The elements of this structure are:

`ftre_action`: the action result (.RESPONSE only);

`ftre_sharedASE`: shared ASE information; and,

`ftre_diags/ftre_ndiag`: any diagnostic information (and the number of diagnostics present, .RESPONSE only).

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

```
void    FTREFREE (ftre)
struct FTAMtransend *ftre;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMtransend` structure itself.

2.5.1 Asynchronous Event Handling

Thus far the discussion on event handling has been synchronous in nature. Some users of FTAM may wish an asynchronous interface. The `FSetIndications` routine is used to change the service associated with an association-descriptor to or from an asynchronous interface.

```
int     FSetIndications (sd, indication, fti)
int     sd;
int   (*indication) ();
struct FTAMindication  *fti;
```

The parameters to this procedure are:

`sd`: the association-descriptor;

`indication`: the address of an event-handler routine to be invoked when an indication occurs; and,

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

If the service is to be made asynchronous, then the event handler is specified, otherwise, if the service is to be made synchronous, then the handler should not be specified (use the manifest constant `NULLIFP`). The most likely reason for the call failing is `FS_GEN_WAITING`, which indicates that an event is waiting for the user.

When an event-handler is invoked, future invocations of the event-hander are blocked until it returns. The return value of the event-handler is ignored. Further, during the execution of a synchronous call to the library, the event-handler will be blocked from being invoked.

When an event occurs, the event-handler routine is invoked with two parameters:

```
(*indication) (sd, fti);
int     sd;
struct FTAMindication *fti;
```

The parameters are:

`sd:` the association-descriptor; and,

`fti:` a pointer to the `FTAMindication` structure containing the indication.

> **NOTE:** The *libftam*(3n) library uses the SIGEMT signal to provide these services. Programs using asynchronous association-descriptors should NOT use SIGEMT for other purposes.

2.5.2 Synchronous Event Multiplexing

A user of FTAM may wish to manage multiple association-descriptors simultaneously; the routine `FSelectMask` is provided for this purpose. This routine updates a file-descriptor mask and associated counter for use with `xselect`.

```
int     FSelectMask (sd, mask, nfds, fti)
int     sd;
fd_set *mask;
int    *nfds;
struct FTAMindication  *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`mask:` a pointer to a file-descriptor mask meaningful to `xselect`;

`nfds:` a pointer to an integer-valued location meaningful to `xselect`; and,

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

If the call is successful, then the `mask` and `nfds` parameters can be used as arguments to `xselect`. The most likely reason for the call failing is `FS_GEN_WAITING`, which indicates that an event is waiting for the user.

If `xselect` indicates that the association-descriptor is ready for reading, `FWaitRequest` should be called with the `secs` parameter equal to `OK`. If the network activity does not constitute an entire event for the user, then `FWaitRequest` will return `NOTOK` with error code `FS_PRO_TIMEOUT`.

2.5.3 Tracing

Users may wish to trace FTAM events as they are processed. The routine `FHookRequest` is used to set (or unset) the user-defined tracing routine.

```
int     FHookRequest (sd, tracing, fti)
int     sd;
int   (*tracing) ();
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`tracing:` the address of a tracing routine to be invoked when an FTAM event occurs; and,

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

If tracing is to be enabled, then the tracing routine is specified, otherwise, if tracing is to be turned off, the manifest constant `NULLIFP` should be used.

When an event occurs, the tracing routine is invoked with five parameters:

```
(*tracing) (sd, event, fpdu, pe, rw)
```

The parameters are:

`sd:` the association-descriptor;

`event:` the name of the underlying service primitive (association control or presentation) associated with the FTAM event;

`fpdu:` the name of the FTAM PDU, if any;

`pe:` a presentation element containing the FTAM PDU associated with the FTAM event (described in Chapter 5 of *Volume One*), if any; and,

`rw:` a flag saying whether the event is being initiated (zero) or being received (non-zero).

The return value of the tracing routine is ignored.

One pre-defined tracing routine is available, `FTraceHook` , which simply appends tracing information to the log determined by

