XFN Programming Examples 12

This chapter presents self-contained executable programs for a namespace browser and a printer client and server.

12

Namespace Browser Example

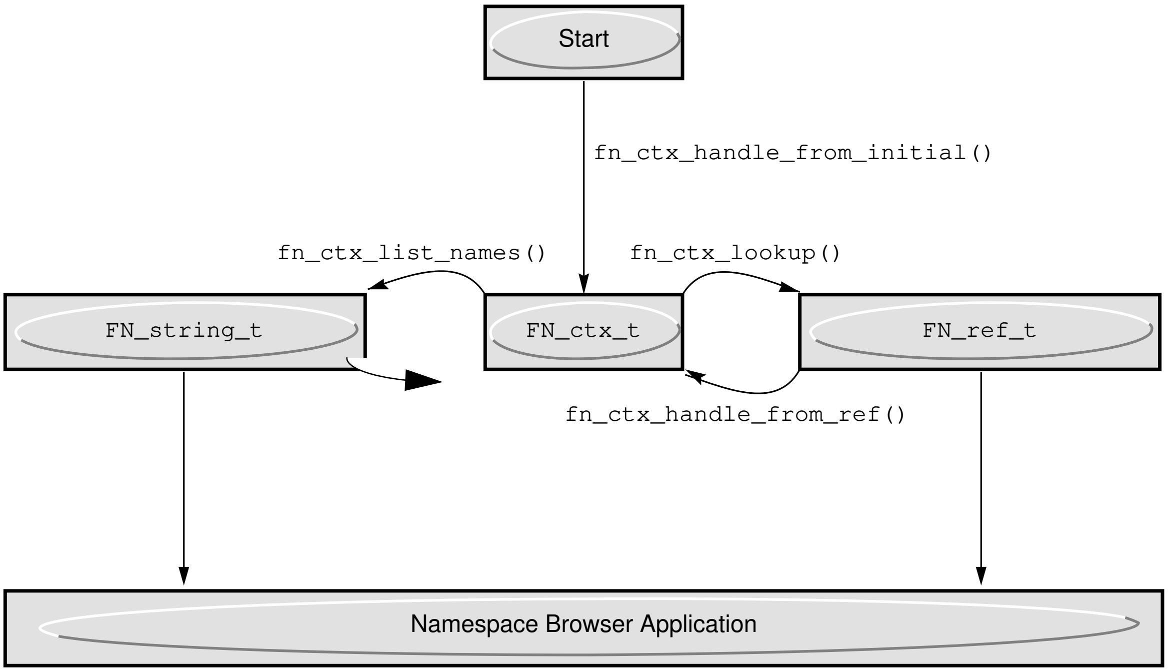

Figure 12-1 Diagram of `fnbrowse` Program

The first example is a browser that lists all names that it finds in the namespace. When the program is invoked, the browser is set at the initial context or the composite name given on the command line.

Figure 12-1 illustrates the XFN APIs that are used by the browser application.

See "Commands" on page 175" and "Sample Output" on page 176".

