Technical Standard

FTAM High-Level API (XFTAM) Version 2

[This page intentionally left blank]

X/Open CAE Specification

FTAM High-Level API (XFTAM) Version 2

X/Open Company Ltd.

X/Open CAE Specification

FTAM High-Level API (XFTAM) Version 2

ISBN: 1-85912-092-X
X/Open Document Number: C415

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.org

Contents

List of Figures

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This XFTAM Version 2 CAE Specification supersedes the previously published XFTAM CAE Specification (C304, January 1994). It includes revisions to align it with the IEEE FTAM Standard which is itself based on the previously published XFTAM CAE Specification.

This Specification defines the X/Open File Transfer, Access and Management (XFTAM) Version 2 API, which is a programming interface to the OSI File Transfer, Access and Management protocol. The IEEE alignment includes support for *context-sensitive* mode of operation of the XFTAM API; this support was at X/Open Preliminary Specification status in the previous XFTAM Specification (C304, January 1994).

XFTAM's functions implement high-level file transfer and file management operations using the service of an FTAM initiator and service provider which underly the API. This specification therefore defines an API to the simple file transfer and management functions of the OSI file manipulation service element, including the functions and data structures which it provides for use by applications writers.

It is not the purpose of this specification to define a particular subset of the FTAM protocol which XFTAM implementations must support.

Structure

- Chapter 1, **Introduction**, describes the positioning of X/Open OSI APIs, and XFTAM dependencies. It then gives a brief introduction to the FTAM file service. After that, it explains particular terminology and conventions used in this document, then conformance requirements for an API implementation and for the underlying FTAM service provider, and it closes with indications of development directions regarding the OSI standards upon which this XFTAM specification is based.
- Chapter 2, **XFTAM Overview**, lists XFTAM's functions and data structures and describes the model on which the API is based. It includes description of the context-sensitive mode of operation. This chapter also describes aspects of the use of the API, including the use of the XOM API to support transfer of control information between XFTAM and the API user.
- Chapter 3, **XFTAM Base Package - XOM Class Definitions**, lists the XOM Class Definitions which are used to pass information control between XFTAM and the API user. This chapter

defines a class hierarchy for the basic XFTAM *XOM Package* and, for each class in the package, it lists the *XOM attributes* of the class, defining the syntax of each attribute and other aspects such as limits on its value or number of occurrences.

- Chapter 4, **XFTAM Function Manual Pages**, presents the manual page definitions for the functions provided by the XFTAM API.
- Chapter 5, **XFTAM Return Codes**, lists the standardised values returned by XFTAM functions, describing the meaning of each code and a possible corrective action.
- Appendix A, **Summary of XOM**, presents a short summary of the X/Open OSI-Abstract-Data Manipulation API (reference **XOM**), describing its functionality. This appendix is provided for the convenience of readers who are not familiar with X/Open's XOM API.
- The **Glossary** provides a short description of the meaning of some key terms used in this specification.

Intended Audience

The intended audience for this specification includes two distinct groups of readers:

- API Implementors
  System Vendors who are implementing an OSI stack may use this specification to design an XFTAM-conformant interface for the services of an FTAM initiator. XFTAM supports the design of portable file transfer applications.
- Applications Programmers
  Implementors of applications which are to use FTAM-based file transfer can use the set of functions and data structures described in this specification in order to produce an application which is portable across OSI protocol stacks from a range of system vendors.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, class names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ). Names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.

- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
- The notation [ABCD] is used to identify a return value ABCD, including if this is an an error value.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.

Trade Marks

Referenced Documents

The following documents comprise the FTAM file service:

ISO / IEC 8571-1
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management - Part 1: General Introduction. ISO / IEC 8571-1:1988 (E)

ISO / IEC 8571-2
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management - Part 2: Virtual Filestore Definition. ISO / IEC 8571-2:1988 (E)

ISO / IEC 8571-3
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management - Part 3: File Service Definition. ISO / IEC 8571-3:1988 (E)

ISO / IEC 8571-4
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management - Part 4: File Protocol Specification. ISO / IEC 8571-4:1988 (E)

ISO / IEC 8571-5
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management - Part 5: Protocol Implementation Conformance Statement Proforma. ISO / IEC 8571-5:1990(E)

Four filestore management addenda to ISO / IEC 8571 FTAM have been published:

ISO / IEC 8571 DAM
: Information processing systems - Open Systems Interconnection - File Transfer, Access and Management:

— Amendment 1:1992 to ISO 8571-1:1988: Filestore Management

— Amendment 1:1992 to ISO 8571-2:1988: Filestore Management

— Amendment 1:1992 to ISO 8571-3:1988: Filestore Management

— Amendment 1:1992 to ISO 8571-4:1988: Filestore Management.

The following documents comprise the FTAM Profile, ISP 10607.

ISP 10607-1
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 1: Specification of ACSE, Presentation and Session Protocols for the use by FTAM. ISO / IEC ISP 10607-1:1990(E)

ISP 10607-2
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 2: Definition of document types, constraint sets and syntaxes. ISO / IEC ISP 10607-2:1990(E)

ISP 10607-3
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 3: AFT11 - Simple File Transfer Service (unstructured). ISO / IEC ISP 10607-3:1990(E)

ISP 10607-4
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 4: AFT12 - Positional File Transfer Service (flat). ISO/IEC ISP 10607-4:1991(E)

ISP 10607-5
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 5: AFT22 - Positional File Access Service (flat). ISO/IEC ISP 10607-5:1991(E)

ISP 10607-6
: Information technology - International Standardized Profiles AFTnn - File Transfer, Access and Management - Part 5: AFT3 - File Management Service. ISO/IEC ISP 10607-6:1991(E)

The following X/Open publications are referenced in this specification:

I/W SG
: Interworking API Style Guide, X/Open Snapshot, S030, December 1990.

XDS
: API to Directory Services (XDS), X/Open CAE Specification, C317, ISBN 1-85912-007-5, December 1993.

XNFS
: Protocols for X/Open Interworking: XNFS, Issue 4, X/Open CAE Specification, C218, ISBN 1-872630-66-9, October 1992.

XOM
: OSI-Abstract-Data Manipulation API (XOM), X/Open CAE Specification, C315, ISBN 1-85912-008-3, December 1993.

Chapter 1

Introduction

This document is a CAE Specification defining the X/Open XFTAM API, a programming interface to the OSI File Transfer, Access and Mangement protocol. XFTAM's functions implement high-level file transfer and file management operations using the service of an FTAM initiator and service provider which underly the API.

Motivation

X/Open has defined Application Programming Interfaces (APIs) which can be used to access the OSI protocol stack at a number of levels. Figure 1-1, **X/Open OSI APIs**, shows some of these interfaces. The X/Open Transport Interface (XTI) provides an interface to the Transport Layer services of a range of protocol stacks including OSI and TCP/IP. XTI facilitates portability of applications among different protocol suites and among operating system and protocol stack platforms. Higher-up the stack, the ACSE/Presentation Services Application Programming Interface (XAP) provides access to the common connection-oriented services of the upper layers of the OSI protocol stack. XAP provides a standardised interface at the highest point of commonality shared by most Application Layer services, making possible the separation of application-specific and common elements of the OSI protocol stack.

Whilst both may be made available to system users for implementation of local applications, these interfaces are considered to be 'low-level', designed principally for use by systems suppliers and software vendors, providing portability interfaces which increase the applicability of software products and allow a range of application products to be supported by a single common protocol stack.

X/Open is also actively developing a number of APIs to application-specific OSI application service elements. These APIs are in general higher-level than those discussed previously, In particular, there may not be a one-to-one mapping between an API's functions and the service primitives of the underlying protocol. The XAP API gives direct access to the primitives of its underlying services, allowing the API user to send and receive individual primitives. On the other hand, these application-specific APIs are likely to operate in a request-response mode, in which an API function initiates an action and returns the response to the caller when it arrives. XFTAM belongs to this category of APIs, providing an API to the simple file transfer and management functions of the OSI file manipulation service element.

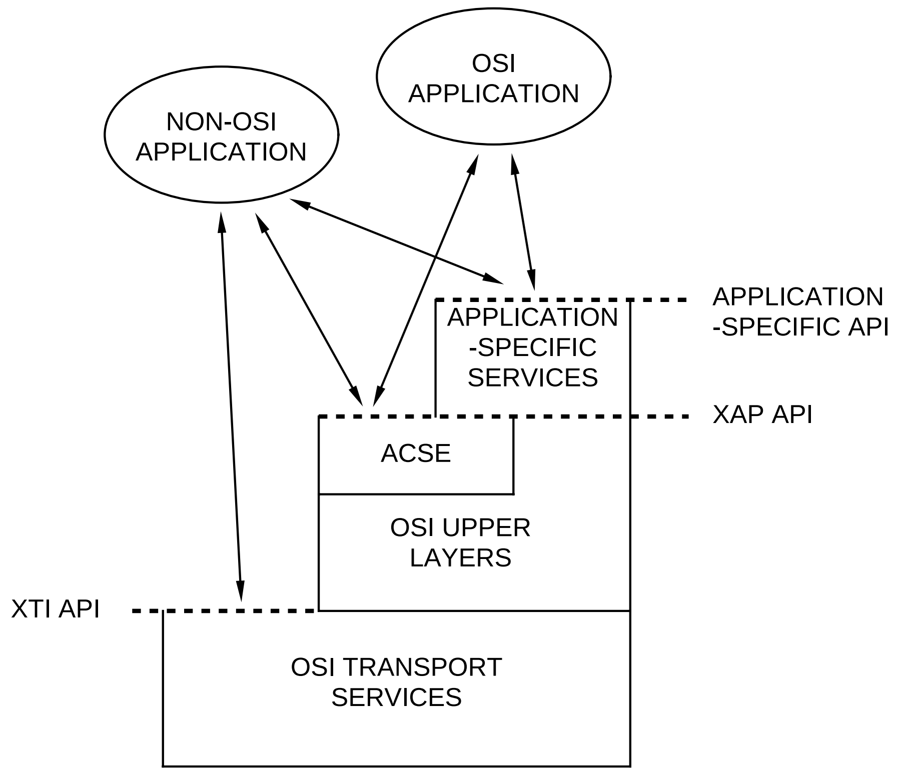

Figure 1-1 X/Open OSI APIs

Purpose of Specification

The purpose of this CAE Specification is to describe the XFTAM API and to define the functions and data structures which it provides for use by applications.

It is not the purpose of this specification to define a particular subset of the FTAM protocol which XFTAM implementations must support. The compliance requirements for an implementation of the API and the underlying FTAM service to which is provides access are defined in Section 1.4 on page 12.

Scope of Specification

This section discusses the scope of this version of the XFTAM specification, it uses some terms defined by the FTAM specification. These terms are described in the **Glossary** at the end of this specification, and in the summary of FTAM functions provided in Section 1.2 on page 5.

XFTAM specifies a 'C' programming interface only. No language independent definition of the API is provided. The API is operating system independent - it is not limited to X/Open CAE-conformant systems.

The ISO FTAM specification (reference **ISO 8571**) provides a wide range of services for the manipulation of an *FTAM virtual filestore*. An international standardised profile exists to define subsets of the full FTAM capabilities that are sufficient to support specific applications of the protocol. This version of the XFTAM API supports the Simple File Transfer and the File Management Services and thus provides functions for file transfer and file management only.

The Positional File Transfer and Positional File Access Services have been excluded from this version of XFTAM, partly because these profiles are not yet stable, and partly because the scope of XFTAM has been limited in order to benefit from implementation experience before developing the API to include other FTAM services. As a consequence, access to the individual *data units* of an FTAM file is not supported.

Extensions to the FTAM protocol to support *Filestore Maintenence* are currently being developed within ISO. These features are excluded from the scope of XFTAM as they do not yet have International Standard status.

FTAM defines two roles for a service user - *initiator* and *responder*. XFTAM provides access to the services of an FTAM initiator only, for the purposes of transferring files to and from a *FTAM virtual filestore* and managing files in such a filestore. XFTAM provides access to the local filestore of the system in which it runs - an XFTAM implementation is not required to implement the full semantics of the FTAM VFS in the local filestore.

The FTAM specification provides a mechanism for the storage and transfer of arbitrarily complex *document types*, and defines a small number of basic document types. This version of the specification supports the *transfer* of *FTAM-1*, *FTAM-2* and *FTAM-3* document types only. The types of file that can be *managed* are not restricted.

Note: Whilst file transfer operations which transfer complete files are supported for the *FTAM-2* document type, *file access* operations which access individual parts of an FTAM-2 file are not supported).

The *NBS-9* document type, defined in the FTAM Profile (reference **ISP 10607**), is included for the purposes of supporting multiple file transfers. Such a file may not be the subject of a file transfer or management function itself.

1.1 Dependencies

An implementation of XFTAM must be supported by the following APIs:

- **XOM -** XFTAM relies upon an implementation of the XOM API, as specified in the **XOM** specification (reference **XOM**), to support the passing of control information between the API user and XFTAM.
- **Directory Service -** In order to use the file transfer and management functions of XFTAM, the API user must supply a presentation address to identify the FTAM responder that serves the remote filestore. The mechanism by which the API user obtains such a presentation address is not defined by XFTAM (it may be via a local lookup mechanism, or by access to a distributed directory service). X/Open have published the XDS API (reference **XDS**) to specify a means to provide portable access to a directory service. Whilst it is not a requirement to use the XDS API, XFTAM uses the *Presentation-Address* object class defined in the *Directory Service (DS)* package of the **XDS** specification in order to facilitate the transfer of presentation addresses between XDS and XFTAM.

1.2 Overview of FTAM

This section gives a brief overview of the FTAM file service for those readers not familiar with the general concepts. For further details the reader is referred to the FTAM specifications outlined below (particularly part 1, the general overview). An understanding of FTAM is assumed in the remaining parts of this specification. Readers who are already familiar with FTAM may wish to proceed with the remainder of the Introduction.

Due to the limited scope of the XFTAM API compared to that of the FTAM file service as a whole, this overview covers only those aspects of the service which are relevant. This means that significant areas of FTAM functionality are not mentioned. In particular, the FTAM hierarchical file model is not discussed in detail, and features applicable only to file access are ignored.

1.2.1 FTAM Specification

The FTAM file service is defined by the referenced FTAM specification, **ISO 8571**, which consists of five parts, as follows:

- Part 1: General introduction
- Part 2: Virtual Filestore Definition
- Part 3: File Service Definition
- Part 4: File Protocol Specification
- Part 5: Protocol Implementation Conformance Statement Proforma.

In addition, an addendum specifying facilities for maintenance of entire file systems is under preparation (see Section 1.5 on page 14).

1.2.2 FTAM Profile

The FTAM file service provides a wide range of features related to the transfer, access and management of files, many of which are optional. An implementation designed for a specific type of application or environment may wish to implement a subset of the FTAM service, leaving out optional features that are not relevant. However, unless there is agreement on which features are to be supported, such an implementation may not be able to interwork with other implementations due to incompatibilities caused by the selection of different subsets.

To address this problem, the referenced International Standardised Profile, **ISO ISP 10607**, has been defined for the FTAM file service. This ISP consists of six parts, not all of which have been ratified as international standards at the time of publication. The profile specifies requirements in addition to those in the FTAM specifications, for an implementation of the FTAM file service. It identifies as mandatory some of the features marked as optional in the base FTAM specification (where a feature is mandatory in FTAM, it is of course mandatory in the profile too). The profile consists of six parts. Parts 3 to 6 specify requirements for specific types of file service applications; parts 1 and 2 specify requirements which apply to all the types of application identified by the profile.

The constituent parts of ISP 10607 are as follows:

- **Part 1: Specification of ACSE, Presentation and Session Protocols for the use by FTAM -** specifies requirements for the use of the services of the upper layers of the OSI protocol stack.
- **Part 2: Definition of document types, constraint sets and syntaxes -** specifies the requirements for support of the various document types defined by the FTAM standard and by other organisations such as the regional OSI workshops.

- **Part 3: AFT11 - Simple File Transfer Service -** supports transfer of files and reading of file attributes.
- **Part 4: AFT12 - Positional File Transfer Service**.
- **Part 5: AFT22 - Positional File Access Service**.
- **Part 6: AFT3 - File Management Service -** supports all file management operations, including writing file attributes and deleting files.

The requirements for support of these profiles by the service provider which underlies an implementation of XFTAM are discussed in Section 1.4 on page 12.

1.2.3 Model for the FTAM file service

The FTAM file service is based upon the *FTAM virtual filestore (VFS)*, a model for describing files and their attributes. The VFS allows two systems with different real file systems to interwork in terms which are mutually understood. FTAM file service users must map the VFS onto the real filestore that exists on the local system. In the VFS, a file consists of a set of *file attributes*, zero or more *data units* representing the contents of the file, plus *structuring information* required to represent the organisation of the information in the file.

