`diag/ndiag`: a list of diagnostics (and the number of diagnostics in the list); and,

`fti`: a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

2.8.3 Canceling Transfer

Either the initiator or the responder may cancel the transfer prior to orderly termination of the data transfer. The `FCancelRequest`, which corresponds to the F-CANCEL.REQUEST action, is used to initiate cancellation.

```
int     FCancelRequest (sd, action, sharedASE, diag, ndiag,
                        fti)
int     sd;
int     action;
PE      sharedASE;
struct FTAMdiagnostic diag[];
int     ndiag;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd`: the association-descriptor;

`action`: the action result;

`sharedASE`: shared ASE information;

`diag/ndiag`: a list of diagnostics (and the number of diagnostics in the list); and,

`fti`: a pointer to an `FTAMindication` structure, which is always updated.

If the call to `FCancelRequest` is successful, then this corresponds to the F-CANCEL.CONFIRMATION event, the `fti_type` element of the `fti` parameter is set to `FTI_CANCEL`, and the `fti_cancel` element contains the results.

Otherwise if the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

Upon receiving an F-CANCEL.INDICATION event, the user is required to generate the F-CANCEL.RESPONSE events using the `FCancelResponse` routine.

```
int     FCancelResponse (sd, action, sharedASE, diag, ndiag,
                         fti)
int     sd;
int     action;
PE      sharedASE;
struct FTAMdiagnostic diag[];
int     ndiag;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`action:` the action result;

`sharedASE:` shared ASE information;

`diag/ndiag:` a list of diagnostics (and the number of diagnostics in the list); and,

`fti:` a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

2.8.4 Terminating Transfer

After generating the F-DATA-END.REQUEST (or after receiving the F-DATA-END.RESPONSE) action, the initiator is required to perform the F-TRANSFER-END.REQUEST action using the `FTransEndRequest` routine.

```
int     FTransEndRequest (sd, sharedASE, fti)
int     sd;
PE sharedASE;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`sharedASE:` shared ASE information; and,

`fti:` a pointer to an `FTAMindication` structure, which is always updated.

If the call to `FTransEndRequest` is successful, then this corresponds to the F-TRANSFER-END.CONFIRMATION event, the `fti_type` element of the `fti` parameter is set to `FTI_TRANSEND`, and the `fti_transend` element contains the results.

Otherwise, if the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

Upon receiving an F-TRANSFER-END.INDICATION event, the responder is required to generate the F-TRANSFER-END.RESPONSE events using the `FTransEndResponse` routine.

```
int     FTransEndResponse (sd, action, sharedASE, diag, ndiag,
            fti)
int     sd;
int     action;
PE      sharedASE;
struct FTAMdiagnostic diag[];
int     ndiag;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`action:` the action result;

`sharedASE:` shared ASE information;

`diag/ndiag:` a list of diagnostics (and the number of diagnostics in the list); and,

`fti:` a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

2.9 Grouped Operations: File Management

The `FManageRequest` routine is used to issue a grouped management request consisting of:

- F-BEGIN-GROUP.REQUEST;
- F-SELECT.REQUEST or F-CREATE.REQUEST;
- optionally, F-READ-ATTRIB.REQUEST;
- optionally, F-CHANGE-ATTRIB.REQUEST;
- F-DESELECT.REQUEST or F-DELETE.REQUEST; and,
- F-END-GROUP.REQUEST.

The user initializes an `FTAMgroup` structure (described in tedious detail on page 44) and then invokes `FManageRequest`.

```
int     FManageRequest (sd, ftg, fti)
int     sd;
struct FTAMgroup *ftg;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`ftg:` a pointer to an `FTAMgroup` structure; and,

`fti:` a pointer to an `FTAMindication` structure, which is always updated.

If the call to `FManageRequest` is successful, then this corresponds to the appropriate .CONFIRMATION events, the `fti_type` element of the `fti` parameter is set to `FTI_MANAGEMENT`, and the `fti_group` element contains the results. Note that the data contained in the structure was allocated via *malloc*(3), and should be released with the `FTGFREE` macro when no longer needed.

Otherwise if the call fails, the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure.

Upon receiving an FTI_MANAGEMENT indication, containing the appropriate .INDICATION events, the responder is required to generate the appropriate .RESPONSE events using the FManageResponse routine.

```
int     FManageResponse (sd, ftg, fti)
int     sd;
struct FTAMgroup *ftg;
struct FTAMindication *fti;
```

The parameters to this procedure are:

sd: the association-descriptor;

ftg: a pointer to an FTAMgroup structure; and,

fti: a pointer to an FTAMindication structure, which is updated only if the call fails.

If the call fails, the FTAMabort structure contained in the FTAMindication parameter fti contains the reason for the failure.

2.10 Association Release

The FTerminateRequest routine is used to request the release of an association, and corresponds to an F-TERMINATE.REQUEST action.

