fn_ref_create(const FN_identifier_t** ∗*ref_type*);

void fn_ref_destroy(FN_ref_t ∗*ref*);

FN_ref_t ∗**fn_ref_copy(const FN_ref_t** ∗*ref*);

FN_ref_t ∗**fn_ref_assign(FN_ref_t** ∗*dst*, **const FN_ref_t** ∗*src*);

const FN_identifier_t ∗**fn_ref_type(const FN_ref_t** ∗*ref*);

unsigned int fn_ref_addrcount(const FN_ref_t ∗*ref*);

const FN_ref_addr_t ∗**fn_ref_first(const FN_ref_t** ∗*ref*, **void** ∗∗*iter_pos*);

const FN_ref_addr_t ∗**fn_ref_next(const FN_ref_t** ∗*ref*, **void** ∗∗*iter_pos*);

int fn_ref_prepend_addr(FN_ref_t ∗*ref*, **const FN_ref_addr_t** ∗*addr*);

int fn_ref_append_addr(FN_ref_t ∗*ref*, **const FN_ref_addr_t** ∗*addr*);

int fn_ref_insert_addr(FN_ref_t ∗*ref*, **void** ∗∗*iter_pos*, **const FN_ref_addr_t** ∗*addr*);

int fn_ref_delete_addr(FN_ref_t ∗*ref*, **void** ∗∗*iter_pos*);

int fn_ref_delete_all(FN_ref_t ∗*ref*);

FN_ref_t ∗**fn_ref_create_link(const FN_composite_name_t** ∗*link_name*);

int fn_ref_is_link(const FN_ref_t ∗*ref*);

FN_composite_name_t ∗**fn_ref_link_name(const FN_ref_t** ∗*link_ref*);

FN_string_t ∗**fn_ref_description(const FN_ref_t** ∗*ref*, **unsigned int** *detail*,
unsigned int ∗*more_detail*);

MT-LEVEL

Safe.

DESCRIPTION

An XFN reference is represented by the type **FN_ref_t**. An object of this type contains a reference type and a list of addresses. The ordering in this list at the time of binding might not be preserved when the reference is returned upon lookup.

The reference type is represented by an object of type **FN_identifier_t**. The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.

Each address is represented by an object of type **FN_ref_addr_t**.

fn_ref_create( ) creates a reference with no address, using *ref_type* as its reference type. Addresses can be later added to the reference using the functions described below. **fn_ref_destroy( )** releases the storage associated with *ref*. **fn_ref_copy( )** creates a copy of *ref* and returns it. **fn_ref_assign( )** creates a copy of *src* and assigns it to *dst*, releasing any

old contents of *dst*. A pointer to the same object as *dst* is returned.

fn_ref_addrcount( ) returns the number of addresses in the reference *ref*.

fn_ref_first( ) returns the first address in *ref* and sets *iter_pos* to be after the address. It returns **0** if there is no address in the list. **fn_ref_next( )** returns the address following *iter_pos* in *ref* and sets *iter_pos* to be after the address. If the iteration marker *iter_pos* is at the end of the sequence, **fn_ref_next( )** returns **0**.

fn_ref_prepend_addr( ) adds *addr* to the front of the list of addresses in *ref*.
fn_ref_append_addr( ) adds *addr* to the end of the list of addresses in *ref*.
fn_ref_insert_addr( ) adds *addr* to *ref* before *iter_pos* and sets *iter_pos* to be immediately after the new reference added. **fn_ref_delete_addr( )** deletes the address located before *iter_pos* in the list of addresses in *ref* and sets *iter_pos* back one address.
fn_ref_delete_all( ) deletes all addresses in *ref*.

fn_ref_create_link( ) creates a reference using the given composite name *link_name* as an address. The reference type of this reference is defined in **fn_ref_is_link( )** tests if *ref* is a link. It returns **1** if it is; **0** if it is not. **fn_ref_link_name( )** returns the composite name stored in a link reference. It returns **0** if *link_ref* is not a link.

fn_ref_description( ) returns a string description of the given reference. It takes as argument an integer, *detail*, and a pointer to an integer *more_detail*. *detail* specifies the level of detail for which the description should be generated; the higher the number, the more detail is to be provided. If *more_detail* is zero, it is ignored. If *more_detail* is non-zero, it is set by the description operation to indicate the next level of detail available, beyond that specified by *detail*. If no higher level of detail is available, *more_detail* is set to *detail*.

RETURN VALUE

The operations **fn_ref_prepend_addr( ), fn_ref_append_addr( ), fn_ref_insert_addr( ), fn_ref_delete_addr( )** and **fn_ref_delete_all( )** return **1** if the operation succeeds, **0** if the operation fails.

APPLICATION USAGE

The reference type is intended to identify the class of object referenced. XFN does not dictate the precise use of this.

Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons, such as the object offering interfaces over more than one communication mechanism.

The client must interpret the contents of a reference based on the type of the addresses and the type of the reference. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

Manipulation of references using the operations described in this manual page does not affect their representation in the underlying naming system. Changes to references in the underlying naming system can only be effected through the use of the interfaces described in **FN_ctx_t**(3N).

SEE ALSO

FN_composite_name_t(3N), **FN_ctx_t**(3N), **FN_identifier_t**(3N), **FN_string_t**(3N), **FN_ref_addr_t**(3N), **FN_string_t**(3N), **fn_ctx_lookup**(3N), **fn_ctx_lookup_link**(3N), **xfn_links**(3N), **xfn**(3N)