Code Example 12-1 `fnbrowse` Source Code

```
*
* fnbrowse.c -- FNS namespace browser.
*
* To keep this example program relatively short, limited error
* checking is done.
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <xfn/xfn.h>


#define LINELEN 128 /* maximum length of input line */

typedef enum {CMD_DOWN, CMD_UP, CMD_LIST, CMD_SHOW, CMD_QUIT}
command;

FN_status_t *status;


/* Look up a context named relative to the initial context. */
FN_ctx_t *lookup(const FN_composite_name_t *name);

/* Set the browser's focus to the given context. */
void browse(FN_ctx_t *ctx);

/* Set the browser's focus to a subcontext of the given context. */
void cmd_down(FN_ctx_t *ctx, const FN_composite_name_t *child);

/* Print the names bound within a context. */
void cmd_list(FN_ctx_t *ctx);

/*
 * Print a description of the reference bound to "child" in the given
 * context or, if "child" is the empty string, the reference of the
 * context itself.
 */
void cmd_show(FN_ctx_t *ctx, const FN_composite_name_t *child);

/*
 * Read and parse the next command typed by the user. If the command
 * has an argument, set *argp to point to the argument.
 */
command read_command(FN_string_t **argp);

/* Print an error message and the status description */
void error(const char *msg);
```

```
int
main(int argc, char *argv[])
{
    FN_string_t *arg;

    switch (argc) {
    case 1:
        arg = fn_string_create();
        break;
    case 2:
        arg = fn_string_from_str((unsigned char *)argv[1]);
        break;
    default:
        fprintf(stderr, "Usage: %s [<composite_name>]\n", argv[0]);
        return (1);
    }
    status = fn_status_create();
    browse(lookup(fn_composite_name_from_string(arg)));
    return (0);
}

FN_ctx_t *
lookup(const FN_composite_name_t *name)
{
    FN_ctx_t *ctx;
    FN_ref_t *ref;
    ctx = fn_ctx_handle_from_initial(status);
    if (ctx == NULL) {
        error("Could not construct initial context");
        exit(1);
    }
    if (fn_composite_name_is_empty(name)) {
        return (ctx);
    }
    ref = fn_ctx_lookup(ctx, name, status);
    fn_ctx_handle_destroy(ctx);
    if (ref == NULL) {
        error("Lookup failed");
        exit(1);
    }
```

```
    ctx = fn_ctx_handle_from_ref(ref, status);
    fn_ref_destroy(ref);
    if (ctx == NULL) {
        error("Could not construct context handle");
        exit(1);
    }
    return (ctx);
}


void
browse(FN_ctx_t *ctx)
{
    FN_string_t *arg;
    FN_composite_name_t*child;

    while (1) {
        switch (read_command(&arg)) {
        case CMD_DOWN:
            child = fn_composite_name_from_string(arg);
            fn_string_destroy(arg);
            cmd_down(ctx, child);
            fn_composite_name_destroy(child);
            break;
        case CMD_UP:
            return;
        case CMD_LIST:
            cmd_list(ctx);
            break;
        case CMD_SHOW:
            child = fn_composite_name_from_string(arg);
            fn_string_destroy(arg);
            cmd_show(ctx, child);
            fn_composite_name_destroy(child);
            break;
        case CMD_QUIT:
            exit(0);
        }
    }
}

void
```

```
cmd_down(FN_ctx_t *ctx, const FN_composite_name_t *child)
{
    FN_ref_t *ref;
    FN_ctx_t *subctx;

    ref = fn_ctx_lookup(ctx, child, status);
    if (ref == NULL) {
        error("Lookup failed");
        return;
    }
    subctx = fn_ctx_handle_from_ref(ref, status);
    fn_ref_destroy(ref);
    if (subctx == NULL) {
        error("Could not construct context handle");
        return;
    }
    browse(subctx);
    fn_ctx_handle_destroy(subctx);
}


void
cmd_list(FN_ctx_t *ctx)
{
    FN_string_t *empty_string = fn_string_create();
    FN_composite_name_t *empty_name;
    FN_namelist_t *children;
    FN_string_t *child;
    unsigned int statcode;
    int has_children = 0;

    empty_name = fn_composite_name_from_string(empty_string);
    fn_string_destroy(empty_string);

    children = fn_ctx_list_names(ctx, empty_name, status);
    fn_composite_name_destroy(empty_name);

    if (children == NULL) {
        error("Could not list names");
        return;
    }
```

```
    while ((child = fn_namelist_next(children, status)) != NULL) {
        has_children = 1;
        printf("%s  ", fn_string_str(child, &statcode));
        fn_string_destroy(child);
    }
    if (has_children) {
        printf("\n");
    }
    fn_namelist_destroy(children, status);
}


void
cmd_show(FN_ctx_t *ctx, const FN_composite_name_t *child)
{
    FN_string_t *desc;
    FN_ref_t *ref;
    unsigned int statcode;

    ref = fn_ctx_lookup(ctx, child, status);
    if (ref == NULL) {
        error("Lookup failed");
        return;
    }

    desc = fn_ref_description(ref, 2, NULL);
    fn_ref_destroy(ref);
    if (desc != NULL) {
        printf("%s", fn_string_str(desc, &statcode));
        fn_string_destroy(desc);
    } else {
        printf("[No description]\n");
    }
}


command
read_command(FN_string_t **argp)
{
    char buf[LINELEN + 1];
    char *cmd;
    char *child;
```

```
    while (printf("\n> "), fflush(stdout), gets(buf) != NULL) {
        cmd = strtok(buf, " \t");
        if (cmd == NULL) {
            continue;
        }
        if (strcmp(cmd, "down") == 0) {
            child = strtok(NULL, " \t");
            if (child != NULL) {
                *argp =
                    fn_string_from_str((unsigned char *)child);
                return (CMD_DOWN);
            }
        }
        if (strcmp(cmd, "up") == 0) {
            return (CMD_UP);
        }
        if (strcmp(cmd, "list") == 0) {
            return (CMD_LIST);
        }
        if (strcmp(cmd, "show") == 0) {
            child = strtok(NULL, " \t");
            *argp = (child != NULL)
                ? fn_string_from_str((unsigned char *)child)
                : fn_string_create();
            return (CMD_SHOW);
        }
        if (strcmp(cmd, "quit") == 0) {
            return (CMD_QUIT);
        }
        fprintf(stderr, "Valid commands are:  "
            "down <child>, up, list, show [<child>], quit\n");
    }
    return (CMD_QUIT);/* EOF */
}


void
error(const char *msg)
{
    FN_string_t *reason;
    unsigned int statcode;
    fprintf(stderr, "%s", msg);
    reason = fn_status_description(status, 0, NULL);
    if (reason != NULL) {
        fprintf(stderr, ": %s",
```

```
            (const char *)fn_string_str(reason, &statcode));
        fn_string_destroy(reason);
    }
    fprintf(stderr, "\n");
}
```