```
LLog *ftam_log;
```

See Chapter 7 in *Volume Two* for all the details.

2.6 Grouped Operations: File Transfer

The `FBulkBeginRequest` routine is used to issue a grouped file transfer 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-OPEN.REQUEST; and,

- F-END-GROUP.REQUEST.

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

```
int     FBulkRequest (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 `FBulkBeginRequest` is successful, then this corresponds to the appropriate .CONFIRMATION events, the `fti_type` element of the `fti` parameter is set to `FTI_BULKBEGIN`, 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_BULKBEGIN` indication, containing the appropriate .INDICATION events, the responder is required to generate the appropriate .RESPONSE events using the `FBulkBeginResponse` routine.

```
int     FBulkBeginResponse (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.

The `FBulkEndRequest` routine is used to issue a grouped file transfer request consisting of:

- F-BEGIN-GROUP.REQUEST;
- F-CLOSE.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 `FBulkEndRequest`.

```
int     FBulkEndRequest (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 `FBulkEndRequest` is successful, then this corresponds to the appropriate .CONFIRMATION events, the `fti_type` element of the `fti` parameter is set to `FTI_BULKEND`, 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_BULKEND` indication, containing the appropriate .INDICATION events, the responder is required to generate the appropriate .RESPONSE events using the `FBulkEndResponse` routine.

```
int     FBulkEndResponse (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.

2.7 File Access

The `FAccessRequest` routine is equivalent to an F-LOCATE.REQUEST or F-ERASE.REQUEST action on the part of the user.

```
int     FAccessRequest (sd, operation, identity, fti)
int     sd;
int     operation;
struct FADUidentity *identity;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`operation:` the operation to be performed, one of the values shown in Table 2.3;

`identity:` the identity of the FADU to locate or erase; and,

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

If the call to FAccessRequest is successful, then this corresponds to the appropriate .CONFIRMATION event, the fti_type element of the fti parameter is set to FTI_ACCESS, and the fti_access element contains the results. Note that the data contained in the structure was allocated via *malloc*(3), and should be released with the FTACFREE 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 F-LOCATE.INDICATION or an F-ERASE.INDICATION event, the user is required to generate the appropriate .RESPONSE action using the FAccessResponse routine.

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

The parameters to this procedure are:

sd: the association-descriptor;

action: the action result;

identity: the identity of the FADU located (not present for the F-ERASE.RESPONSE action);

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 AccessResponse is successful, then the .RESPONSE event has been queued for sending to the initiator. Otherwise the FTAMabort structure contained in the FTAMindication parameter fti contains the reason for the failure.

2.8 Data Transfer

Once group file transfer request has been made, and after any file access requests, data transfer may occur. This occurs in three steps: the initiator requests reading or writing of an FADU, the source sends the data, the transfer is then either cancelled (by either side) or terminated by the initiator.

2.8.1 Read/Write

The `FReadWriteRequest` routine is equivalent to a F-READ.REQUEST or F-WRITE.REQUEST action on the part of the user.

```
int     FReadWriteRequest (sd, operation, identity, context,
                level, lock, fti)
int     sd;
int     operation;
struct FADUidentity *identity;
int     context,
        level,
        lock;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`operation:` the operation to be performed, one of the values shown in Table 2.4 on page 56;

`identity:` the identity of the FADU to be transferred;

`context/level:` the access context, one of the values shown in Table 2.5;

`lock:` lock FADU; 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.

2.8.2 Sending Data

The FADU is transmitted as a series of presentation elements. A group of presentation elements may be sent using the `FDataRequest` routine, which corresponds to the F-DATA.REQUEST action.

```
int     FDataRequest (sd, fadus, nfadu, fti)
int     sd;
PE      fadus[];
int     nfadu;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd:` the association-descriptor;

`fadus/nfadu:` the list of data elements (and the number of data elements which may not exceed the manifest constant `NPDATA` described in Chapter 2), consult Section 2.5 on page 55 for important information on the use of presentation context identifiers; 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.

When the last data element in the FADU has been sent, the F-DATA-END.REQUEST action should be performed using the `FDataEndRequest` routine.

```
int     FDataEndRequest (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:` the action result;