File Exchange Client Library

Bruce R. Lewis

March 12, 1990

Contents

1 Overview **2**

2 Opening a File Exchange **2**
2.1 Using fx_open() and fx_close() 2
2.2 Overriding the Default Server Path 3
2.3 Error Handling 3

3 Creating a New File Exchange **4**
3.1 Using fx_create() 4
3.2 Overview of Access Control Lists 4
3.3 Using fx_acl_add() and fx_acl_del() 4
3.4 Using fx_acl_list() 4

4 Sending a Paper **5**
4.1 A Simple Example using paper_clear(), fx_send_file() and fx_send() 5
4.2 Setting Paper Properties 7

5 Getting a List of Papers **7**
5.1 Using fx_list() 7

6 What To Do With a List of Papers **8**
6.1 Using paper_copy() 8
6.2 Using fx_retrieve_file() and fx_retrieve() 8
6.3 Using fx_copy() and fx_move() 8
6.4 Using fx_delete() 9

1 Overview

The File Exchange Client Library helps C programmers write applications which imitate the paper handling done in a classroom. A variety of paper types is supported.

EXCHANGE papers can be written and read by anyone.[1] An EXCHANGE paper can be removed by its author or by any grader.

TURNEDIN papers can be written by anyone but can be read only by graders and the papers' authors. The various types distinguish the stages in grading.

TAKEN is the paper type to which papers are moved when a grader decides to grade them. This avoids a condition of more than one grader taking the same paper at the same time. Any grader can also move a paper back to type TURNEDIN upon deciding that it was taken by mistake, or that the grader who took it will never grade it.

GRADED papers are written by graders. (Usually they are papers previously turned in by someone else, which the grader has modified to add comments and possibly a grade). A graded paper can be read by graders and by the paper's author.

PICKEDUP is the paper type to which the orignal author moves a GRADED paper to show that it's now ok for a grader to remove the paper from the server (assuming the grader doesn't want to keep it for archival purposes.)

HANDOUT papers can be written by graders and read by anyone.

TEACHERS_ARCHIVE and TEACHERS_HANDOUT papers can be written and read only by graders.

This document describes the library functions needed to handle these papers by communicating with the file exchange server.

2 Opening a File Exchange

A file exchange is a distributed database of papers on some number of servers, but this does not affect client programs; the distributed nature of the file exchange is hidden below the abstraction barrier of the FX structure, which is handled much like the Unix FILE structure.

2.1 Using fx_open() and fx_close()

```
#include <fxcl.h>

FX *fx_open(s, codep)
```

[1] For a more precise idea of who "anyone" is and who "graders" are, see the section on access control lists.

```
    char *s;
    long *codep;

void fx_close(fxp)
    FX *fxp;
```

Here `s` is the name of the file exchange, and `codep` (if non-null) will be filled in with an error code. There are three possibilities for the error status of `fx_open()`.

1. A non-null pointer is returned and `*codep` is zero, meaning no error was encountered.
2. A non-null pointer is returned and `*codep` is non-zero, meaning some error was encountered that may limit access to the file exchange.
3. A null pointer is returned and `*codep` is non-zero, meaning the file exchange could not be opened.

Normally, a file exchange should not be held open while waiting for user input, because each open file exchange uses up at least one file descriptor at the server. If more than fifty users might be using a server at the same time, it is imperative that the client program open the file exchange, do what needs to be done, and close the file exchange immediately.

NOTE: On some systems, closing a file exchange twice may cause subsequent calls to `malloc()` not to function correctly. To be sure this doesn't happen, follow this convention:

```
if (fxp) { fx_close(fxp); fxp = NULL; }
```

2.2 Overriding the Default Server Path

Normally, `fx_open()` uses hesiod information[2] to find which server machines it should try to contact. A user can override this by setting the environment variable FXPATH to host1:host2:etc.