Compiling and Executing Browser Example

To compile Code Example 12-1, type:

```
% cc -o fnbrowse fnbrowse.c -lxfn
```

To browse the namespace starting from the initial context, the program is invoked as

```
% fnbrowse
```

Or to browse a composite name and its descendents, type

```
% fnbrowse composite_name
```

Commands

The commands supported by the `fnbrowse` program are summarized in Table 12-1.

Table 12-1 Namespace Browser Commands

Command	Usage
`down` *child*	Sets the browser at the subcontext of the child
`up`	Sets the browser at one level higher than the current context
`list`	Lists the names bound within the current context
`show`	Prints the reference of the current context
`show` *child*	Prints the reference of the current context's child
`quit`	Exits the browser

Sample Output

Sample output for navigating the entire namespace is displayed here.

Note the following:

- The first `list` command shows the initial context bindings.
- The `fnbrowse` program lists all names it finds in the namespace, including names with underscores. These names are explained in "Initial Context Bindings for Naming Within the Enterprise" on page 54.
- The three dots (`...`) represent the global namespace.

```
% fnbrowse
> list
_myorgunit  ...  _myself  thishost  myself  _orgunit  _host
_thisens  myens  thisens  org  orgunit  thisuser  _thishost
myorgunit  _user  thisorgunit  host  _thisorgunit  _myens  user
```

Navigating the namespace is accomplished with the `up` and `down` commands. In the following output, the `down` command brings the focus of the browser to the enterprise root of the namespace, `thisens` (can also be `myens`). The `show` command displays information about the reference and address type for `thisens`.

```
> down thisens
> show
Reference type: onc_fn_enterprise
Address type: on_fn_nisplus
  length: 20
  context type: enterprise root
  representation: normal
  version: 0
  internal name: eng.wiz.com

> up
> down thisorgunit
```