```
int     FTerminateRequest (sd, sharedASE, ftr, fti)
int     sd;
PE      sharedASE;
struct FTAMrelease *ftr;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`sharedASE:` shared ASE information;

`ftr:` a pointer to an `FTAMrelease` structure, which is updated only if the call succeeds; and,

`fti:` a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call to `FTerminateRequest` is successful, then this corresponds to an F-TERMINATE.CONFIRMATION event, and it returns information in the `ftr` parameter, which is a pointer to an `FTAMrelease` structure.

```
struct FTAMrelease {
    PE     ftr_sharedASE;

    struct FTAMcharging ftr_charges;
};
```

The elements of this structure are:

`ftr_sharedASE:` shared ASE information; and,

`ftr_charges:` any charges.

Note that the data contained in the structure was allocated via *malloc*(3), and should be released with the `FTRFREE` macro when no longer needed. The `FTRFREE` macro behaves as if it was defined as:

```
void    FTRFREE (ftr)
struct FTAMrelease *ftr;
```

The macro frees only the data allocated by `FTerminateRequest`, and not the `FTAMrelease` structure itself. Further, `FTRFREE` should be called only if the call to the `FTerminateRequest` routine returned `OK`.

If the call to `FTerminateRequest` is successful, then the association has been released. Otherwise the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for failure.

Upon receiving an F-TERMINATE.INDICATION event, the user is required to generate an F-TERMINATE.RESPONSE action using the `FTerminateResponse` routine.

```
int     FTerminateResponse (sd, sharedASE, charging, fti)
int     sd;
PE      sharedASE;
struct FTAMcharging *charging;
struct FTAMindication  *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`sharedASE:` shared ASE information;

`charging:` charging information (if any); and,

`fti:` a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call to `FTerminateResponse` is successful, then the association has been released.

2.11 Association Abort

The `FUAbortRequest` routine is used to request the ungraceful release of an association, and corresponds to an F-U-ABORT.REQUEST action.

```
int     FUAbortRequest (sd, action, diag, ndiag, fti)
int     sd;
int     action;
struct FTAMdiagnostic diag[];
int     ndiag;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`action:` an action result;

`diag/ndiag:` a list of diagnostics (and the number of diagnostics in the list); and,

`fti:` a pointer to an `FTAMindication` structure, which is updated only if the call fails.

If the call to `FUAbortRequest` is successful, then the connection is immediately closed, and any data queued for the connection may be lost.

2.12 Error Conventions

All of the routines in this library return the manifest constant `NOTOK` on error, and also update the `fti` parameter given to the routine. The `fti_abort` element of the `FTAMindication` structure contains a `FTAMabort` structure detailing the reason for the failure. For each diagnostic present (typically only one) in the `FTAMabort` structure, the `ftd_identifier` element of each diagnostic can be given as a parameter to the procedure `FErrString` which returns a null-terminated diagnostic string.

```
char   *FErrString (c)
int     c;
```

2.13 Compiling and Loading

Programs using the *libftam*(3n) library should include `<isode/ftam.h>`. These programs should also be loaded with `-lftam` and `-lisode`.

2.14 An Example

Read Section 4.1 on page 80.

2.15 For Further Reading

The ISO specification for the International Standard on File Transfer, Access, and Management is found in [ISO88], which is a four-part document.

Chapter 3

The ISO Documents Database

The database *isodocuments* in the ISODE `ETCDIR` directory (usually the directory */usr/etc/*) contains a simple mapping between textual descriptions of FTAM document types and the various object identifiers which compose each document type.

The database itself is an ordinary ASCII text file containg information regarding the known FTAM document types on the host. Each line contains

- the entry number of the document type, a simple string;
- the document type, an object identifier;
- the constraint set, an object identifier;
- the abstract syntax, an object identifier; and,
- the transfer syntax, an object identifier

Blanks and/or tab characters are used to seperate items. However, double-quotes may be used to prevent separation for items containing embedded whitspace. The sharp character (‘`#`’) at the beginning of a line indicates a commentary line.

3.1 Accessing the Database

The *libftam*(3n) library contains the routines used to access the database. These routines ultimately manipulate an `isodocment` structure, which is the internal form.

```
struct isodocument {
    char   *id_entry;

    OID     id_type;
    OID     id_constraint;
    OID     id_abstract;
    OID     id_transfer;
};
```

The elements of this structure are:

`id_entry:` the entry number of the document type;

`id_type:` the document type;

`id_constraint:` the constraint set;

`id_abstract:` the abstract syntax; and,

`id_transfer:` the transfer syntax.

The routine `getisodocument` reads the next entry in the database, opening the database if necessary

```
struct isodocument *getisodocument ()
```

It returns the manifest constant `NULL` on error or end-of-file.

The routine `setisodocument` opens and rewinds the database.

```
int     setisodocument (f)
int     f;
```

The parameter to this procedure is:

`f:` the "stayopen" indicator, if non-zero, then the database will remain open over subsequent calls to the library.

The routine `endisodocument` closes the database.

```
int     endisodocument ()
```