2.3 Error Handling

The `fx_open()` function may have to try several servers before it can actually connect to one. In the worst case, several timeouts may occur, so that the user does not know why a program is taking so long. To combat this situation, `fx_open()`, when it fails to connect to a host, will print an error message if there are still hosts to try to contact. This behavior can be modified by changing `fx_open_error_hook`.

```
void (*fx_open_error_hook)() = fx_open_perror;

void fx_open_perror(fxp, code)
    FX *fxp;
```

[2]In the Athena computing environment, one types `hesinfo turnin sloc` to see the default server path.

```
     long code;
{
  if (code)
    com_err(fxp->host, code, "(%s)", fxp->name);
}
```

Client programs may use alternate error reporting routines with the same parameters as `fx_open_perror()`. Such routines may make use of `fxp->host`, the machine for which the error occurred, and `fxp->name`, the file exchange for which the error occurred. For details on `com_err`, see the MIT Student Information Processing Board's document, *A Common Error Description Library for UNIX*.

3 Creating a New File Exchange

Not all users can create file exchanges, only those in the server's global access-control list. One authenticates to the server upon opening a file exchange. A nameless file exchange ("") can always be opened for the purpose of authentication before creating new file exchanges.

3.1 Using fx_create()

```
long fx_create(fxp, name)
     FX *fxp;
     char *name;
```

Here `fxp` is the file exchange already opened for authentication, and `name` is the name of the file exchange to be created. The error code returned can be diagnosed by `com_err`.

3.2 Overview of Access Control Lists

The first thing to do after creating a file exchange is to adjust the access control lists `ACL_TURNIN`, `ACL_GRADER`, and `ACL_MAINT`. Initially only the creator of the course is in each of these lists. Everyone connected with the file exchange should be in the turnin list. Add "*" if all authenticated users in your kerberos realm should have access. (To give access to users in, say, the LCS.MIT.EDU, add "`*@LCS.MIT.EDU`".) Graders get the privileges described in the overview. Users in the maint list are allowed to change the access control lists.

3.3 Using fx_acl_add() and fx_acl_del()

```
long fx_acl_add(fxp, aclname, person)
     FX *fxp;
```

```
     char *aclname, *person;

long fx_acl_del(fxp, aclname, person)
     FX *fxp;
     char *aclname, *person;
```

Here `fxp` is the file exchange whose access control lists are to be changed, `aclname` is one of `ACL_TURNIN`, `ACL_GRADER`, `ACL_MAINT`, and `person` is a username or kerberos principal.

3.4 Using fx_acl_list()

The contents of the access control lists may be retrieved from the server into a `stringlist_res` structure, which should be destroyed after use to free allocated memory.

```
typedef struct stringnode *stringlist;

struct stringnode {
        char *s;
        stringlist next;
};
typedef struct stringnode stringnode;

struct stringlist_res {
        long errno;
        union {
                stringlist list;
        } stringlist_res_u;
};
typedef struct stringlist_res stringlist_res;
```

Client programs are only concerned with the stringlist_res_u.list member of the structure, since errno will be returned by the `fx_acl_list()` function. It may seem awkward to have to deal with a pointer to such an unwieldy structure rather than just a simple stringlist, but this is the best way to allow the entire structure returned by the server to be destroyed at the discretion of the client program.

The structure is retrieved by `fx_acl_list()`, which returns error status, and destroyed by `fx_acl_list_destroy()`, which always succeeds.

The strings returned are the usernames of members of the access control list if those users are in the same kerberos realm as the user who retrieves the ACL. If there are members of the list from other realms, the full kerberos principal will be returned.

```
long fx_acl_list(fxp, aclname, ret)
     FX *fxp;
     char *aclname;
     stringlist_res **ret;

void fx_acl_list_destroy(list)
     stringlist_res **list;
```

4 Sending a Paper

4.1 A Simple Example using paper_clear(), fx_send_file() and fx_send()

The following program sends a file called "essay.dvi" to the file exchange given on the command line, and then sends a corresponding "essay.PS" without actually creating the file locally. It serves as an example of how easy it is to use paper_clear(), fx_send_file() and fx_send() to send files.

```
#include <stdio.h>
#include <fxcl.h>

main(argc, argv)
     int argc;
     char *argv[];
{
  FX *fxp;
  long code;
  Paper p;
  FILE *pipe;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s <exchange>\n", argv[0]);
    exit(1);
  }
  fxp = fx_open(argv[1], &code);
  if (!fxp) {
    com_err(argv[0], code, "while opening %s", argv[1]);
    exit(1);
  }
  /* Send the essay.dvi file */
  paper_clear(&p);
  code = fx_send_file(fxp, &p, "essay.dvi");
  if (code) {
    com_err(argv[0], code, "while sending essay.dvi");
    exit(1);
  }
  pipe = popen("dvi2ps -r essay.dvi", "r");
  if (!pipe) {
    fprintf(stderr, "%s: Could not run dvi2ps\n", argv[0]);
    exit(1);
  }
  /* Send the essay.PS file */
  p.filename = "essay.PS";
  code = fx_send(fxp, &p, pipe);
  if (code) {
    com_err(argv[0], code, "while sending essay.PS");
    exit(1);
  }
  fx_close(fxp);
  (void) pclose(pipe);
  exit(0);
}
```