Continuing with the example, this `list` command shows the contexts for `thisorgunit`.

```
> list
service  _fs  _host  _service  _site  site  _user  host  fs  user

> down usr
Lookup failed: Name Not Found: 'usr'

> down service
> list
printer

> down printer
```

The `list` command shows the printer names that are bound in the `printer` context. The `show` command displays the reference for the child, `colorful`.

```
> list
celeste  _default  color  colorful  quartz  nuttree  puffin

> show colorful
printer
Reference type: onc_printers
Address type: onc_printers_bsdaddr
  length: 12
  data: 0x00 0x00 0x00 0x08 0x62 0x6c 0x61 0x63 0x6b 0x63
....blackc 0x61 0x74 at

> down colorful
Could not construct context handle: No Supported Address
> quit
%
```

12

Printer Programming Example

Printer client and server software can take advantage of FNS to advertise and to browse the printers available with respect to organizations, sites, users and hosts. The APIs used by the server and the client are XFN APIs, thereby ensuring that the application will be portable across the different naming services used for storing printer bindings.

The programming example in this section shows how printer clients and servers obtain and store printer bindings. Users can then make use of the FNS commands, `fnlist` and `fnlookup`, to browse the `printer` context.

For example, use `fnlist` to look at the user `printer` context for `jsmith`:

```
% fnlist user/jsmith/service/printer
celeste
lp
_default
myprinter
```

Similarly, you can look at the organization’s printers:

```
% fnlist org/wiz.com/service/printer
sales_printer
mktg_printer
eng_printer
```

Alternatively, you can type

```
% fnlist thisorgunit/service/printer
```

You can look at the printers at a specific site, for example, the printers in the `MTV` site:

```
% fnlist thisorgunit/site/MTV/service/printer
b1_printer
b2_printer
```

Client

The scenario for Code Example 12-2 is a user who would like to print to a printer named `colorful` in his organization's context, `thisorgunit/service/printer/colorful`. The example printer client illustrates how the bindings for a specific printer are obtained.

The variable *printer_binding* contains the reference (the binding information) of the named printer. Using the binding information, the printer client can connect to the server and send the printer request. Note that the `fn_ctx_lookup()` function can be replaced by `fn_ctx_list_name()` or `fn_ctx_list_bindings()` to list all the names and their bindings.

Code Example 12-2 Printer Client

```
#include <stdio.h>
#include <xfn/xfn.h>
#include <string.h>
#include <stdlib.h>

/* Routine to obtain the address of a specific printer */

/* This routine takes the printer name and the address type
   as the input arguments and returns the address of the requested
   printer */

char *

get_address_of_printer(const char *printer_name,
    const char *address_type)

{

    /* Variable list */
    FN_string_t *printer_name_string;
    FN_composite_name_t *printer_name_comp;
    FN_status_t *status;
    FN_ctx_t *initial_context;
    FN_ref_t *printer_ref;
    const FN_identifier_t *addr_id;
    const FN_ref_addr_t *address;
    char *addr_data; /* Return value */
    void *ip;
    size_t address_type_len, addr_len;

    /* Convert the printer name to a composite name */
    printer_name_string =
        fn_string_from_str((unsigned char *) printer_name);
```

```
printer_name_comp =
    fn_composite_name_from_string(printer_name_string);
fn_string_destroy(printer_name_string);

/* Get the initial context */
status = fn_status_create();
initial_context = fn_ctx_handle_from_initial(status);
/* Check status for any error messages */
if (!fn_status_is_success(status)){
    fprintf(stderr, "Unable to obtain the initial context\n");
    return (0);
}
/* Perform a lookup for the printer name */
printer_ref = fn_ctx_lookup(initial_context,
    printer_name_comp, status);
/* Check status for any error messages */
if (!fn_status_is_success(status)){
    fprintf(stderr, "Lookup failed on: %s\n",
        printer_name);
    return (0);
}
fn_ctx_handle_destroy(initial_context);
fn_composite_name_destroy(printer_name_comp);
address_type_len = strlen(address_type);

/* Obtain the requested address from the address type */
for (address = fn_ref_first(printer_ref, &ip);
    address != NULL;
    address = fn_ref_next(printer_ref, &ip)) {
        addr_id = fn_ref_addr_type(address);
        if (addr_id->length == address_type_len &&
            strncmp(address_type, (char *)addr_id->contents,
                    address_type_len) == 0)

                break;
}
if (address == NULL)
    return (0);

addr_len = fn_ref_addr_length(address);
addr_data = (char *)(malloc(addr_len + 1));
strncpy(addr_data,(char*)(fn_ref_addr_data(address)),
        addr_len);
addr_data[addr_len] = '\0';
```

```
    fn_ref_destroy(printer_ref);
    return (addr_data);
}
```