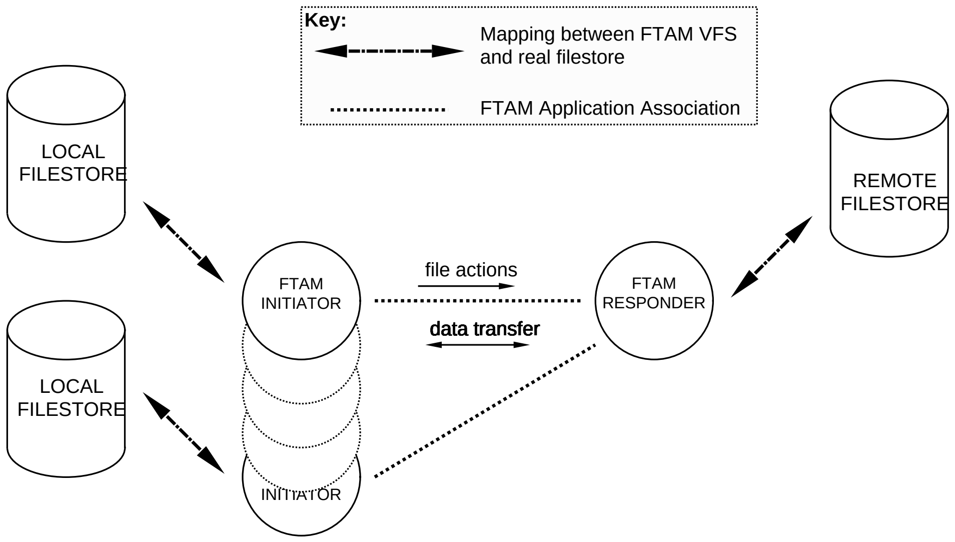

Figure 1-2 FTAM File Service Model

FTAM implements an *asymetric* dialogue between the two file service users participating in an FTAM association. The roles in the dialogue are as follows:

Initiator The initiator of an association is the controlling party responsible for initiating all activity in order accomplish the objective of the file service user, such as transfer of file data or a file maintenance operation.

Responder The responder takes a purely passive role in an association, performing actions as requested by the initiator of the association.

A common mode of FTAM usage allows initiator *clients* to access a remote virtual filestore controlled by an FTAM responder. In this mode an FTAM initiator performs a file transfer between its local filestore and a filestore controlled by the remote responder. Alternatively, the initiator may perform maintenance actions on files in the remote filestore. Figure 1-2 on page 6, **FTAM File Service Model**, illustrates this mode of use.

1.2.4 File Attributes

FTAM defines a comprehensive set of attributes that can be used to describe the characteristics and content of a file. These are listed below. A particular responder implementation may not support every attribute that FTAM defines because they are not all relevant to the real filestore to which it provides access. For this reason the attributes are grouped, with support of some groups being optional for a responder implementation. A responder may support an optional attribute group but only *partially support* one or more of the attributes in the group, supplying the value ’no value available’ when the value of the attribute is passed to an initiator. (As an initiator implementation, XFTAM must support all groups in the sense that it must be capable of accepting and supplying values for all FTAM file attributes.) The FTAM file attributes are listed below. Some of them have no relevance to the types of FTAM file operations supported by an XFTAM implementation. As described in Chapter 2, only a few of them *must* be specified when a file is created in a remote filestore.

Kernel Group

This group contains attributes that are always supported. An FTAM responder always stores and returns values for these attributes. The kernel attributes are:

- **Filename -** allowing the file to be referenced in the VFS without ambiguity.
- **Permitted actions -** identifies the set of actions that are permitted on the file (for example, *read*, *replace*, *extend* and *change attribute*). Permitted actions also identifies the set of *FADU identity styles* that are permitted for the file. However, as this aspect of FTAM is not relevant to the types of file access supported by XFTAM, they are not described here.

  The value of this attribute is set when the file is created and cannot be modified subsequently (the attribute is intended to reflect the basic characteristics of the file as opposed to file access permissions which may vary during the lifetime of the file).
- **Contents Type -** indicates the structure of the file and the type of data stored in it. The contents type is used to preserve the meaning of the file during transfer. The type may be specified in one two ways. For the purposes of XFTAM, however, contents type may only be specified as a *Document Type Name* (described below).

  This attribute is set when the file is created and cannot be modified subsequently.

Storage Attribute Group

Support of this attribute group is optional for a responder. If the group is supported, each attribute is either *fully supported* (a meaningful value is returned by the responder) or *partially supported* (the value "no value available" is returned).

- **Storage Account -** identifies who is responsible for charges associated with storing the file.
- **Date and time attributes -** these attributes indicate the date and time of file creation and of the last content modification, read access, and attribute modification.

  These attributes are set by the FTAM responder as a result of the completion of the related file actions and cannot be modified directly by a file service user.
- **Identity attributes -** these attributes indicate the identity of the file creator and the previous content modifier, reader or attribute modifier.

These attributes are set by the FTAM responder as a result of the completion of the related file actions and cannot be modified directly by a file service user.

- **File availability -** indicates the delay that should be expected when the file is opened. The value is *immediately available* (i.e. the file is stored on a non-demountable device) or *deferred availability* (the file may be stored on a demountable device such as magnetic tape).

  The attribute is set when the file is created and can be modified subsequently.

- **Filesize -** indicates the size (in octets) of the file. This attribute is set by the FTAM responder as a result of modification or extension.

  This attribute is set by the FTAM responder as a result of the completion of the related file actions and cannot be modified directly by a file service user.

- **Future filesize -** indicates the size (in octets) to which the file may grow as a result of modification or extension.

Access Control Attribute Group

Support of this attribute group by a responder is optional. If the group is supported, each attribute is either *fully supported* (a meaningful value is returned by the responder) or *partially supported* (the value "no value available" is returned).

- **Access Control Attribute -** specifies a set of conditions under which access to the file is to be permitted. Conditions may include the identity of the initiator, the location of the initiator, and provision of passwords required to perform file actions. Chapter 2, provides more details of how this attribute may be used.

- **Legal Qualifications -** indicates the legal status of the file in respect of national data protection legislation. The FTAM specification makes no further statement about the use of this attribute.

Private Group

This optional attribute group consists of a single attribute *private use.* The form and meaning of this attribute is not defined by the FTAM specification.

1.2.5 Document Types

The FTAM VFS provides a flexible *file model* for describing the attributes, structure and contents of file in a manner independent of the real filestore in which it resides. The FTAM specification defines a small set of named *document types* which use the file model to represent simple commonly encountered types of file. Other organisations, such as the regional OSI workshops, have defined other document types for specific applications. The following sections describe document types supported by the XFTAM API.

A document type is uniquely identified by an ASN.1 *object identifier* and *object descriptor*. However, the type is usually referred to by an *entry number* (for example *FTAM-1*). A document type specifies rules for the structure of the file (known as the *constraint set*) and for its content (known as the *abstract syntax*). In addition, a document type can define one or more qualifying *parameters* which allow it to apply to a group of closely related document types.

FTAM-1 Document Type

The FTAM-1 document type specifies an unstructured file consisting of zero or more text *strings*. An application may only access the contents of an unstructured file as a whole; access to portions of the file is not possible. Optional parameters may be used to specify characteristics of the strings in the file. These are the *universal class number* (ASN.1 character set for the characters which make up the strings of the file), and a *maximum string length* and a *string significance*, which together constrain the strings in the file.

This document type is the natural one for representing a real file containing lines of text, where the strings of the document type may be mapped to the text strings of the file as delimited by the local end-of-line convention. Note that FTAM-1 does not associate any semantics with lines of text, so that such mapping is outside the scope of the FTAM standard. It may also represent text files such as screen or print images, where the contents contain format effectors which control the display or printing of the information. All XFTAM implementations support this document type, as do responder implementations which conform to the FTAM profile.

FTAM-2 Document Type

The FTAM-2 document type specifies a sequential file consisting of zero or more records, each of which consists of zero or more text strings. The records in the file may be referenced by position (record number) or in sequence (first/next, last/previous). As for the FTAM-1 document type, the records of the file contain text strings, with the same set of optional parameters to specify their characteristics.

Support for transfer of this document type is optional in the FTAM profile. A particular XFTAM implementation, or the responders with which it interacts, may not support it. XFTAM does NOT support file access operations on individual parts of an FTAM-2 document.

FTAM-3 Document Type

The FTAM-3 document type defines an unstructured file containing zero or more strings of binary data. The structure and content of the file are as for the FTAM-1 document type with the exception that there is no *universal class number* parameter; each string in the file is a simple sequence of octets, unrestricted in value.

This document type can be used to represent files where the content of the file is not restricted. All XFTAM implementations support this document type, as do responder implementations which conform to the FTAM profile.

NBS-9 Document Type

This document type, defined by the FTAM Profile (reference **ISP 10607**), allows the transfer of information about the directory structure of a real file system (FTAM currently defines the name of a file as an uninterpreted string and does not recognise the concept of a filestore structure). The document type contains zero or more records, each listing the attributes of a file in the named directory. The list of attributes returned in a particular request contains only those supported by the responder - thus it might contain only the attributes in the *kernel group*.

XFTAM provides indirect access to this file type to assist in the implementation of multiple file transfers. Support of the file type is optional in the FTAM profile - XFTAM implementations must be able to request it but some responder implementations may not return the requested information.

1.2.6 File Actions

The FTAM VFS defines a set of generalised actions that can be performed on its files. XFTAM uses these actions to implement its functions. FTAM provides access to these actions via its service primitives. An initiator invokes these primitives to implement a file operation such as a file transfer or retrieval of file attributes. Only those actions which are relevant to understanding XFTAM are listed below (although other actions may be used in implementing a particular XFTAM function):

- *Change attribute* - change the values of one or more file attributes of the currently selected file
- *Read attribute* - read the values on one or more file attributes of the currently selected file
- *Delete file*
- *Read*
- *Replace*
- *Extend* - add data to the end of a file.

1.2.7 FTAM Quality of Service

FTAM recovery action is based upon an *FTAM Quality of Service* agreed between the initiator and responder when an FTAM regime is initialised. This quality of service indicates the error classes to which the application is susceptible, i.e. the error classes for which the initiating user wishes the underlying FTAM implementation to attempt recovery.

FTAM defines an error recovery protocol, with associated service primitives, that implements the requested recovery action. The actual FQoS negotiated for an association is based upon the level of support of these optional recovery primitives, along with local considerations (such as the inherent reliability of the systems involved). Consequently, the FQoS agreed for an association may be lower than that requested by the initiating user. In this case it is up to the FTAM service user to decide whether to terminate the regime or to continue despite the reduced service quality.

1.3 Terminology and Conventions

The special terminology and conventions used in this specification are derived primarily from the referenced **XOM** and **FTAM** specifications. An overview of the former is presented in Appendix A, and of the latter in the **FTAM Overview**, Section 1.2.

Both specifications make use of the term *attribute*. XOM uses it to describe a value type which forms part of an XOM *Object Class*, whilst XFTAM uses it to describe a characteristic of an FTAM file. In this specification, in general, the particular meaning intended is clear from the context of its use. However, where this is not clear, the reference is qualified as *FTAM attribute* or *XOM attribute*. The term *XFTAM attribute* is used to refer to an XOM attribute of an object class defined by the *XFTAM package*. .

Similarly there is a clash between FTAM's use of the term *parameter*, which refers to a value passed in an FTAM service primitive, and this specification, where it refers to a value passed to or from an API function. When used on its own, it refers to the API meaning. It is qualified as *FTAM parameter* when used to refer to the other meaning.

The XFTAM functions and associated identifiers are defined as a binding to the 'C' language, using the general typographical conventions established for X/Open APIs. The prefix *ft* or *FT* is used for all names to ensure uniqueness of identifiers which appear in header files or are visible when a program is linked. In addition, all identifiers beginning with *ftp* or *FTP* are reserved for internal use within an implementation of XFTAM. Identifiers beginning with *ftx* or *FTX* are reserved for use by vendor extensions to this specification.

The convention for XOM packages is to use *abstract* names from which the 'C' binding can be derived mechanically. The rules for deriving the 'C' binding are defined by the X/Open *Interworking API Style Guide* (reference **I/WSG**). The following list summarises how those rules are applied to derive the 'C' binding for the XTAM Base Package.

- XOM class identifiers are derived from the abstract class name by converting it to upper case, converting hyphens to underscores, and adding the prefix *FTC_*. Thus, *Access-Control-Element* becomes *FTC_CONTROL_ELEMENT.*
- XOM attribute identifiers are derived from the abstract attribute name in the same way as XOM class identifiers except that the prefix is *FT_*. Thus, *File-Action-List* becomes *FT_FILE_ACTION_LIST.*
- Enumeration tag identifiers are derived from the name of the value set by converting hyphens to underscores (but preserving case) and adding the prefix *FTA_*. Thus, *File-Action-List* becomes *FTA_FILE_ACTION_LIST.*
- Enumeration constant identifiers are derived from the abstract name of the constant in the same way as XOM attribute identifiers. Thus, *Printable-String* becomes *FT_PRINTABLE_STRING.*

1.4 Conformance

Two groups of conformance requirements are defined in this specification: conformance requirements for an API implementation, and conformance requirements for the underlying FTAM service provider.

Conformance Requirements for an API Implementation

A conformant XFTAM implementation provides, as a minimum, all the functions defined in Chapter 4, and all the OM classes of the *XFTAM Base Package* defined in Chapter 3. The following points should be noted:

- When the API user requests a feature which is supported by the API but which the underlying FTAM service provider does not support, the API returns an *FTE_NOTSUP_XXX* error.
- An implementation of the API defines values for all the return codes defined by this specification. However, the functions of the API return only those values which the underlying FTAM service provider is capable of indicating. Conversely, where an FTAM service provider indicates an error for which XFTAM defines a return code, the API returns that code rather than any other (XFTAM provides an optional vendor-defined return code to further qualify the result of a function).
- An implementation may extend the API by adding new functions or by defining new XOM packages containing additional classes (which may be sub-classes of classes defined by the XFTAM API package). An implementation may not change the definition or semantics of existing XFTAM functions or XOM classes or add new OM attributes to existing XOM classes.

List of Optional Features

The following list highlights the features of the XFTAM API that are defined as optional. In general, features of the underlying FTAM initiator that are defined as optional in the base specification or associated ISPs are not included in this list.

- *ft_abandon* function - supported if the implementation supports asynchronous operations.
- *ft_rcvresult* function - supported if the implementation supports asynchronous operations.
- *Asynchronous* XOM Attribute - supported if the implementation supports asynchronous operations.
- *Invoke-ID* XOM Attribute - supported if the implementation supports asynchronous operations.
- *Session-Handle* XOM Attribute - supported if the implementation supports XFTAM instances in the local event handling mechanisms.
- *Source_attributes* or *return_attributes* parameter of *ft_receive*( ) specifying *Document-Type-FTAM-2* - supported if the implementation supports the FTAM-2 document type.
- *Initial_attributes* or *return_attributes* parameter of *ft_send*( ) specifying *Document-Type-FTAM-2* - supported if the implementation supports the FTAM-2 document type.
- *FQoS* enum values other than *No Recovery* - supported if the implementation provides the corresponding level of recovery support.
- *Contents type lookup service* - supported if the implementation is capable of determining a file contents type by *ft_fsend*( ) function when no contents type specified by the XFTAM user.

Conformance Requirements for the Underlying Service Provider

An implementation which complies with this specification shall also comply with the International Standardized Profiles (ISP) which are listed in the XFTAM Component Definition.

International Standardized Profiles place additional constraints on the PICS (the Protocol Implementation Conformance Statements). These constraints are in terms of a requirement list - effectively deltas to the (protocol) PICS status colum - and contain additional questions relevant to the profile (for example, required ranges of supported parameter values).

The ISPs relevant to this specification are part of the multi-part profile **ISO/IEC ISP 10607**, which references the ISO / IEC Session, Presentation, ACSE and FTAM protocols. PICS Proforma for the ISO / IEC Session, Presentation and ACSE protocols are currently under ballot.

An implementation of XFTAM must be accompanied by a completed set of the available PICS Proforma, showing that the requirement lists of the ISPs are met by the implementation.

The following points should be noted:

- An FTAM service provider must support all FTAM file attributes when sending files to a remote filestore, reading attributes and changing attributes. When receiving files for storage in the local filestore, the service provider must accept values for all attributes but XFTAM is NOT required to store those attributes which do not map to an equivalent local filestore file attribute.
- An FTAM implementation is required to support the file attributes within the minimum attribute range defined in ISO 8571-2, section 15.
- Implementation, by the underlying FTAM initiator, of the error recovery procedures (and the associated RESTART and RECOVER functional units) is optional. Where the XFTAM user requests a level of FTAM quality of service (FQoS) that is not supported by the initiator because of implementation-defined restrictions, the requested XFTAM operation fails with the error code [FTE_NOTSUP_FQOS]. Where the requested FQoS is reduced during negotiation of the FTAM association, the operation fails with error code [FTE_FQOS_NOT_NEGOTIATED].

1.5 Future Directions

This section highlights possible future developments in scope and functionality of the FTAM specification. Mention of these possible developments does not imply a commitment on the part of X/Open to produce a revised version of this specification.