4.2 Setting Paper Properties

Notice that in the preceding example the programmer was not obligated to specify the paper type; it defaults to EXCHANGE. Assignment defaults to 1. Filename defaults to the name of the local file for `fx_send_file()` and to "x" for `fx_send()`. Author defaults to the sender, but a grader may attribute the authorship of a paper to someone else. (The file exchange server will set authorship to the sender for everyone else.) In summary, the preceding program could have set

- `p.assignment` to an integer, the assignment number.
- `p.type` to any PaperType listed in the overview of this document.
- `p.filename` to any valid Unix base filename.
- `p.author` to any string (preferably username or kerberos principal).

5 Getting a List of Papers

5.1 Using fx_list()

Lists of papers may be retrieved from the server into a `Paperlist_res` structure, which should be destroyed after use to free allocated memory.

```
typedef struct PaperNode *Paperlist;

struct PaperNode {
        Paper p;
        Paperlist next;
};
typedef struct PaperNode PaperNode;

struct Paperlist_res {
        long errno;
        union {
                Paperlist list;
        } Paperlist_res_u;
};
typedef struct Paperlist_res Paperlist_res;
```

Client programs are only concerned with the Paperlist_res_u.list member of the structure, since errno will be returned by the `fx_list()` function. It may seem awkward to have to deal with a pointer to such an unwieldy structure rather than just a simple Paperlist, but this is the

best way to allow the entire structure returned by the server to be destroyed at the discretion of the client program.

The structure is retrieved by `fx_list()`, which returns error status, and destroyed by `fx_list_destroy()`, which always succeeds.

The paper pointer, pp, passed to `fx_list()` should first be cleared with `paper_clear()` (see the example program in a previous section), and then filled in with criteria for the list. For example, if you were only searching for papers associated with assignment 2, you would set `pp->assignment = 2`.

A user who is not in the grader list will not be able to list papers by other authors except for types EXCHANGE and HANDOUT.

```
long fx_list(fxp, pp, ret)
     FX *fxp;
     Paper *pp;
     Paperlist_res **ret;

void fx_list_destroy(plist)
     Paperlist_res **plist;
```

6 What To Do With a List of Papers

6.1 Using paper_copy()

Modifying the contents of a paper in the list returned by `fx_list()` may cause problems when you later destroy that list, it is better to copy into a new paper first.

```
paper_copy(src, dest)
     Paper *src, *dest;
```

Do not destroy the list until you are done with all Papers copied from nodes of the list.

6.2 Using fx_retrieve_file() and fx_retrieve()

These functions take the same types of parameters as `fx_send_file()` and `fx_send()`, but the paper pointer passed must be taken from the list returned by `fx_list()`.

6.3 Using fx_copy() and fx_move()

In each of the declarations below, src should come from the list returned by `fx_list()`, and dest should be a modified copy of src.

```
long fx_move(fxp, src, dest)
     FX *fxp;
     Paper *src, *dest;

long fx_copy(fxp, src, dest)
     FX *fxp;
     Paper *src, *dest;
```

The most frequent use of `fx_move()` is to change the PaperType during grading. A TURNEDIN file should be changed to TAKEN before a grader retrieves it for annotating. The annotated version should be sent with type GRADED. After retrieving the file, a student should change the status to PICKEDUP.

One possible use of `fx_copy()` would be to use a TURNEDIN file as an example for the whole class to look at, changing it to type HANDOUT.

6.4 Using fx_delete()

In the declaration below, p should come from the list returned by `fx_list()`.

```
long fx_delete(fxp, p)
     FX *fxp;
     Paper *p;
```