Calling the Printer Client Function

The following code could be used to call the `get_address_of_printer()` routine shown above.

```
char* printer_server;

printer_server = get_address_of_printer(
    "thisorgunit/service/printer/colorful",
    "onc_bsdaddr");
```

Server

Using the XFN APIs, print servers can advertise their services. Code Example 12-3 illustrates a host, `labpc`, that would like to advertise the binding for the color printer `colorful`. The FNS name for this printer is `thisorgunit/service/printer/colorful`.

The main tasks are to obtain the composite name for the printer name, to generate the binding (reference) for the printer, and to bind the name and references to the FNS namespace.

Code Example 12-3 Printer Server

```
#include <stdio.h>
#include <xfn/xfn.h>
#include <string.h>

/* Routine to export the printer binding to the FNS name space */

/* This routine takes the printer name along with its reference type,
   address type, and address. Returns the status. */

int
export_printer_to_fns(const char *printer_name,
              const char *reference_type,
              const char *address_type,
              const char *address_data)
{
    /* Variable list */
```

```
int return_status;
FN_string_t *printer_name_string;
FN_composite_name_t *printer_name_comp;
FN_identifier_t ref_id, addr_id;
FN_status_t *status;
FN_ref_t *printer_ref;
FN_ref_addr_t *address;
FN_ctx_t *initial_context;

/* Obtain the initial context */
status = fn_status_create();
initial_context = fn_ctx_handle_from_initial(status);
/* Check status for any error messages */
if ((return_status = fn_status_code(status)) != FN_SUCCESS) {
    fprintf(stderr, "Unable to obtain the initial context\n");
    return (return_status);
}

/* Construct the composite name for the printer name */
printer_name_string =
    fn_string_from_str((unsigned char *) printer_name);
printer_name_comp =
    fn_composite_name_from_string(printer_name_string);
fn_string_destroy(printer_name_string);

/* Construct the printer address */
addr_id.format = FN_ID_STRING;
addr_id.length = strlen(address_type);
addr_id.contents = (void *) address_type;
address = fn_ref_addr_create(&addr_id,
    strlen(address_data), (const void *) address_data);

/* Construct the printer reference */
ref_id.format = FN_ID_STRING;
ref_id.length = strlen(reference_type);
ref_id.contents = (void *) reference_type;
printer_ref = fn_ref_create(&ref_id);

/* Add the printer address to the printer reference */
fn_ref_append_addr(printer_ref, address);

/* Bind the reference to the context */
```

```
    fn_ctx_bind(initial_context, printer_name_comp, printer_ref, 0,
            status);

    /* Check the error status and return */
    return_status = fn_status_code(status);
    fn_composite_name_destroy(printer_name_comp);
    fn_ref_addr_destroy(address);
    fn_ref_destroy(printer_ref);
    fn_status_destroy(status);
    fn_ctx_handle_destroy(initial_context);
    return (return_status);
}
```

Calling the Printer Server Function

The following code could be used to call the `export_printer_to_fns` routine shown above.

```
export_printer_to_fns(
    "thisorgunit/service/printer/colorful",
    "onc_printers",
    "onc_bsdaddr",
    "labpc");
```