Filestore Management

Four addenda to the FTAM specification (see reference **ISO 8571-x**) have been published by ISO. These addenda extend the FTAM file model and introduce new *filestore maintenance* actions.

File Access Functions

X/Open has no plans at the time of writing to extend the scope of this API to encompass file access functions.

Transparent File Access

X/Open has no plans at the time of writing to provide lower level APIs to the FTAM file service. Other organisations involved in the development of standards are working on a definition of the protocols and file system semantics required to provide *transparent file access* using FTAM. When this work is complete, X/Open may define a mapping of the X/Open CAE filestore onto the FTAM VFS for the purposes of supporting transparent file access over FTAM. This is similar to the definition of the semantics of the X/Open CAE filestore when accessing NFS-based files, published in the referenced **XNFS** specification.

Chapter 2

XFTAM Overview

This chapter provides an overview of the features of the XFTAM API, listing its functions and providing information on how to use them, including details of how the OSI-Abstract-Data Manipulation API supports passing of control information between the API and the user and the use of the XDS API to support addressing the remore filestore.

2.1 XFTAM Model

The elements of the XFTAM API model are shown in Figure 2-1. It should be noted that whilst the figure shows the FTAM initiator and service provider as separate boxes, XFTAM makes no statements about the architecture or implementation of the software underlying the API other than defining the functions that it performs. Also the responder elements show a typical arrangement, and do not imply any requirement imposed on the types of responder architectures with which an implementation of XFTAM may interwork.

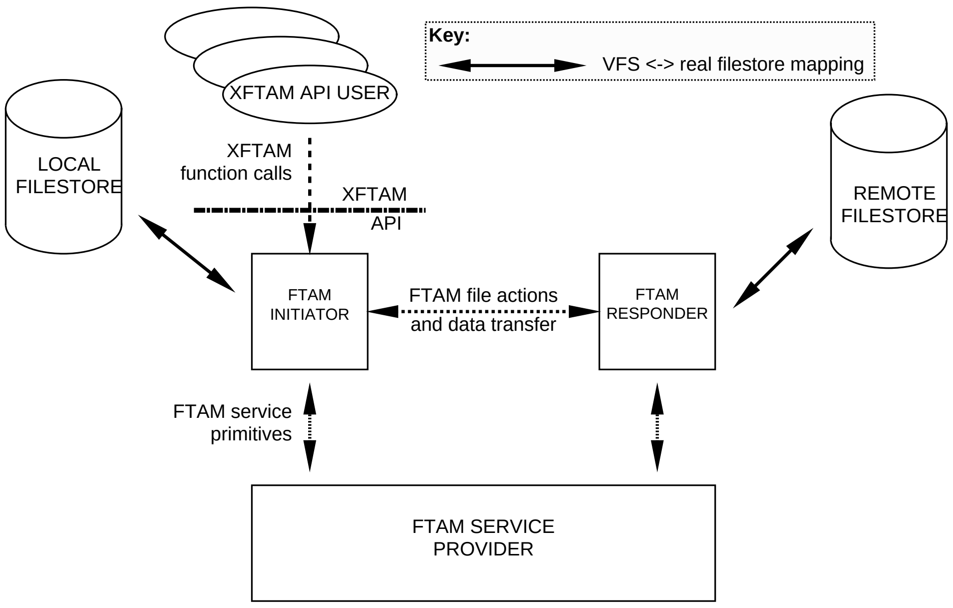

Figure 2-1 XFTAM Model

The XFTAM API provides a set of functions which implement high-level file transfer and management tasks. These are termed *XFTAM operations* in this specification, as an operation can be a request to transfer a file from the local filestore to a remote one, or to read the attributes of a remote file.

Underlying the API is an FTAM *initiator* that implements the XFTAM operations using the services of an FTAM service provider.

2.1.1 Context Free Operations

In the context-free processing mode, in order to implement an operation, the local initiator establishes an *application association* with an *FTAM responder* that serves the remote filestore, and performs a series of FTAM *file actions* (for example, going through the steps required to establish the FTAM *select regime*, invoking the *read attributes* file service, and then terminating the regime created), closing the association and returning the result of the operation to the API user once all the steps are complete. This is termed *context free operation* and is the default mode of operation for XFTAM.

2.1.2 Context Sensitive Operations

In the alternative, context-sensitive processing mode, the FTAM initiator establishes the application association with the FTAM responder (using the F-INITIALIZE FTAM service to establish an FTAM regime), and this association remains to provide a context for subsequent operations (for example, invoking the delete file servies). When each operation completes, it returns the result to the API user and the association remains, ready for another operation.

When the API user has completed all the required operations the association is destroyed (using the F-TERMINATE FTAM service).

The initiator interfaces to the local filestore in order to implement file transfers, mapping between the attributes and content of a file as defined by the FTAM VFS, and the local file semantics - for example, mapping the local file name to the FTAM attribute *filename*, or mapping the strings of an *FTAM-1* unstructured text file into lines of text in a local file, terminated by the local end-of-line convention. The initiator uses the FTAM service to set up an association with a remote FTAM responder which provides access to the remote VFS. Of course the responder is usually implementing a similar mapping between the semantics of the VFS and those of its local filestore. However no assumptions are made about the architecture of the remote FTAM implementation.

A user interacts with the API via an *XFTAM instance*, which is the collection of state information required to perform XFTAM operations on the user's behalf. XFTAM instances are created and deleted by the API user as required. A user may create many independent XFTAM instances within a single program (for example where distinct sections of the application make independent use of the API).

2.2 XFTAM Feature Summary

The XFTAM API provides a set of functions to support FTAM file transfer and management operations. That is, XFTAM allows an application to transfer complete files to and from a remote *Virtual Filestore* (VFS) and to perform management operations on those files (read attributes, change attributes, and delete file). A function is provided to list the contents of a directory on a remote filestore (this function relies upon an optional feature of FTAM - the *NBS-9* document type - which may not be supported by the underlying initiator or the responder which provides access to the remote filestore).

The XFTAM API provides the following functions and features:

API Support Functions

- *ft_open*( ) - create an XFTAM instance and allocate an XOM workspace for the associated package(s).
- *ft_close*( ) - destroy an XFTAM instance and free the associated workspace.
- *ft_connect*( ) - create an association to a remote FTAM responder within an existing XFTAM instance
- *ft_disconnect*( ) - destroy an association established by *ft_connect*( )
- *ft_abort*( ) - abort an association and any outstanding asynchronous operations.
- *ft_abandon*( ) - abandon an outstanding asynchronous operation (optional).
- *ft_rcvresult*( ) - receive the results of an outstanding asynchronous operation (optional).
- *ft_gperror*( ) - translate an XFTAM return code into a printable string.

File Transfer Functions

- *ft_fsend*( ) - send a file from the local filestore to a remote VFS.
- *ft_freceive*( ) - receive a file from a remote VFS into the local filestore.

File Management Functions

- *ft_frattributes*( ) - read the attributes of a file in a remote VFS.
- *ft_fcattributes*( ) - change the attributes of a file in a remote VFS.
- *ft_fdelete*( ) - delete a file in a remote VFS.

Miscellaneous Functions

- *ft_frdir*( ) - return a list of entries from a directory in a remote VFS.

Document Types

XFTAM supports the following document types as the source or destination for file transfers.

- **FTAM-1** - unstructured text.
- **FTAM-2** - sequential text (optional).
- **FTAM-3** - unstructured binary.

Support for transfer of the FTAM-2 document type is optional in the FTAM profile - a particular XFTAM implementation, or the responders with which it interacts, may not support it. XFTAM does NOT support file access operations on individual parts of an FTAM-2 document. XFTAM optionally supports the NBS-9 document type for listing contents of a directory on a remote filestore.

XFTAM Base Package

In order to facilitate the passing of control information between the API user and XFTAM, the services of the OSI-Abstract-Data Manipulation API (XOM) are used. An *XOM package* has been defined which contains object class definitions for the information to be passed.

Supporting APIs

XFTAM relies upon an implementation of the X/Open XOM API, as specified in the referenced **XOM** specification, to support the passing of control information between the API user and XFTAM.

Also, optionally an implementation of the X/Open XDS API (reference **XDS**) may be used to provide an interface to a directory service for the purposes of obtaining addressing information needed to locate a remote FTAM responder.

2.3 Using XFTAM

The remainder of this Chapter is devoted to discussing specific aspects of usage of the XFTAM API that the user must understand in order to use this API successfully.

2.3.1 Include files

The macros and constant definitions which support the use of the XFTAM API are collected in a header file that must be included in each source file that makes use of XFTAM API functions. In addition, the header file for the XOM must also be included in source files that reference definitions associated with this API. The following *#include* statements are required for a source file that references definitions both APIs:

```
#include <xom.h>          /* definitions for the XOM API   */
#include <xftam.h>        /* definitions for the XFTAM API */
```

2.3.2 Library files

XFTAM provides a library which allows the API user to link its functions into a program. The name of this library, and the mechanisms provided to link a program are a local matter and are not defined in this specification.

2.3.3 XFTAM Instances

Before an API user can invoke file transfer and management operations, an XFTAM *instance* must be created by a call to *ft_open*( ), which initialises the associated resources (such as an XOM *workspace*). *ft_open()* returns a pointer to a read-only private object of class *Session*. This session object is subsequently used to identify the created instance in calls to other XFTAM functions.

A single application program may create multiple XFTAM instances by calling *ft_open*( ) as required. This allows independent parts of a program such as code loaded as link libraries to use XFTAM functions independently.

Once an application program has completed its use of XFTAM functions, it can release the resources associated with an XFTAM instance by calling *ft_close*( ), passing it the associated *Session* object.

2.3.4 Using XOM

Appendix A describes the basic services provided by XOM. This section provides specific guidance on how the API user uses XOM to exchange control information with XFTAM. The XFTAM API relies upon the services of the XOM API to support the passing of control information to and from the API user. The following description discusses aspects of XOM and the XFTAM API that a user must understand in order to use XFTAM successfully.

Creating and deleting the XFTAM workspace

The private objects passed to and returned from an XFTAM *instance* are stored in an XOM workspace which is associated with that instance. As noted above, this workspace is created by the function *ft_open*( ), which returns a handle of type **OM_workspace**. This handle can be used in calls to the XOM functions to manage private objects on behalf of the XFTAM instance.

By default, the workspace created contains the *XFTAM Base Package*, defined in this specification. *ft_open*( ) allows the API user to specify additional *optional* package for inclusion in the workspace created. *ft_open*( ) returns an indication of which of the optional additional packages have been included. This specification does not define any optional XOM packages for use with the XFTAM API. However, future version of XFTAM may specify additional packages to support extensions to the base functionality. Alternatively, individual implementations of XFTAM may choose to define optional packages to support proprietary extensions to the API.

The workspace, and any objects contained in it, are deleted by the *ft_close*( ) function.

Importing XOM Classes

In order to make reference to a particular XOM class within a particular source file, the user of the XFTAM API must *import* the data declarations associated with it (using the macro *OM_IMPORT*). Thus, each source file includes a set of calls to this macro at the beginning of the file (that is before any reference to the XOM class being imported). For example, a reference in a source file to the *FTAM-Input-Parameters* XOM class must be preceded by the following statement:

```
OM_IMPORT(FTC_FTAM_INPUT_PARAMETERS)
```

In addition, one source file within the program must arrange for the actual storage associated with these declarations to be allocated. Thus one source file within the program must *export* the definition instead of importing it (using the macro *OM_EXPORT*). For example, source files which import the data associated with the *FTAM-Input-Parameters* XOM class must be supported by one source file that contains the following statement:

```
OM_EXPORT(FTC_FTAM_INPUT_PARAMETERS)
```

Passing Objects to XFTAM

Many of the input parameters to XFTAM are passed as XOM objects. There are two strategies for creating the objects that are to be passed. One is a static method in which a *public object* is created at compile-time, and the other is a dynamic method in which a *private object* is created and attribute values added to it at run-time. Both methods make use of the structure **OM_descriptor**, which is used to specify a list of one or more XOM attribute values.

Figure 2-2 shows how this structure is used to represent values for the XOM attributes *Initiator-Identity*, *Filestore-Password* and *File-Action-List* from the class *FTAM-Input-Parameters*:

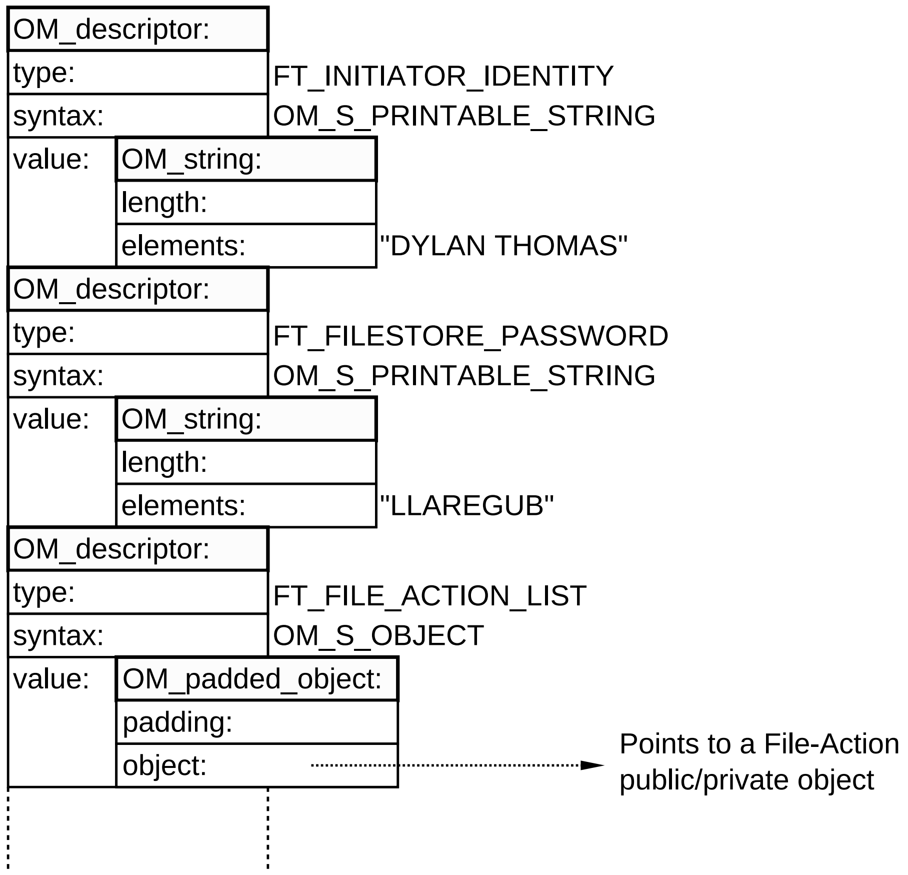

Figure 2-2 Example OM_Descriptor List

In general, if none of the attributes of an interface object class are required for a particular function call, the API user may either supply an object with no attribute values, or supply a null object handle (null pointer).

Creating a Public Object

A public object is created by declaring and initialising a list of *OM_descriptor* structures. Such an object can be set up at compile time using static structures. Each descriptor represents one XOM attribute value in the object. The first XOM attribute in the list is the *Class* attribute (initialised using the *OM_OID_DESC* macro), and the list is terminated by a null descriptor (initialised using the *OM_NULL_DESCRIPTOR* macro). Of course, one or more of the XOM attributes of the class may themselves be public objects, thus creating a hierarchy of descriptor lists.

For example, the following code fragment creates a public object of class *FTAM-Input-Parameters* and sets values for its *Initiator-Identity* and *Filestore-Password* attributes (the *OM_STRING* macro is used to initialise the *OM_string* sub-structure of the **value** union):

```
/*
 * declare a list of OM_descriptors for the FTAM-Input-Parameters class
 */
static ftam_input OM_descriptor[] = {
    OM_OID_DESC( OM_CLASS, FTC_FTAM_INPUT_PARAMETERS),
    {FT_INITIATOR_IDENTITY, OM_S_PRINTABLE_STRING, {OM_STRING("Dylan Thomas")}},
    OM_NULL_DESCRIPTOR
};
```

XOM specifies that the ordering of the descriptors in a public object is not significant except that multiple values of a single attribute type appear in the list consecutively. For example, an object of class *Access-Control-Element* may contain one or more attribute values of type *File-Action-List*. Descriptors for these values may appear anywhere in the list representing an object of this class. However, all such values must be grouped together.

Creating a Private Object

A private object is created by first calling *OM_create*( ), and then calling *OM_put*( ) as required, adding one or more attributes to the object at each call. Again, a list of one or more *OM_descriptor* structures are used to pass the attributes to be *put* into the class. The following example uses XOM functions to create a similar object to that of the previous example:

```
{
    /*
     * set up a blank public object to use for adding attributes to the
     * private object being created.
     */
    static desc_list OM_descriptor[] = {
        OM_NULL_DESCRIPTOR,
        OM_NULL_DESCRIPTOR
    };

    /*
     * create the object, FTC_FTAM_INPUT_PARAMETERS is an imported
     *   global variable containing the object id of the class we are
     *   creating.
     * Returns handle for created object in private_ftam_input
     * For simplicity, errors are ignored
     */
    return_code = OM_create( FTC_FTAM_INPUT_PARAMETERS, OM_TRUE,
                             workspace_handle, &private_ftam_input) ;

    /*
     * prompt for and read an Initiator-Identity string from the terminal,
     *   skip attribute if the string we get is null
     */
    puts( stdin, "Enter Initiator Identity for remote filestore: ");
    if ( strlen( gets( &input_string) ) != 0)
    {
        /*
         * set up a descriptor for the Initiator-Identity attribute
         * and add it to the object just created
         * For simplicity, errors are ignored
         */
        desc_list[1].type = FT_INITIATOR_IDENTITY ;
        desc_list[1].syntax = OM_S_PRINTABLE_STRING ;
        desc_list[1].value.string.length =
                        (OM_element_position) strlen( input_string) ;
        desc_list[1].value.string.elements = (void *) input_string ;
        return_code = OM_put( private_ftam_input, OM_INSERT_AT_END,
                              desc_list, 0, 0, 0);
    }

    /*
     * repeat for other attributes
     */
}
```

Objects Returned by XFTAM

All of the output parameters returned by XFTAM functions are *private objects*. This means that a handle is returned which points to some private representation of an object of the required class. In order to examine the contents of such an object, the API user must extract one or more of its attribute values into a system-generated *public object* using one or more calls to the *OM_get*( ) function. When calling get, the API user specifies the set of attributes to be retrieved, using the *OM_exclusions* and *OM_type_list* parameters. By default, *OM_get*( ) returns the full hierarchy of objects associated with the retrieved attributes. That is, where the value of an attribute is itself an object (a *sub-object*), selected attribute values of that object are also returned. If this is not the required behaviour, the *exclude-subobjects* flag in the *OM_exclusions* parameter can be use to restrict the get to a single level. In this case, a sub-object is represented by a private object handle and must be retrieved as required using the get function.

The rules regarding the ordering of the descriptors in a public object, summarised above for user-created public objects, are also applied by XFTAM in the creation of system-created public objects. The storage associated with a system-created public object is directly accessible by an API user for the purposes of examining and copying the information within it. The application must not modify the contents of such an object. The effect of doing so is undefined.

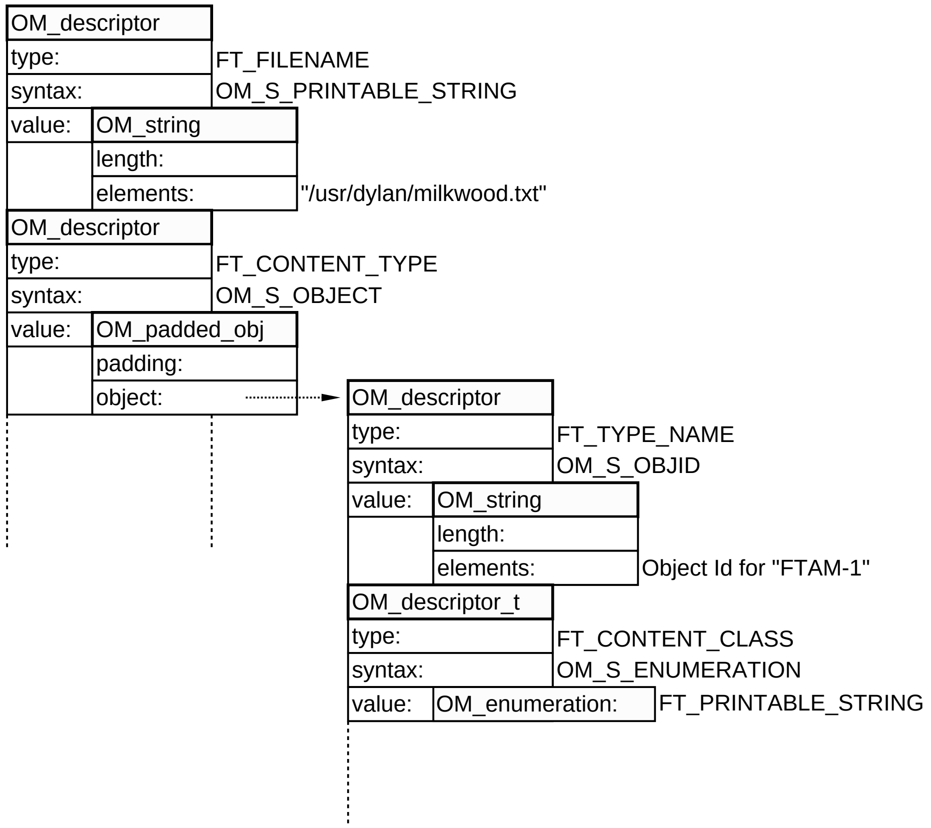

Figure 2-3 Descriptors for the FTAM-Attributes Class

The following code fragment demonstrates how the various elements of an object of class *FTAM-Attributes* are retrieved and processed, and the associated Figure 2-3 shows the structure retrieved by the *OM_get( )* call.

```
/*
 * retrieve selected values for the FTAM Attributes returned by some
 *   function return the sub-objects too
 */
{
    static OM_type ftam_att_attribute_types[] = {
        FT_FILENAME, FT_CONTENT_TYPE, FT_TYPE_NAME, FT_CONTENT_CLASS,
            .
            .
    } ;
    xom_return = OM_get(ftam_attributes,OM_EXCLUDE_ALL_BUT_THESE_TYPES,
                        ftam_att_attribute_types, OM_FALSE, 0, 0
                        &public_p, &descriptor_count ) ;

    /*
     * Now process each of the descriptors in turn, printing the info.
     * Last descriptor is the null descriptor that terminates the list
     */
    for ( descriptor_p = public_p ;
          descriptor_p->type != OM_NO_MORE_TYPES; descriptor_p++ )
        switch ( descriptor_p->type) {
        FT_FILENAME:
            printf( "Filename: %*s0,
                    descriptor_p->value.string.length,
                    descriptor_p->value.string.elements) ;
            break ;
        FT_CONTENT_TYPE:
            /*
             * extract pointer to the Content Type sub-object,
             *   loop through its attributes
             */
            for ( content_p = descriptor_p->value.object.object ;
                  content_p->type != OM_NO_MORE_TYPES; content_p++ )
                switch ( content_p->type) {
                FT_TYPE_NAME:
*
* interpret and print an object identifier
*/
                    break ;
                FT_CONTENT_CLASS:
                    printf( "    Content class: %d0,
                            content_p->value.integer) ;
                    break ;
             .
             .
```

Deleting objects and workspaces

The storage associated with an object created by XFTAM (either public or private) is allocated by the API and, if the space is to be reused, the API user must call the *OM_delete*( ) function to free the associated storage. A pointer which points at part of a deleted public object is invalid. The effect of using such a pointer is undefined. Any information which is to be retained must be copied into storage controlled by the API user before the object is deleted. The storage associated with a user-created public object is of course under the control of the API user, such an object cannot be the subject of an *OM_delete*( ) call.

Before an application exits, or once it has finished using the XFTAM API, it must call *ft_close*( ) to release the resources associated with the API instance.

2.3.5 Addressing the Remote Filestore

All of the file transfer and management functions provided by XFTAM require the API user to identify a remote FTAM responder which provides access to a virtual filestore. A responder is identified to XFTAM by its presentation address.

The mechanism by which the API user obtains the presentation address identifying a particular FTAM responder is not defined by XFTAM (for example, it may be by a local look-up service or a distributed directory service). X/Open has defined the XDS API (defined in reference **XDS**) as an API to directory services. Whilst XFTAM does not require the use of XDS for obtaining presentation addresses, XFTAM uses the *Presentation-Address* object class defined in the XDS-defined *Directory Service (DS)* package in order to facilitate the transfer of presentation addresses between XDS and XFTAM.

To pass an XDS *Presentation-Address* to XFTAM, the API user must obtain the address as a service-generated public object, using *OM_get*( ). The resulting public object handle can then be passed to XFTAM as the src_p_address or dest_p_address parameter for an operation.

2.3.6 Filenames

When specifying the source or destination filename for an XFTAM file transfer function, the name is specified using the semantics defined by the filestore being addressed (local or remote). XFTAM places no restrictions on this. A remote filename is passed to the remote responder as supplied by the API user - no conversion is performed. For files sent, the remote responder may modify the filename when creating the destination file, to match it to the local filename conventions.

The destination filename may be left blank in a file transfer request, implying that the source filename should be used for the destination file. In this case the transfer may fail if the resulting filename is not compatible with the file naming conventions of the destination filestore.

2.3.7 Source Effect

The *src_effect* parameter to a file transfer function determines what happens to the source file when the transfer is completed successfully. Possible values are *copy*, where the source file for the transfer is left in place when the transfer is complete; or *move*, where the source file is deleted once the transfer has completed successfully.

Of course, in order for the source file to be deleted, the API user must have established the appropriate access rights for the file. For files *received*, this requires permission to perform the FTAM *delete action* (see Section 2.3.16 on page 34. For files *sent*, permission is controlled by the API user's access rights in the local filestore and is outside the scope of FTAM and the XFTAM API.

2.3.8 Destination Effect

The *dest_effect* parameter to a file transfer function determines the required action when the destination file for a file transfer function already exists. The values for this parameter depend on the direction of transfer:

- For files being received:
  The possible values are *fail*, *extend* or *overwrite*. These are self-explanatory.

- For files being sent:
  The possible values for this enumeration correspond to the values of the FTAM *override* parameter (that is *create-failure*, *select-old-file*, *delete-and-create-with-old-attributes*, and *delete-and-create-with-new-attributes*). *Select-old-file* has the effect of appending the contents of the local file to those of the remote one. For both *select-old-file* and *delete-and-create-with-new-attributes* values, the transfer fails if the contents type for the file being transferred are not compatible with those of the target file.

Again, for any destination effect which results in an existing file being modified or overwritten, the API user must have established the appropriate access rights for the affected file. For files *sent*, this is permission to perform the FTAM file actions defined by the *ft_fsend*( ) manual page (see Section 2.3.16 on page 34). For files *received*, permission is controlled by the API user's access rights in the local filestore and is outside the scope of FTAM and the XFTAM API.

2.3.9 Synchronous and Asynchronous Processing Modes

An XFTAM file transfer or management function performs a high-level *operation* (for example, file transfer to a remote filestore) which is *implemented* by a series of low-level *file actions*. By default, when an XFTAM function is called, execution control is passed to the API, which only returns control to the caller once the requested operation invocation has been completed. The result code returned by the function indicates the result of the requested operation (such as [FTE_SUCCESS] if a file transfer succeeded). This is termed *synchronous processing* and is the default mode of execution for XFTAM. In a interrupt handler function the API user can initiate the cancellation of an interrupted synchronous invocation by calling *ft_abandon*( ) and passing the implementation defined constant [FT_CANCEL_SYNC_OP] as the Invoke-ID. The interrupted XFTAM function will then return the error code [FTE_CANCEL].

Where the API user is to perform other activities whilst an XFTAM operation is in progress (for example, an application might initiate several concurrent file transfers), XFTAM provides a second *optional* mode of operation, termed *asynchronous processing*. In this mode the function call returns as soon as its parameters have been validated and the requested operation initiated. The operation then proceeds in the background, with the result being returned later.

Support for this mode of execution is optional because, in some operating system environments, similar functionality is achieved by alternative means (for example, by implementing an asynchronous request as a separate thread of execution, allowing the initiating thread to continue with other tasks). Support is indicated by a non-zero value for the constant FT_MAX_ASYNC_OPS, which indicates the number of asynchronous XFTAM operation invocations that may exist concurrently. A value of FT_MAX_ASYNC_UNLIMITED indicates that the implementation imposes no fixed limit. If an implementation supports asynchronous operation mode, it must support it at least for *ft_fsend*( ) and *ft_freceive*( ). The support for all other functions remains optional. When asynchronous XFTAM operations are permitted, no assumptions can be made regarding the sequence in which the operations are performed.

Attempting to invoke more asynchronous operations than is allowed causes the function to return the error code [FTE_TOO_MANY_OPS]. Attempting to invoke an XFTAM function asynchronously causes the function to return the error code [FTE_NOTSUP_ASYNC] if the implementation doesn't support asynchronous operation either generally (FT_MAX_ASYNC_OPS is defined to be zero) or specific to this function.

The result code returned by an asynchronous function call refers to the result of *initiating* the operation, not the result of the operation invocation itself. Thus the return code [FTE_SUCCESS] in asynchronous processing mode indicates that the requested operation has been successfully invoked. In this case no output object handles are returned for *API-Output-Parameters*, *FTAM-Output-Parameters*, and other output objects returned by XFTAM functions. These objects are returned by a subsequent call to the *ft_rcvresult*( ) function which returns the result of the XFTAM operation invocation.

The asynchronous mode of processing is selected by setting the *Asynchronous* attribute of the *API-Input-Parameters* object to *TRUE*. The function then returns an *invocation identifier* in the *Invoke-ID* attribute of the *API-Output-Parameters* object. This identifier is guaranteed to be unique amongst the concurrent requests outstanding for this particular XFTAM instance and can be matched to the *Invoke-ID* returned by a subsequent call to the *ft_rcvresult*( ) function to indicate that the associated XFTAM operation invocation has completed. It may also be used in a call to the *ft_abandon*( ) function to abandon a particular outstanding operation invocation.

When *ft_rcvresult*( ) returns an *Invoke-ID* that indicates a particular operation invocation has completed, the call also returns the output object handles that would have been returned by the initiating function if it had been executed in synchronous mode. These objects can then be used in the same way as they are for the equivalent synchronous function call, to determine the outcome of the requested operation invocation and to receive any output information. Thus, *ft_rcvresult*( ) returns object handles for the *API-Output-Parameters* and *FTAM-Output-Parameters* objects associated with the completed operation. An additional object handle may be returned in the *result_return* output parameter for those XFTAM operations which return a result object (for example, the *FTAM-Attributes* object returned in the *return_attributes* output parameter of the *ft_frattributes*( ) function).

In summary, the sequence of steps involved when using the asynchronous processing mode are as follows:

1. Call the required XFTAM function, passing it an *API-Input-Parameters* object with the *Asynchronous* attribute set to *TRUE*.
2. Check the function return code to confirm that the asynchronous operation has been invoked successfully.
3. Save the *Invoke-ID* attribute from the *API-Output-Parameters* output object to identify the completion of the operation invocation.
4. Call the *ft_rcvresult*( ) function to receive the *Invoke-ID* of a completed operation invocation. The function returns a completion flag which may be FT_COMPLETED_INVOKATIONS to indicate that a completed invocation has been returned.
5. Process the *API-Output-Parameters* object pointed to by the *op_api_out* parameter to determine the result of the asynchronous operation.
6. Process the *FTAM-Output-Parameters* object pointed to by the *operation_ftam_out* parameter and the output object pointed to by *operation_output_object* as required, depending on the result of the operation.

2.3.10 Context Free and Sensitive Processing Modes

As outlined in Section 2.1 on page 16, XFTAM supports two processing modes: context free and context-sensitive.

To use XFTAM for context free operation, all the parameters needed to carry out identification of the remote file filestore authentication and negotiate service levels (FTAM-input-parameters) are provided with each operation, in addition to those needed to complete the file operation. This is the default mode of operation for both synchronous and asynchronous operations.

However, when a number of XFTAM operations are to be performed with the same FTAM responder it is desirable to avoid the overhead of creating an association for each operation, and provide a consistent context for successive operations. In this case, context-sensitive operation may be used.

To use context-sensitive processing mode, an association is created to the FTAM responder using *ft_connect*( ) (which maps onto the FTAM F-INITIALISE service). This requires the supply of the FTAM-Input-Parameters needed to identify the responder, complete the authentication and negotiate service levels. Upon completion, *ft_connect*( ) returns an Association-ID as an identifier used to identify the association for subsequent operations within the FTAM instance.

To use an existing association, subsequent XFTAM functions supply only the Association-Id as an input parameter, in addition to the parameters needed to complete the file transfer or file management operation.

When operating in context-sensitive mode, XFTAM functions do not use the FTAM F-INITIALISE service, and do not use the values needed for it. Instead, they use the existing association. Upon completion of their function, they do not use F-TERMINATE to destroy the association, but leave it intact. In the same way, *ft_abandon*( ) will abandon any active file operation using the association but will leave the association intact.

An association may be ended in an orderly manner using the *ft_disconnect*( ) function (which uses F-TERMINATE to carry out an orderly close of the association) or it can be destroyed using *ft_abort*( ) (which uses F-U-ABORT) to bring the FTAM regime to an abrupt end.

Support for context sensitive processing mode is mandatory. However, the number of concurrent FTAM associations that may be created by a single XFTAM instance may be limited by an implementation. The error code [FTE_TOO_MANY_ASSOC] is returned when this limit is exceeded.

2.3.11 Interrupt Handling

It is strongly recommended not to call XFTAM functions from within interrupt handler functions. XFTAM implementations are not required to be reentrant. There are only two exceptions:

- *ft_abandon*( ) may be called to terminate an interrupted synchronous operation as described in Section 2.3.9 on page 28. This mechanism enables the API user to install a watch dog timer.
- *ft_close*( ) may be called for workplace shutdown if the API user intends to terminate the XFTAM application. If the interrupt handler function returns control to an interrupted synchronous XFTAM function, this function returns the error code [FTE_SESSION] to indicate, that an workspace shutdown occured.

2.3.12 Session Handle

Where an implementation supports it, the *Session* object, returned by *ft_open*( ) to identify the created XFTAM *instance*, contains a *Session Handle* attribute that can be used in conjunction with the local operating system's asynchronous event handling mechanisms to poll for events relating to a particular XFTAM instance. Where provided, this handle may be used by an application to identify when an outstanding operation completion is available for collection using the *ft_rcvresult*( ) function. The manner in which the handle is used is of course implementation-defined.

2.3.13 XFTAM Result Reporting

There are two levels of result reporting provided by the XFTAM API — *API-level* and *FTAM-level*.

API-Level result reporting

Two types of API-level return codes are defined - *return code* and *vendor code*. A value of the first type is returned as the function value and also as the *Return-Code* attribute of the *API-Output-Parameters* object. The return code is used to indicate the successful completion of an XFTAM function, or return the reason why it failed. It may indicate a failure detected by the API, the underlying FTAM responder or service provider, or by some remote entity. Values are defined by this specification, along with the meaning of the code and a possible action, in Chapter 5 on page 109.

The values returned in the *vendor code* (returned as the *Vendor-Code* attribute of the *API-Output-Parameters* class) are not defined in this specification. The documentation for a particular implementation of the API may define values for the vendor code that provide additional implementation-specific information about why a particular function failed.

FTAM-level result reporting

This level of result reporting is specified by FTAM itself and is based upon *diagnostic* structures that may be returned as parameters in FTAM service primitives. XFTAM makes these structures available to the API user as zero or more *Diagnostic* objects returned as attributes of the *FTAM-Output-Parameters* object class. The diagnostic structures may be used to report errors or to return information deemed to be of interest to the FTAM responder. Because a single XFTAM function results in a series of FTAM service primitives being issued, more than one of these structures may be returned from a function

2.3.14 VFS Mapping

XFTAM is responsible for accessing the local file system when transferring files to and from a remote filestore. In order to do this it performs a mapping between the FTAM VFS representation of a file and that of the local filestore. Two mappings are defined, one for files received and one for files sent.

Mapping Received Files

When receiving a file, XFTAM maps the contents and attributes into the semantics defined by the local filestore. For text files (*FTAM-1* and *FTAM-2*), this involves mapping the *strings* of the FTAM file into lines of text in the local file, using the local convention for representing the end-of-line. For implementations which support the optional *FTAM-2* document type, the mechanism for representing the records of the file in the local filestore is a local matter, not defined by XFTAM. See *ft_freceive*( ) on page 92 for detailed information about mapping received files into the local filestore.

Mapping File Sent

For files sent, by default a file is transferred as an FTAM-1 file with the text lines from the local file being transferred as individual FTAM file *strings*. If the content of the local file means that the default handling is not appropriate, the API user must use the file attribute *content type* to specify an alternative document type, or override the default document type parameters. See *ft_fsend*( ) on page 97 for detailed information about mapping sent files from the local filestore.

2.3.15 FTAM Attributes

The FTAM VFS defines a set of file attributes that are used to describe the characteristics and contents of a file (see Section 1.2.4 on page 7). As an FTAM initiator, an implementation of XFTAM supports all file attribute groups. However the capabilities of the local filestore and remote responders may limit the attributes that can be specified or returned for a file (see below for individual cases).

XFTAM provides the object class *FTAM-Attributes* to pass FTAM file attribute information to and from the API. This class includes an XOM attribute for each file attribute defined by FTAM. Where the FTAM attribute consists of a vector of values, multiple values of the equivalent XOM attribute type may appear in an *FTAM-Attributes* object. Absence of an XOM attribute in the FTAM-Attributes object implies that the equivalent FTAM file attribute is also absent.

As described in Section 1.2.4 on page 7, FTAM attributes are divided into groups. A particular FTAM responder must support the kernel group of attributes but support of the other groups is optional. When XFTAM establishes an FTAM regime with a remote responder, it negotiates which attribute groups are to be supported for this association. The remote responder is entitled to reject support of any attribute group other than the kernel group. If the underlying FTAM service provider fails to negotiate the use of an FTAM attribute group, attempts to specify an attribute from an unsupported group causes the function in question to return an error.

It is possible to perform a file transfer to or from the local filestore without having to deal with FTAM file attributes directly. However, most of the XFTAM functions either accept or return attribute information. The following description highlights salient aspects of the way that some of the XFTAM functions utilise the *FTAM-Attributes* object.

ft_fcattributes( )

The class *New-Attributes*, a sub-class of *FTAM-Attributes*, is used to specify the modified file attributes to be set for a file in a remote filestore. The *FTAM-Attributes* class is used to return the resulting values which the remote responder sets for the file.

Specification of values for those file attributes not directly settable by an FTAM initiator (for example: *date and time of creation* or *identity of creator*), and those only settable at file creation time, causes the function to fail, as does specifying attributes from groups not supported on this association. The *Access-Control-List* attribute of the *FTAM-Attributes* object class is used to specify *additional* conditions to be added to the file's current list. An additional OM attribute, *Delete-Access-Control-List*, is used to delete existing conditions from the list. Deletion of a condition only occurs if all attributes specified for a condition in this call match those of an existing condition. The output attributes are those returned by the remote responder as a result of modifying the files attributes. These can be interrogated to determine the effect of the change attributes request. Return of attribute information as a result of the change attribute file action is optional - some responders may not do so.

ft_frattributes( )

For this function, the *FTAM-Attributes* class is used to return the current attributes for the specified destination file. On input, a related object class, *FTAM-Attribute-Names*, can be used to specify which file attribute values are to be returned by the function. Absence of this object implies that all attributes of the negotiated attribute groups are to be read.

ft_freceive( )

On input, only the XOM *Content-Type* attribute of the *FTAM-Attributes* class may be set. This is used to specify what type of file the API user is expecting to receive and causes the transfer to fail if the document type of the source file does not match exactly the specified type. On output, the FTAM-Attributes class is used to return the actual filename (which may differ from that specified in the request) and the actual content type of the file accessed.

ft_fsend( )

For *ft_fsend*( ), the *FTAM-Attributes* class is used both to specify the initial attribute values proposed for the file being created in the remote filestore, and to return the actual values which the remote responder has set for the file.

For the input attributes, the presence of those file attributes not directly settable by an FTAM initiator (for example: *date and time of creation* or *identity of creator*) cause the function call to fail, as does specifying attributes from groups not supported on this association. In some cases FTAM defines a default action for the responder when an attribute value is not passed. Apart from the exceptions noted below, values of file attributes not included in the *FTAM-Attributes* object are not passed to the remote responder:

- **Filename**.
  The filename for the created file is specified using the *dest_filename* parameter to the function. The equivalent XOM attribute in the *FTAM-Attributes* class is ignored if present.

- **Content type**.
  If not specified by the user, XFTAM assumes a default *Document Type* of *FTAM-1* as specified in the *ft_fsend*( ) manual page -see *ft_fsend*( ) on page 97. This means that the file is assumed to contain lines of text (terminated by the local end-of-line convention). If the user wishes to transfer a file containing binary data or text which is not arranged into lines (for example, a character screen image), an appropriate alternative document type should be specified using this FTAM file attribute.

The output attributes are those returned by the remote responder as a result of creating the file. On output, the FTAM-Attributes class may be used to report local modifications performed by the responder to the filename of the file created, and its permitted actions attribute.

ft_frdir()

This function returns an object of class *Directory-List* which contains one *FTAM-Attribute* object per file in the specified remote directory. Only values of attributes supported by the remote responder for the *NBS-9* document type are returned.

2.3.16 Access Control

FTAM provides three related mechanisms for control of file access. These are access control file attributes, filestore access control, and access passwords. They are discussed individually below.

Access Control File Attribute

The *access control* file attribute provides a set of conditions which may be attached to each of the FTAM actions permitted for a file. A condition consists of an optional password which may be attached to the action, which must be supplied by an initiator wishing to perform the action. In addition the responder may limit *who* may perform the action (*identity*) and *from where* (*location*). (Use of a particular identity may in turn be controlled by the *filestore password* as described below.)

This attribute may be set when the file is created, and modified subsequently using the change attributes operation. XFTAM provides the *Access-Control* object class to set and retrieve the value of this file attribute. Figure 2-4 shows how a list of such objects can be used to specify a set of access conditions for a file, allowing anyone running the application identified by the *application entity title* ‘‘Milkwood’’ and possessing the appropriate passwords to access its contents or attributes, but limiting file modification actions to user ‘‘Captain Cat’’.

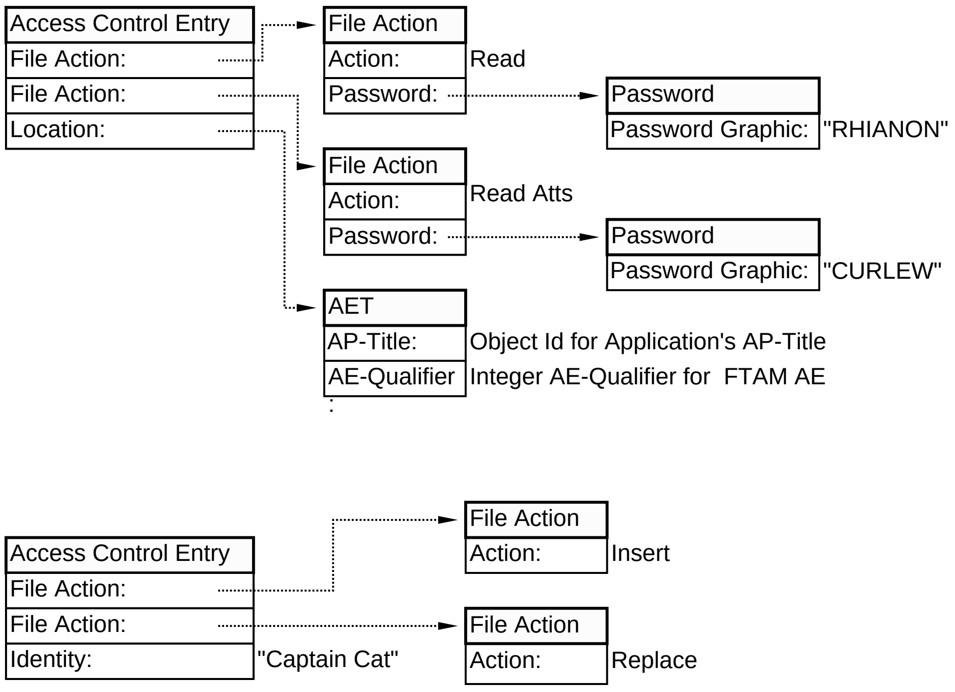

Figure 2-4 Access Control Attribute Example OM Objects

Filestore Access Control

When an FTAM service Provider establishes an FTAM regime with a remote responder, it may optionally specify an *initiator identity* and optional *filestore password*. These FTAM parameters define *who* is requesting the access and authenticate that this is a valid use of the specified identity. FTAM does not specify how the identity and password are used - that is up to the remote responder. (For example, a responder on a CAE conformant system might use the FTAM identity parameter as the login user name, with the associated password matched against the FTAM filestore password parameter.)

XFTAM allows the API user to specify values for these FTAM parameters using the *Initiator-Identity* and *Filestore-Password* attributes of the *FTAM-Input-Parameters* object class.

Access Passwords

When FTAM service provider identifies a file in the remote responder's filestore that is to be the target of an operation, it lists the set of file actions it is going to perform (the content of this list depends upon the operation implemented by the particular XFTAM function called). At the same time it may supply a list of *access passwords* associated with the action list. These passwords are subsequently used by the responder to authorise the requested actions in the case where the selected file's access control file attributes requires a password to be supplied for a particular action.

XFTAM allows the API user to specify values for this FTAM parameter using the *File-Action-List* attribute of the *FTAM-Input-Parameters* object class. The individual functions identify the file actions that are performed as part of the operation implemented.

Chapter 3

XFTAM Base Package - XOM Class Definitions

This chapter defines the XOM classes that make up the XFTAM Base Package. The classes are listed in alphabetical order, in the form defined by the referenced XOM specification (summarised in Appendix A, **Summary of XOM**). To avoid repetitive detail, an XOM class is assumed to be a sub-class of the XOM class *Object* unless otherwise stated. Similarly, an XOM class is assumed to be a *concrete class* unless otherwise stated.

3.1 Package Definition

This specification defines a single XOM package which contains all the classes required by the XFTAM API. The package is known as the *XFTAM Base Package*, and is uniquely identified to XOM by the object identifier:

ISO(1) National Body Member (2) BSI(826) DISC(0) X/Open(1050) XFTAM API(2) Base Package(1)

The class hierarchy for the XFTAM Base package is given below, using indentation to show the relationship between classes. Classes listed in bold type are *concrete classes*, whilst those in normal type are *abstract classes*. Those marked "(Read Only)" are used by XFTAM to return information to the user and may not be created or modified by the API user.

- **Access-Control-Element**
- **AE-Title**
- **API-Input-Parameters**
- **API-Output-Parameters** (Read Only)
- **Charging** (Read Only)
- Content-Type
  - **Document-Type**
    - Document-Type-NBS-9 (Read Only)
    - Document-Type-Text
      - **Document-Type-FTAM-1**
      - **Document-Type-FTAM-2**
    - Document-Type-Binary
      - **Document-Type-FTAM-3**
- **Directory-List** (Read Only)
- **File-Action**
- **FTAM-Attribute-Names**
- **FTAM-Attributes**
  - **New-Attributes**
- **FTAM-Diagnostic** (Read Only)
- **FTAM-Input-Parameters**
- **FTAM-Output-Parameters** (Read Only)
- **Presentation-Address** (See note)
- **Session** (Read Only)

Note: **Presentation-Address** is defined in the **Directory Services (DS)** package defined in the referenced XDS specification.

3.2 Access-Control-Element

This XOM class specifies a single condition under which access to a file is valid. The condition consists of a list of permitted FTAM file actions and a number of optional *terms* which must be satisfied if the identified actions are to be allowed.

XOM Attribute	**Value Syntax**	**Value Length**	**Value Number**	**Value Initially**
File-Action-List	Object(File-Action)	-	1 or more	-
Identity	String(Graphic)	-	0-1	-
Location	Object(AE-Title)	-	0-1	-

OM Attributes of an Access-Control-Element object

File-Action-List
A set of zero or more *File-Action* objects, each identifying an FTAM file action permitted on this file, along with an optional access password and concurrency key for that action. Each FTAM file action should appear once in this list at most, additional occurrences are ignored.

Identity
Optional XOM attribute indicating the *current initiator identity* activity attribute value for which the identified file actions are to be permitted on this file.

Location
Optional *AE-Title* object indicating the *current calling application entity title* activity attribute value for which the identified file actions are to be permitted on this file.

3.3 AE-Title

This XOM class is used to pass the collection of naming parameters (defined by the ACSE service as an *AE-Title* for the convenience of other application layer standards) to XFTAM. An object of this class may appear in an *Access-Control-Element* object to specify a *location* from which the identified FTAM actions may be performed.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
AP-Title	String(Object-Identifier)	-	0-1	-
AE-Qualifier	Integer	-	0-1	-

OM Attributes of an AE-Title object

AP-Title
Optional object identifier that identifies a particular *application process*.

AE-Qualifier
Optional integer that identifies a particular *application entity* of the application process.

3.4 API-Input-Parameters

This XOM class is used to pass API-specific parameters to an XFTAM function call.

XOM Attribute	**Value Syntax**	**Value Length**	**Value Number**	**Value Initially**
Association-ID	integer	-	0-1	-
Asynchronous	Boolean	-	0-1	-

OM Attributes of an API-Input-Parameters object

Association-Id

For context-sensitive operating mode, this attribute is used to identify an association which is to be used to perform the operation. The value of the attribute is the integer returned by *ft_connect*( ) when creating the association.

If this parameter is not present, XFTAM operates in context free mode and creates and destroys an association to perform the requested operation.

If this parameter is present, context-sensitive processing mode is selected and XFTAM does not require the p_address parameter and FTAM-Input-Parameters which relate to establishing an FTAM regime. If these parameters are supplied, an error code is returned [FT_CONTEXT_MISMATCH].

Supplying a value for *Association-Id* which is not valid for the identified XFTAM instance (does not represent an existing association) causes an error code [FT_INV_ASSOC].

Asynchronous

If *TRUE*, the associated XFTAM operation is to be executed *asynchronously*. In this case an *Invoke-ID* attribute is returned by the called function in the *API-Output-Parameters* object. If this attribute is not present, or is *FALSE*, the function is executed *synchronously*.

An implementation's support of asynchronous operations is indicated by a non-zero value for the constant FT_MAX_ASYNC_OPS, which indicates the maximum number of asynchronous operations that may be outstanding at any one time. Attempting to invoke more asynchronous operations than is allowed causes the function to return the error code [FTE_TOO_MANY_OPS]. Attempting to invoke an XFTAM function asynchronously causes the function to return the error code [FTE_NOTSUP_ASYNC] if the implementation doesn't support asynchronous operation either generally (FT_MAX_ASYNC_OPS is defined to be zero) or specific to this function.

3.5 API-Output-Parameters

This XOM class is used to return API-specific output parameters from an XFTAM function call.

XOM Attribute	**Value Syntax**	**Value Length**	**Value Number**	**Value Initially**
Return-Code	Integer	-	1	-
Vendor-Code	Integer	-	0-1	-
Invoke-ID	Integer	-	0-1	-

OM Attributes of an API-Output-Parameters object

Return-Code
The result code for an XFTAM function, the values for this XOM attribute are listed in Chapter 5. Each XFTAM function lists its possible return codes.

Vendor-Code
An optional, implementation-defined return code which further qualifies the result of a function. These codes may be defined in the documentation accompanying a particular implementation.

Invoke-ID
This attribute is returned if the API user requests that the XFTAM operation associated with the called function is to be executed in *asynchronous* mode. The returned ID may be matched against *Invoke-ID* values returned by subsequent calls to *ft_rcvresult*( ) to determine when the associated XFTAM operation has completed. This attribute is only returned if the *Return-Code* attribute is [FTE_SUCCESS].

3.6 Charging

This XOM class is mapped from the FTAM *charging* parameters which may be returned by a number of FTAM confirmation service primitives, it conveys information on the costs incurred during an FTAM regime.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Resource-Identifier	String(Graphic)	-	1	-
Charging-Unit	String(Graphic)	-	1	-
Charge-Value	Integer	-	1	-

OM Attributes of a Charging object

Resource-Identifier
A string identifying the resource for which a charge is being levied. The contents of this string are specific to the responder implementation accessed.

Charging-Unit
The units in which the *Charge-Value* are measured. The contents of this string are specific to the responder implementation accessed.

Charge-Value
The number of *Charge-Units* that have been accumulated for use of the identified resource.

3.7 Content-Type

Identifies the abstract data types of an FTAM VFS file's contents plus other structuring information necessary to maintain its structure and semantics.

FTAM defines two forms in which this information can be specified. XFTAM supports only the *Document Type* form. *Content-Type* is an abstract XOM class with no specific XOM attributes.

There is only one sub-class defined: *Document-Type*.

3.8 Directory-List

This XOM class is used by the *ft_frdir*( ) to return a list of directory entries, each of which is an *FTAM-attributes* object listing the FTAM file attributes for a single file.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Directory-Entry	Object(FTAM-Attributes)	-	0 or more	-

OM Attributes of a Directory-List object

Directory-Entry
This XOM Attribute is used to return the FTAM file attributes for a single file in the source directory.

3.9 Document-Type

This is XOM class is a sub-class of *Content-Type*. It represents the *Document Type* form of the equivalent FTAM file attribute. Sub-classes are defined for each supported document type. Two abstract sub-classes are currently defined in this specification representing *generic* document types which have parameters in common, these are *Document-Type-Text* and *Document-Type-Binary*.

In addition, an object of this class may be returned by the *ft_frattributes* and *ft_frdir*( ) functions when returning FTAM file attribute information for files of a type not supported by XFTAM (that is, no information is returned about the values of any parameters that may be associated with an un-supported document type).

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Type-Name	String(Object-Identifier)	-	1	-

OM Attributes of a Document-Type object

Type-Name
The object identifier of the selected document type, defined in the FTAM specification and elsewhere.

3.10 Document-Type-Text

This is an *abstract* XOM class which is a sub-class of the *Document-Type* XOM class. It represents a generic document type *text*, which defines XOM attributes for the parameters common to the FTAM text document types (*FTAM-1* and *FTAM-2*). Sub-classes are defined for the *concrete* classes *Document-Type-FTAM-1* and *Document-Type-FTAM-2*.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Content-Class	Enum(Universal-Class)	-	0-1	-
String-Length	Integer	-	0-1	-
String-Significance	Enum(String-Significance)	-	0-1	-

OM Attributes of a Document-Type-FTAM-1 object

Content-Class

This XOM attribute defines the character set (Universal Class) from which the file's strings are formed. It can take the following values:

— Printable-String (Optional)

— Teletex-String (Optional)

— Videotex-String (Optional)

— IA5-String (Mandatory for FTAM-1 files)

— Graphic-String (Mandatory)

— Visible-String (Optional)

— General-String (Mandatory for FTAM-1 files and Optional for FTAM-2 files).

If this XOM attribute is not present, the character set is *General-String*.

Note that for file transfer operations, support of IA5-String and *General-String* is mandatory for FTAM-1 files, and support of *General-String* is mandatory for FTAM-2 files. Support of the other classes is optional in each case. That is, some implementations may not allow these values to be specified for files being sent, for files being received, the transfer may fail if the source file specifies one of these values as its *universal class number*.

String-Length

Maximum length of a file's strings. If this XOM attribute is omitted, the string length for this file is unlimited. The *minimum* value for the maximum-string-length that an XFTAM implementation must support is 134 characters for the FTAM-1 and FTAM-2 document types. The [FTE_INV_STRING_LENGTH] error is returned if the user attempts to send or receive a document which exceeds the maximum supported value for this parameter.

String-Significance

This parameter defines the significance of the string length for this file:

— Variable - the length of the character strings is less than or equal to *String-Length*.

— Fixed - the length of the character strings is exactly equal to *String-Length*.

— Not-Significant - the boundaries of the character strings is not necessarily preserved when the file is stored.

If this XOM attribute is omitted, the string significance for this file is *not significant*.

3.11 Document-Type-FTAM-1

This XOM class is a sub-class of the *Document-Type-Text* class.

The value of the *Type-Name* attribute, inherited from the *Document-Type* superclass, is ‘‘1 0 8571 5 1’’. No additional attributes are defined.

3.12 Document-Type-FTAM-2

This XOM class is a sub-class of the *Document-Type-Text* class.

The value of the *Type-Name* attribute, inherited from the *Document-Type* superclass, is ‘‘1 0 8571 5 2’’. No additional attributes are defined.

Support for this document type is optional. Setting the *Content Type* XOM attribute to point to an object of this type may cause an XFTAM function to return the error code [FTE_NOTSUP_FTAM2].

3.13 Document-Type-Binary

This is an *abstract* XOM class which is a sub-class of the *Document-Type* class. It represents a generic document type *binary*, which defines XOM attributes for the parameters common to the FTAM binary document types (*FTAM-3* and *FTAM-4*). A sub-class is defined for the *concrete* class *Document-Type-FTAM-3* only, (*FTAM-4* is not supported by XFTAM).

XOM Attribute	**Value Syntax**	**Value Length**	**Value Number**	**Value Initially**
String-Length	Integer	-	0-1	-
String-Significance	Enum(String-Significance)	-	0-1	-

OM Attributes of a Document-Type-FTAM-3 object

String-Length

Maximum length of this file's strings. If this XOM attribute is missing, the string length for this file is unlimited. The *minimum* value for the maximum-string-length that an XFTAM implementation must support is 512 octets for the FTAM-3 document type. The [FTE_INV_STRING_LENGTH] error is returned if the user attempts to send or receive a document which exceeds the maximum supported value for this parameter.

String-Significance

This parameter defines the significance of the string length for this file:

— Variable - the length of the character strings is less than or equal to *String-Length*.

— Fixed - the length of the character strings is exactly equal to *String-Length*.

— Not-Significant - the boundaries of the character strings is not necessarily preserved when the file is stored.

If this XOM attribute is omitted, the string significance for this file is not significant.

3.14 Document-Type-FTAM-3

This XOM class is a sub-class of the *Document-Type-Binary* class.

The value of the *Type-Name* attribute, inherited from the *Document-Type* superclass, is ‘‘1 0 8571 5 3’’. No additional attributes are defined.

3.15 Document-Type-NBS-9

This XOM class is a sub-class of the Document-Type Class.

The value of the Type-Name attribute, inherited from the Document-Type superclass, is ‘‘1 3 14 5 5 9’’. No additional attributes are defined.

This document type may be returned as the Content-Type attribute of a file, by the *ft_frattributes*( ) and *ft_frdir*( ) functions, in the case where the identified file is directory.

Such a file may not be the subject of a file transfer request. An attempt to transfer such a file will result in the error code [FTE_INV_DOC_RCVD] or [FTE_INV_DOC_SPEC] being returned, as appropriate.

3.16 File-Action

This XOM class is used to identify an FTAM file action and an optional password and concurrency lock associated with that action. It is used as part of the *Access-Control-Element* class to specify conditions which must be satisfied for a file access to be valid. It may also appear as an attribute of the *FTAM-Input-Parameters* class to specify the passwords and concurrency keys used to validate a file access.

	Value	**Value**	**Value**	**Value**
XOM Attribute	**Syntax**	**Length**	**Number**	**Initially**
File-Action	Enum(File-Actions)	-	1	-
Conc_Key	Enum(Concurrency-Key)	-	0-1	-
Access-Password	Object(Password)	-	0-1	-

OM Attributes of a File-Action object

File-Action

Identifies a single FTAM file action. Possible values are:

— Read

— Insert

— Replace

— Extend

— Erase

— Read-Attribute

— Change-Att

— Delete-File.

Concurrency-Key

Where the *File-Action* object is used as a sub-object of an *Access-Control-Element* object, this optional attribute may be used to specify a concurrency lock to be associated with the target file for the identified action. The specified lock must then be acquired as part of a subsequent attempt to perform this action on the target file.

Alternatively, where the *File-Action* object is used as a sub-object of an *FTAM-Input-Parameters* object, this optional attribute may be used to specify the lock to be acquired in the case where it is required in order to perform the identified action on the target file:

— Not Required

— Shared

— Exclusive

— No Access.

Access-Password

An object of class *Password*.

Where the *File-Action* object is used as a sub-object of an *Access-Control-Element* object, this optional attribute may be used to specify a password to be associated with the target file for the identified action. The specified password must then be supplied as part of a subsequent attempt to perform this action on the target file.

Alternatively, where the *File-Action* object is used as a sub-object of an *FTAM-Input-Parameters* object, this optional attribute may be used to specify a password in the case where one must be supplied in order to perfom the identified action on the target file.

3.17 FTAM-Attribute-Names

This XOM class is used to identify the FTAM file attributes that are to be returned from a call to *ft_frattributes*( ).

XOM Attribute	**Value Syntax**	**Value Length**	**Value Number**	**Value Initially**
FTAM-Attribute-Name-List	Enum(FTAM-Attribute-Name)	-	0 or more	-

OM Attributes of an FTAM-Attribute-Names object

FTAM-Attribute-Name-List

A set of one or more attribute names, each identifying an attribute that is to be returned by an XFTAM function call. Values are:

— Filename

— Permitted-Actions

— Content-Type

— Storage-Account

— Creation-Date-Time

— Modification-Date-Time

— Read-Date-Time

— Attribute-Mod-Date-Time

— Creator-Identity

— Modifier-Identity

— Reader-Identity

— Attribute-Mod-Identity

— File-Availability

— Filesize

— Future-Filesize

— Access-Control

— Legal-Qual

— Private-Use.

3.18 FTAM-Attributes

This XOM class is used to pass FTAM file attribute values between the API user and XFTAM. It has one XOM attribute per FTAM file attribute. Where an object of this class is passed to an XFTAM function, the description of the function may define which attributes are valid for the function and any additional constraints upon the values of the object's attributes.

This class has a sub-class, *New-Attributes*, which adds an additional XOM attribute required for the *ft_fcattributes( )* function.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Filename	String(Graphic)	-	0-1	-
Permitted-Actions	Enum(Permitted-Action)	-	0 or more	-
Content-Type	Object(Content-Type)	-	0-1	-
Storage-Account	String(Graphic)	-	0-1	-
Creation-Date-Time	String(Generalised Time)	-	0-1	-
Modification-Date-Time	String(Generalised Time)	-	0-1	-
Read-Date-Time	String(Generalised Time)	-	0-1	-
Attribute-Mod-Date-Time	String(Generalised Time)	-	0-1	-
Creator-Identity	String(Graphic)	-	0-1	-
Modifier-Identity	String(Graphic)	-	0-1	-
Reader-Identity	String(Graphic)	-	0-1	-
Attribute-Mod-Identity	String(Graphic)	-	0-1	-
File-Availability	Enum(File-Availability)	-	0-1	-
Filesize	Integer	-	0-1	-
Future-Filesize	Integer	-	0-1	-
Access-Control-List	Object(Access-Control-Element)	-	0 or more	-
Legal-Qualifications	String(Graphic)	-	0-1	-
Private-Use	Object(External)	-	0-1	-

OM Attributes of an FTAM-Attributes object

Filename
The name of a file in the FTAM filestore. XFTAM supports only single-component filenames.

Permitted-Actions
A set of zero or more FTAM file actions and FADU-identity styles that are permitted for the file. Values are:

— Read

— Insert

— Replace

— Extend

— Erase

— Read-Attribute

— Change-Att

— Delete-File

— Traversal

— Reverse-Traversal

— Random-Order.

Content-Type
Identifies the abstract data types of the file's contents plus other structuring information necessary to maintain its structure and semantics. An object of class *Content-Type*.

Storage-Account
The accountable authority responsible for accumulating file storage charges.

Creation-Date-Time
Indicates when the file was created.

Modification-Date-Time
Indicates when the contents of the file were last modified.

Read-Date-Time
Indicates when the contents of the file were last read.

Attribute-Mod-Date-Time
Indicates when an attribute of the file was last modified.

Creator-Identity
Indicates the *initiator identity* activity attribute when the file was created.

Modifier-Identity
Indicates the *initiator identity* activity attribute when the contents of the file were last modified.

Reader-Identity
Indicates the *initiator identity* activity attribute when the contents of the file were last read.

Attribute-Mod-Identity
Indicates the *initiator identity* activity attribute when an attribute of the file was last modified.

File-Availability
Indicates whether a delay should be expected before the file can be opened. Values are:

— Immediate-Availability - meaning no delay is expected.

— deferred-Availability - meaning that the file MAY be stored on a demountable device.

Filesize
Nominal size in octets of the file when it was last closed.

Future-Filesize
Nominal size in octets to which the file may grow as a result of modification and extension.

Access-Control-List
A set of zero or more access control elements, each specifying a single condition under which access to the file is valid. Objects of class *Access-Control-Element*.

Legal-Qualifications
Conveys information about the the legal status of the file and its use.

Private-Use
The meaning of this attribute is not defined.

3.19 FTAM-Diagnostic

This XOM class is mapped from the FTAM *diagnostic* parameter returned by most FTAM confirmation service primitives, providing information about the result of the collection of FTAM actions performed as part of the high-level XFTAM operation.

	Value	**Value**	**Value**	**Value**
XOM Attribute	**Syntax**	**Length**	**Number**	**Initially**
Error-Type	Enum(FTAM-Error-Type)	-	1	-
Error-Identifier	Integer	-	1	-
Observer	Enum(FTAM-Entity)	-	1	-
Source	Enum(FTAM-Entity)	-	1	-
Suggested-Delay	Integer	-	0-1	-
Text-Message	String(Graphic)	-	0-1	-

OM Attributes of a Diagnostic object

Error-Type

The type of error being reported by this diagnostic. *Error types* are defined by the FTAM specification. Possible values are:

— Information

— Transient-Error

— Permanent-Error.

Error-Identifier

An integer identifying a specific error and error class. *Error identifiers* and *Error Classes* are defined by the FTAM specification. They are listed in the **xftam.h** header file.

Observer

The type of entity which detected the error. *Error observers* are defined by the FTAM specification. Possible values are:

— Initiating-User - the initiating file service user.

— Initiating-FPM - the initiating file protocol machine.

— Responding-User - the responding file service user (filestore).

— Responding-FPM - the responding file protocol machine.

Source

The type of entity which is believed to have caused the error. Error sources are defined by the FTAM specification. Possible values are those listed above for the *Observer* XOM attribute, plus the following:

— None - no categorisation possible.

— Supporting-Service - service supporting the file protocol machines.

Suggested-Delay

Where the *Error-Type* is *Transient-Error*, an optional period that should elapse before the action is tried again. For a value of ‘‘x’’, the suggested delay is ‘‘2 to the power x’’.

Text-Message

An optional text message in natural language.

3.20 FTAM-Input-Parameters

This XOM class groups together FTAM-specific input parameters that are common to a number of different XFTAM functions or whose values may remain constant over a series of calls to XFTAM functions. All of these parameters are optional unless otherwise specified in the description of an XFTAM function call. Individual XFTAM function descriptions may define specific requirements or significance for some of these parameters.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Initiator-Identity	String(Graphic)	-	0-1	-
Filestore-Password	Object(Password)	-	0-1	-
Create-Password	Object(Password)	-	0-1	-
Account	String(Graphic)	-	0-1	-
FQos	Enum(FQoS)	-	0 or more	-
File-Action-List	Object(File-Action)	-	0 or more	-

OM Attributes of an FTAM-Input-Parameters object

Initiator-Identity
This optional XOM attribute is used to identify the calling user to the FTAM responder.

Filestore-Password
This optional XOM attribute is an object of type *Password*, used to authenticate the calling user to the FTAM responder. FTAM defines the equivalent parameter as either *OctetString* or *GraphicString*.

Create-Password
This optional XOM attribute is an object of type *Password*, used to authorise the calling user to create a file in the called filestore. FTAM defines the equivalent parameter as either *OctetString* or *GraphicString*.

Account
This optional XOM attribute is used as the account to which costs incurred in the FTAM regime to be used by this XFTAM request are charged. For context-sensitive file transfer or file management operations, this parameter may be used to override the account context established when the association was created by *ft_connect*( ). In this case, such charges are returned when the operation completes.

FQoS
This optional XOM attribute is used to indicate what level of recovery should be attempted by FTAM in the case of errors during an XFTAM file transfer operation. For file management operations, any value of the FQoS attribute other than No-Recovery will be ignored. Possible values are:

— No-Recovery - do not attempt recovery, any error is reported as a *permanent error*.

— Class-1-Recovery - recover from errors which damage the data transfer regime.

— Class-2-Recovery - recover from errors which damage the select or open regimes.

— Class-3-Recovery - recover from errors which lose the association supporting the FTAM regime.

If this attribute is not present, the requested operation is performed with a FQoS of No-Recovery.

The user may supply multiple values for this attribute where a number of levels of recovery are acceptable. The underlying FTAM initiator requests the highest level of recovery which has been specified by the user and which it supports. If none of the specified FQoS values are supported by the underlying FTAM service, the XFTAM operation fails with an error code of [FTE_NOTSUP_FQOS]. Where the requested FQoS is reduced by the underlying FTAM service during negotiation of the FTAM regime and the resulting level is not one of those requested by the user in this attribute, the association is terminated and the XFTAM operation fails with an error code of [FTE_FQOS_NOT_NEGOTIATED]. Otherwise, the operation proceeds using the negotiated FQoS value.

File-Action-List.
This optional XOM attribute is a set of zero or more *File-Action* objects, used to specify passwords and concurrency locks required for the *file actions* to be performed on the target file. Each XFTAM function identifies the FTAM file actions which it performs, for which the accessed file may require the API user to supply passwords or specify concurrency locks. Each FTAM file action should appear once in this list at most; subsequent occurrences are ignored.

3.21 FTAM-Output-Parameters

This XOM class groups together FTAM-specific output parameters that are common to a number of XFTAM functions.

	Value	Value	Value	Value
XOM Attribute	**Syntax**	**Length**	**Number**	**Initially**
Charging-List	Object(Charging)	-	0 or more	-
FTAM-Result	Enum(FTAM-Result)	-	1	-
FTAM-Diagnostic-List	Object(FTAM-Diagnostic)	-	0 or more	-

OM Attributes of an FTAM-Output-Parameters object

Charging-List

This XOM attribute provides information on the costs attributed to the account during the file action. Zero or more charging objects may be returned, each detailing charges for a particular resource. Responders for CAE-conformant file-stores may not return charging information. For context-sensitive processing mode, file transfer or file management operations only return charges when an override account has been specified in the Ftam-Input-Parameters. All connection charges are returned when the association is destroyed by *ft_disconnect*( ).

FTAM-Result

This XOM attribute is mapped from the FTAM *action result* parameter and returns the overall result of the series of FTAM actions which were performed as part of the XFTAM function. Possible values are:

— Success

— Permanent-Error.

If *Permanent-Error* is returned, one of the FTAM actions performed as part of the XFTAM function has failed. Exactly which action has failed may not be precisely identifiable; however the *error identifier* of the FTAM *diagnostic* parameter includes an *error classification* which may provide clarification, and the *Return-Code* XOM attribute of *API-Output-Parameters* object may provide further information. If *Success* is returned, Diagnostics-List may contain informational FTAM *diagnostic* parameters returned by one or more of the FTAM actions performed.

FTAM defines a third value for the *action result* parameter - *Transient Error*. However, this value is for use by the FTAM Error Protocol and is never returned to FTAM External File Service Users. All errors are reported to XFTAM users as *Permanent Error*.

Diagnostic-List

Each occurrence of this XOM attribute is an XOM object of class *Diagnostic*, mapped from the FTAM *Diagnostic* parameter, providing additional information about the result of an FTAM action.

3.22 Password

This XOM class is mapped to the FTAM *password* type, used in a number of FTAM PDUs. It provides the ability to specify a password as either GraphicString or OCTET STRING.

	Value	**Value**	**Value**	**Value**
XOM Attribute	**Syntax**	**Length**	**Number**	**Initially**
Password-Octet	String(Octet)	-	0-1*	-
Password-Graphic	String(Graphic)	-	0-1*	-

* Both attributes are *optional*, but one or other must be present in a valid instance of this class. A valid instance of this class may not contain both attributes.

OM Attributes of a Password object

Password-Octet
This XOM attribute maps to the OCTET STRING form of an FTAM password.

Password-Graphic
This XOM attribute maps to the GraphicString form of an FTAM password.

3.23 New-Attributes

This XOM class is a sub-class of the *FTAM_Attributes* class, used to pass modified FTAM file attribute values to the *ft_fcattributes( )* function. In addition to the attributes inherited from the FTAM-Attributes object class, it has one specific attribute which specifies a set of access control conditions that are to be deleted.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Delete-Access-Control-List	Object(Access-Control-Element)	-	0 or more	-

OM Attributes of an FTAM-Attributes object

Delete-Access-Control-List
A set of zero or more access control elements to be deleted from the target file's *access control* attribute. FTAM requires that a specified condition exactly matches one in the current attributes if the delete is to succeed. Each element of the list is an object of class *Access-Control-Element*.

3.24 Session

This XOM class is used to identify a particular XFTAM *instance*. Instances of this object are read-only, created by the *ft_open*( ) function and destroyed by the *ft_close*( ) function.

XOM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Session-Handle	Integer	-	0-1	-

OM Attributes of a Session object

Session-Handle

An optional attribute that may be used to return an implementation-specific identifier for use in the local asynchronous event handling mechanisms. (For example, in a UNIX® system, this handle might be a *file descriptor* that is used in subsequent calls to a *poll*( ) or *select*( ) function to wait for events on a number of such file descriptors).

3.25 Declaration Summary

This section lists the C identifiers that are required by an implementation of the XFTAM Base Package for use by applications. Definitions of the identifiers and their values appear in the XFTAM header file **<xftam.h>**.

Whilst it is not required to define actual values for such identifiers in order to enable *portability* of applications, values are defined here in order facilitate *interoperability* amongst APIs which are based upon XOM and which may share the definitions of one or more OM classes. That is, to support an application which wishes to obtain an OM object from one API and pass it to another API (which may be from a different supplier and be supported by a different implementation of the XOM API from that of the first), it is necessary for the implementations to share common definitions of the class and attribute identifiers, and enumeration values used by the shared classes.

3.25.1 Class Identifiers

In the following identifier list, each class in the XFTAM Base Package has an identifier (prefixed with the characters *FTC_*), which defines the ASN.1 object identifier that is used to denote the class. The object identifier is derived from the object identifier that denotes the XFTAM Base Package, by appending to it the integer from the list. Thus the object identifier defined by FTC_ACCESS_CONTROL_ELEMENT is:

1.2.826.0.1050.2.1.1

3.25.2 Attribute Type Identifiers

In order to retain uniqueness amongst the values assigned to attribute types for the APIs defined by X/Open, the attribute types for the XFTAM Base Package have been allocated in the range 5000 - 5099.

3.25.3 C Identifier List

```
FTC_ACCESS_CONTROL_ELEMENT       <Base Package> (1)
    FTA_FILE_ACTION_LIST                 5001
    FTA_IDENTITY                         5002
    FTA_LOCATION                         5003

FTC_AE_TITLE                     <Base Package> (2)
    FTA_AP_TITLE                         5004
    FTA_AE_QUALIFIER                     5005

FTC_API_INPUT_PARAMETERS         <Base Package> (3)
    FTA_ASSOCIATION                      5059
    FTA_ASYNCHRONOUS                     5006

FTC_API_OUTPUT_PARAMETERS        <Base Package> (4)
    FTA_RETURN_CODE                      5007
    FTA_VENDOR_CODE                      5008
    FTA_INVOKE_ID                        5009

FTC_CHARGING                     <Base package> (5)
    FTA_RESOURCE_IDENTIFIER              5010
    FTA_CHARGING_UNIT                    5011
    FTA_CHARGE_VALUE                     5012

FTC_CONTENT_TYPE                 <Base Package> (6)

FTC_DIRECTORY_LIST               <Base Package> (7)
    FTA_DIRECTORY_ENTRY                  5013

FTC_DOCUMENT_TYPE                <Base Package> (8)
    FTA_TYPE_NAME                        5014

FTC_DOCUMENT_TYPE_TEXT           <Base Package> (9)
    FTA_CONTENT_CLASS                    5015
        FT_PRINTABLE_STRING                      1
        FT_TELETEX_STRING                        2
        FT_VIDEOTEX_STRING                       3
        FT_IA5_STRING                            4
        FT_GRAPHIC_STRING                        5
        FT_VISIBLE_STRING                        6
        FT_GENERAL_STRING                        7
    FTA_STRING_LENGTH                    5016
    FTA_STRING_SIGNIFICANCE              5017
        FT_VARIABLE                              1
        FT_FIXED                                 2
        FT_NOT_SIGNIFICANT                       3

FTC_DOCUMENT_TYPE_FTAM_1         <Base Package> (10)

FTC_DOCUMENT_TYPE_FTAM_2         <Base Package> (11)

FTC_DOCUMENT_TYPE_BINARY         <Base Package> (12)

FTC_DOCUMENT_TYPE_FTAM_3         <Base Package> (13)

FTC_FILE_ACTION                  <Base Package> (14)
    FTA_FILE_ACTION                      5018
        FT_READ                                  1
```

```
            FT_INSERT                                  2
            FT_REPLACE                                 3
            FT_EXTEND                                  4
            FT_ERASE                                   5
            FT_READ_ATTRIBUTE                          6
            FT_CHANGE_ATTRIBUTE                        7
            FT_DELETE_FILE                             8
        FTA_CONCURRENCY_KEY                    5019
            FT_NOT_REQUIRED                            1
            FT_SHARED                                  2
            FT_EXCLUSIVE                               3
            FT_NO_ACCESS                               4
        FTA_ACCESS_PASSWORD                    5020

    FTC_FTAM_ATTRIBUTE_NAMES          <Base Package> (15)
        FTA_FTAM_ATTRIBUTE_NAME_LIST           5021
            FT_FILENAME                                1
            FT_PERMITTED_ACTIONS                       2
            FT_CONTENT_TYPE                            3
            FT_STORAGE_ACCOUNT                         4
            FT_CREATION_DATE_TIME                      5
            FT_MODIFICATION_DATE_TIME                  6
            FT_READ_DATE_TIME                          7
            FT_ATTRIBUTE_MOD_DATE_TIME                 8
            FT_CREATOR_IDENTITY                        9
            FT_MODIFIER_IDENTITY                       10
            FT_READER_IDENTITY                         11
            FT_ATTRIBUTE_MOD_IDENTITY                  12
            FT_FILE_AVAILABILITY                       13
            FT_FILESIZE                                14
            FT_FUTURE_FILESIZE                         15
            FT_ACCESS_CONTROL                          16
            FT_LEGAL_QUAL                              17
            FT_PRIVATE_USE                             18

    FTC_FTAM_ATTRIBUTES               <Base Package> (16)
        FTA_FILENAME                           5022
        FTA_PERMITTED_ACTIONS                  5023
            FT_TRAVERSAL                               9
            FT_REVERSE_TRAVERSAL                       10
            FT_RANDOM_ORDER                            11
        FTA_CONTENT_TYPE                       5024
        FTA_STORAGE_ACCOUNT                    5025
        FTA_CREATION_DATE_TIME                 5026
        FTA_MODIFICATION_DATE_TIME             5027
        FTA_READ_DATE_TIME                     5028
        FTA_ATTRIBUTE_MOD_DATE_TIME            5029
        FTA_CREATOR_IDENTITY                   5030
        FTA_MODIFIER_IDENTITY                  5031
        FTA_READER_IDENTITY                    5032
        FTA_ATTRIBUTE_MOD_IDENTITY             5033
        FTA_FILE_AVAILABILITY                  5034
            FT_IMMEDIATE_AVAILABILITY                  1
            FT_DEFERRED_AVAILABILITY                   2
        FTA_FILESIZE                           5035
        FTA_FUTURE_FILESIZE                    5036
        FTA_ACCESS_CONTROL                     5037
        FTA_LEGAL_QUAL                         5038
        FTA_PRIVATE_USE                        5039
```

```
FTC_FTAM_DIAGNOSTIC             <Base Package> (17)
    FTA_ERROR_TYPE                      5040
        FT_INFORMATION                          1
        FT_TRANSIENT_ERROR                      2
        FT_PERMANENT_ERROR                      3
    FTA_ERROR_IDENTIFIER                5041
    FTA_OBSERVER                        5042
        FT_INITIATING_USER                      1
        FT_INITIATING_FPM                       2
        FT_RESPONDING_USER                      3
        FT_RESPONDING_FPM                       4
    FTA_SOURCE                          5043
        FT_NONE                                 5
        FT_SUPPORTING_SERVICE                   6
    FTA_SUGGESTED_DELAY                 5044
    FTA_TEXT_MESSAGE                    5045

FTC_FTAM_INPUT_PARAMETERS       <Base Package> (18)
    FTA_INITIATOR_IDENTITY              5046
    FTA_FILESTORE_PASSWORD              5047
    FTA_CREATE_PASSWORD                 5048
    FTA_ACCOUNT                         5049
    FTA_FQOS                            5050
        FT_NO_RECOVERY                          1
        FT_CLASS_1_RECOVERY                     2
        FT_CLASS_2_RECOVERY                     3
        FT_CLASS_3_RECOVERY                     4
    FTA_FILE_ACTION_LIST                5001

FTC_FTAM_OUTPUT_PARAMETERS      <Base Package> (19)
    FTA_CHARGING_LIST                   5052
    FTA_FTAM_RESULTS                    5053
        /* FT_PERMANENT_ERROR  implied */
        FT_SUCCESS                              4
    FTA_DIAGNOSTIC_LIST                 5054

FTC_PASSWORD                    <Base Package> (20)
    FTA_PASSWORD_OCTET                  5055
    FTA_PASSWORD_GRAPHIC                5056

FTC_NEW_ATTRIBUTES              <Base Package> (21)
    FTA_DELETE_ACCESS_CONTROL_LIST      5057

FTC_SESSION                     <Base Package> (22)
    FTA_SESSION_HANDLE                  5058

FTC_DOCUMENT_TYPE_NBS_9                        (23)
```

Chapter 4

XFTAM Function Manual Pages

This chapter contains a manual page for each of the functions provided by the XFTAM API.

NAME

ft_abandon - abandon an outstanding or interrupted operation

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_abandon(
       OM_private_object   session ,
       OM_sint             invoke_id ,
       OM_private_object   *api_out
);
```

DESCRIPTION

This function cancels an interrupted synchronous function or abandons the result of an asynchronous function invocation. A cancelled synchronous function will return the error code [FTE_CANCEL]. An abandoned asynchronous function is no longer outstanding after *ft_abandon*( ) returns and its results will never be returned by *ft_rcvresult*( ).

Note that this function may cause the associated XFTAM operation to be aborted, or a data transfer to be cancelled. However, XFTAM only guarantees that the result of the operation will not be returned. No statement is made regarding the state of the file(s) and filestore(s) involved in the abandoned operation.

Asynchronous execution mode is an optional feature of XFTAM. If an implementation does not support this feature, *ft_abandon*( ) returns the error code [FT_NO_SUCH_INVOKATION] since the API user cannot pass a valid Invoke-ID.

ARGUMENTS

session (Private Object (Session))

This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

invoke-ID (Integer)

Selects the operation invocation to be aborted. To cancel an interrupted synchronous operation, the implementation defined constant [FT_CANCEL_SYNC_OP] has to be passed. To abandon an asynchronous operation, the value of Invoke-ID must be that which was returned by this function call.

api_out (Private Object (API-Output-Parameters))

This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

ft_abandon( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS.

ERRORS

Operation Error Codes
- FT_NO_SUCH_INVOKATION

API Error Codes
- FTE_VENDOR
- FTE_SESSION

NAME

ft_abort - abort an association and any outstanding operations using it.

SYNOPSIS

```
#include <xftam.h>

ft_return_code ft_abort(
        OM_private_object       session ,
        OM_uint32               association_id ,

        OM_private_object       *api_out
);
```

DESCRIPTION

This function aborts an association and any operation currently in progress on that association. Any such operation is no longer outstanding and its result will never be returned by *ft_rcvresult*( ).

This function differs from *ft_abandon*( ) in that it will destroy the XFTAM association as well as any oustanding operation for this association. As with *ft_abandon*( ), no statement is made regarding the state of the file(s) and filestore(s) involved in the aborted operation.

If the value of *Association-Id* does not represent an existing association within the XFTAM instance referenced by the session parameter, an error code is returned [FT_INV_ASSOC].

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of the class session which identifies the particular XFTAM instance that is to perform the required operation.

association-ID (OM_uint32)
This parameter is an integer which identifies the association to be aborted.

The value of Association-Id must be that returned by *ft_connect*( ) when the association was created.

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for the private object of class *API-Output-Parameters*. It returns API-specific output parameters for this function call.

RETURN VALUES

ft_abort( ) returns either [FTE_SUCCESS] or one of the values listed in ERRORS.

ERRORS

Operation error codes
FT_INV_ASSOC

API Error Codes
FTE_VENDOR
FTE_SESSION

NAME

ft_close - destroy an XFTAM instance and release associated resources

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_close(
      OM_private_object   session
);
```

DESCRIPTION

This function deletes an XFTAM *instance* established by *ft_open*( ) and releases associated resources. No XFTAM function may reference the specified session and its associated workspace once it has been closed.

Any asynchronous operations that are outstanding when a session is closed are abandoned in the manner defined by the *ft_abandon*( ) function. The warnings stated there with regard to the state of the files and filestores involved in a file transfer apply in this case too. Underlying XFTAM associations are destroyed in the manner described by either the *ft_abort*( ) or *ft_disconnect*( ) functions.

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

RETURN VALUES

ft_close( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS.

ERRORS

API Error Codes
FTE_VENDOR
FTE_SESSION

NAME

ft_connect - establish an association with an FTAM filestore

SYNOPSIS

```
#include <xftam.h>

ft_return_code ft_connect(
        OM_private_object       session ,
        OM_object               p_address ,
        OM_object               ftam_in,
        OM_object               api_in,

        OM_uint32               *association_id ,
        OM_private_object       *ftam_out,
        OM_private_object       *api_out
);
```

DESCRIPTION

The *ft_connect*( ) function creates an association with an FTAM filestore identified by p_address and establishes an FTAM regime.

The function returns association_id as an integer which is used to identify the new association to other XFTAM functions which will operate within the context of the association created.

ft_connect( ) must be invoked first for context-sensitive operations to operate on any association. It may be invoked multiple times within a single XFTAM instance and may be to the same p_address as each call creates a separate association.

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of the class session which identifies the particular XFTAM instance within which the association is to be created.

p_address (Object (Presentation-Address))
This parameter is a handle for an object of the class Presentation Address. It is a mandatory parameter that identifies the FTAM responder which serves the remote filestore.

ftam_in (Object(FTAM-Input-Parameters))
This parameter is a handle for an object class *FTAM-Input-Parameters*, specifying general FTAM parameters for use in this function. The parameter is optional. However, failure to specify some of its OM attributes may result in the remote responder rejecting the association request. The *ft_connect*( ) function has the following specific requirements for the input object:

- **Initiator-Identity**.
  This is provided to identify the FTAM initator to the responder.
- **Filestore-Password**.
  This password is provided to authenticate the initiator to the FTAM responder.
- **Account**.
  The account given is charged for all costs incurred by the FTAM regime. This may be overridden by providing an account parameter for subsequent FTAM operations on the association.

- **FQoS**.
  The FTAM quality of service parameter is used to indicate the level of error recovery available at the FTAM initiator.

association_id (OM_uint32)
Upon sucessful completion, this parameter is used to return an integer which identifies the XFTAM association created. This value may be passed to other XFTAM functions in order to identify the association to be used for a context-sensitive request.

ftam_out(Object(FTAM-Output-Parameters))
This parameter is a handle for a private object of class *FTAM-Output-Parameters*, and is returned only if there are relavent FTAM output parameters to be returned as a result of the FTAM actions performed.

api_out (Private Object(API-Output-Parameters)
This parameter is always returned and is a handle for a private object of class API-Output-Parameters. It returns API-Specific output parameters for this function call.

RETURN VALUES

ft_connect( ) returns either [FTE_SUCCESS] or one of the values listed in ERRORS.

ERRORS

FTAM Error Codes
FTE_FTAM_CANCEL
FTE_FTAM_PERMANENT
FTE_PROVIDER_ABORT
FTE_USER_ABORT
FTE_FQOS_NOT_NEGOTIATED

Operation Error Code
FTE_ATTR_GRP_NOT_NEGOTIATED

API Error Codes
FTE_NO_RESOURCES
FTE_VENDOR
FTE_INV_PADDRESS
FTE_SESSION
FTE_TOO_MANY_ASSOC

NAME

ft_disconnect - disconnect an XFTAM association

SYNOPSIS

```
#include <xftam.h>

ft_return_code ft_disconnect(
        OM_private_object       session ,
        OM_uint32               association_id ,
        OM_private_object       *ftam_out,
        OM_private_object       *api_out
);
```

DESCRIPTION

This function terminates an FTAM regime and destroys the underlying association.

If the identified association has active functions in progress, *ft_disconnect*( ) returns an error code of [FTE_PENDING_OP] and takes no action.

If the Association-Id provided does not represent an active association within the specified FTAM instance, *ft_disconnect*( ) returns an error code of [FTE_INV_ASSOC].

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of the class Session which identifies the particular XFTAM instance within which the association exists.

association_Id (integer)
This parameter is an integer which identifies the FTAM association to be closed by *ft_disconnect*( ) and must be the value returned by *ft_connect*( ) when the association was created.

ftam_out (Object(FTAM-Output-Parameters))
This parameter is a handle for a private object of class *FTAM-Output-Parameters*, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed. *ft_disconnect*( ) has the following specific requirement for the output object:

- **Charging-List**
  When this attribute is present, charges associated with the FTAM regime and any operations completed using it are returned. The charges returned do not include charges for file transfer or file management operations for which a different account was provided in the *FTAM-Input-Parameters*, these charges being returned upon completion of that function.

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for the private object of class *API-Output-Parameters*. It returns API-specific output parameters for this function call.

RETURN VALUES

ft_disconnect( ) returns either [FTE_SUCCESS] or one of the values listed in ERRORS.

ERRORS

Operation Error Codes
- FT_INV_ASSOC
- FT_PENDING_OP

API Error Codes
- FTE_VENDOR
- FTE_SESSION

NAME

ft_fcattributes - change the file attributes of an FTAM file

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_fcattributes(
    OM_private_object    session ,
    OM_object       p_address ,
    OM_string       *filename ,
    OM_object       new_attributes ,
    OM_object       ftam_in ,
    OM_object       api_in ,

    OM_private_object    *return_attributes ,
    OM_private_object    *ftam_out ,
    OM_private_object    *api_out
) ;
```

DESCRIPTION

The *ft_fcattributes*( ) function modifies the FTAM file attributes of *filename* in the filestore identified by *p_address*.

The function may only be used to modify those attributes listed in the description of the *new_attributes* parameter below. In addition, the FTAM attributes *date_time_of_attribute_mod* and *identity_of_attribute_mod* are changed by the responder as a result of invoking *ft_fcattributes*( ).

If *ft_fcattributes*( ) is not successful, all FTAM file attributes of the target file (except attributes implicitly changed by the responder) have the same value as before invoking *ft_fcattributes*( ). If *ft_fcattributes*( ) is successful, the values of all FTAM attributes of the target file are returned in the *return_attributes* object.

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

p_address (Object (Presentation-Address))
This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

filename (String(Graphic))
The name of the file for which attributes are to be modified. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

new_attributes (Object (New_Attributes))
This parameter is a handle for an object of class *New-Attributes* which contains the new

values for the FTAM file attributes to be changed. It is a *mandatory* parameter which must contain at least one FTAM attribute value. Only the following attributes may be changed by a call to this function:

FILENAME
STORAGE_ACCOUNT
FILE_AVAILABILITY
FUTURE_FILESIZE
ACCESS_CONTROL
LEGAL_QUAL
PRIVATE_USE

If the underlying FTAM service provider fails to negotiate the use of an FTAM attribute group required for one of the specified attibutes, the function returns an error. A request to change attributes other that those listed also results in an error.

In the case of access-control, additional conditions, to be *added* to the target file's current list are specified in the OM attribute *Access-Control-List*, conditions to be *deleted* are specified using *Delete-Access-Control-List*. FTAM states that a condition in the deletion list must exactly match one in the target file's current access control attribute if the deletion is to succeed.

ftam_in (Object (FTAM-Input-Parameters))
This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions. The *ft_fcattributes*( ) function has the following specific requirements for these parameters:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present) this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case charges for this operation are returned upon completion.

  When not present, the account identified when *ft_connect*( ) created the association is used for any charges and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **File-Passwords**.
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the *read* and *change_attribute* passwords if *filename* contains an access control element which specifies passwords for these actions.

- **Concurrency-Control**.
  This parameter is used to specify concurrency locks for the FTAM file actions to be performed. Set the *read* and *change_attribute* concurrency keys if *filename* contains an access control element which specifies locks required for these actions.

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))
This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

return_attributes (Private Object (FTAM-Attributes))
If successful, this parameter is a handle for a private object of class *FTAM-Attributes* which contains the new values of all the target file's attributes (not just those for which new values were supplied). Return of attribute information as a result of the change attribute file action is optional; some responders may not do so.

Values are returned only for FTAM attributes from groups which the underlying FTAM service provider was able to negotiate for the association.

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUE

For synchronous calls:
ft_fcattributes( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS. The function return code and the *Return-Code* XOM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:
ft_fcattributes( ) returns either [FTE_SUCCESS] or one of the values in the *API Error Codes* list of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out* and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than [FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes

- FTE_FTAM_CANCEL
- FTE_FTAM_PERMANENT
- FTE_PROVIDER_ABORT
- FTE_USER_ABORT

Operation Error Codes

- FTE_INV_ATTRIBUTES
- FTE_ATTR_GRP_NOT_NEGOTIATED
- FTE_SERV_CLS_NOT_NEGOTIATED
- FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes

- FTE_CANCEL
- FTE_NO_RESOURCES
- FTE_VENDOR
- FTE_NOTSUP_ASYNC
- FTE_INV_PADDRESS
- FTE_SESSION
- FTE_TOO_MANY_OPS
- FTE_INV_ASSOC
- FTE_CONTEXT_MISMATCH

WARNINGS

The *Access-Passwords* OM attribute of the *Attributes* class is never returned.

NAME

ft_fdelete - delete an FTAM file

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_fdelete(
    OM_private_object    session ,
    OM_object            p-address ,
    OM_string            *filename ,
    OM_object            ftam_in ,
    OM_object            api_in ,

    OM_private_object      *ftam_out ,
    OM_private_object      *api_out
) ;
```

DESCRIPTION

The *ft_fdelete*( ) function removes *filename* from the filestore identified by *p-address*.

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

p_address (Object (Presentation-Address))
This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

filename (String(Graphic))
The name of the file to be deleted. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

ftam_in (Object (FTAM-Input-Parameters))
This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions. The *ft_fdelete*( ) function has the following specific requirements for these parameters:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present), this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case, charges for this operation are returned upon completion.

When not present, the account identified when *ft_connect*( ) created the association is used for any charges, and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **File-Passwords**.
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the *delete* password if *filename* contains an access control element which specifies passwords for these actions.

- **Concurrency-Control**.
  This parameter is used to specify concurrency locks for the FTAM file actions to be performed. Set the *delete* concurrency key if *filename* contains an access control element which specifies locks for these actions.

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))
This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

For synchronous calls:

ft_fdelete( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS. The function return code and the *Return-Code* OM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:

ft_fdelete( ) returns either [FTE_SUCCESS] or one of the values in the *API Error Codes* list of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out* and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than [FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes

- FTE_FTAM_CANCEL
- FTE_FTAM_PERMANENT
- FTE_PROVIDER_ABORT
- FTE_USER_ABORT

Operation Error Codes

- FTE_ATTR_GRP_NOT_NEGOTIATED
- FTE_SERV_CLS_NOT_NEGOTIATED
- FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes

- FTE_CANCEL
- FTE_NO_RESOURCES
- FTE_VENDOR
- FTE_NOTSUP_ASYNC
- FTE_INV_PADDRESS
- FTE_SESSION
- FTE_TOO_MANY_OPS
- FTE_INV_ASSOC
- FTE_CONTEXT_MISMATCH

NAME

ft_frattributes - read the FTAM attributes of a file

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_frattributes(
    OM_private_object    session ,
    OM_object       p-address ,
    OM_string       *filename ,
    OM_object       attribute_names ,
    OM_object       ftam_in ,
    OM_object       api_in ,

    OM_private_object    *return-attributes ,
    OM_private_object    *ftam_out ,
    OM_private_object    *api_out
) ;
```

DESCRIPTION

The *ft_frattributes* function reads the values of the specified FTAM file attributes of *filename* in the filestore identified by *p-address*. The set of attributes returned depend on the FTAM file attribute groups negotiated for the connection and on the level of support for the attributes in groups which the responder supports partially. Some responders may return an integer or string indicating "No value available" for partially supported attributes. The values of the **Access-Passwords** OM attribute of the **Attributes** class are never returned.

The filestore changes the FTAM file attributes *date_time_of_read* and *identity_of_reader* as a result of the read attribute action.

ARGUMENTS

session (Private Object (Session))

This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

p_address (Object (Presentation-Address))

This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

filename (String(Graphic))

The name of the file for which attributes are to be read. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

attribute_names (Object (FTAM-Attribute-names))

This **optional** parameter is a handle for an object of class *FTAM-Attribute-Names* which indicates which file attributes from the kernel or negotiated FTAM attribute groups to read.

If not specified, *attribute_names* defaults to all the attributes associated with the attribute groups negotiated for the association.

The attributes which may be read are:

FILENAME,
PERMITTED_ACTIONS,
CONTENTS_TYPE,
CREATE_DATE_TIME,
MOD_DATE_TIME,
READ_DATE_TIME,
ATT_MOD_DATE_TIME,
ID_OF_CREATOR,
ID_OF_MODIFIER,
ID_OF_READER,
ID_OF_ATT_MOD,
STORAGE_ACCOUNT,
FILE_AVAILABILITY,
FILESIZE,
FUTURE_FILESIZE,
ACCESS_CONTROL,
LEGAL_QUAL,
PRIVATE_USE.

If the underlying FTAM service provider fails to negotiate the use of an FTAM attribute group required for one of the specified attibutes, the function returns an error. A request to read attributes other than those listed here also results in an error.

ftam_in (Object (FTAM-Input-Parameters))
This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions. The *ft_frattributes*( ) function has the following specific requirements for these parameters:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present), this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case, charges for this operation are returned upon completion.

  When not present, the account identified when *ft_connect*( ) created the association is used for any charges, and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **File-Passwords**.
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the *read* and *read_attribute* passwords if *filename* contains an access control element which specifies passwords for these actions.

- **Concurrency-Control**.
  This parameter is used to specify concurrency locks for the FTAM file actions to be

performed. Set the *read* and *read_attribute* concurrency keys if *filename* contains an access control element which specifies locks for these actions.

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))
This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

return_attributes (Private Object (FTAM-Attributes))
If sucessful, this parameter is a handle for a private object of class *FTAM-Attributes* which contains the values of the attributes requested.

If the accessed file is of a type supported by XFTAM, the object handle returned in the *Content-Type* XOM attribute will point to an object describing the specific document type of the file (for example *Document-Type-FTAM-1*), including any associated parameters. Alternatively, if the document type is not supported, the returned object will be a simple *Document-Type* object (that is, no information is returned about the values of any parameters that may be associated with the document type).

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

For synchronous calls:
ft_frattributes( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS. The function return code and the *Return-Code* OM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:
ft_frattributes( ) returns either [FTE_SUCCESS] or one of the values in the *API Error Codes* list

of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out* and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than [FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes
- FTE_FTAM_CANCEL
- FTE_FTAM_PERMANENT
- FTE_PROVIDER_ABORT
- FTE_USER_ABORT

Operation Error Codes
- FTE_ATTR_GRP_NOT_NEGOTIATED
- FTE_SERV_CLS_NOT_NEGOTIATED
- FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes
- FTE_CANCEL
- FTE_NO_RESOURCES
- FTE_VENDOR
- FTE_NOTSUP_ASYNC
- FTE_INV_PADDRESS
- FTE_SESSION
- FTE_TOO_MANY_OPS
- FTE_INV_ASSOC
- FTE_CONTEXT_MISMATCH

WARNINGS

The *Access-Passwords* XOM attribute of the *FTAM-Attributes* XOM class is never returned.

SEE ALSO

ft_gperror( ), *ft_rcvresult*( ).

NAME

ft_frdir - read the contents of an FTAM file-store directory

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_frdir(
    OM_private_object    session ,
    OM_object       p_address ,
    OM_string       *pathname ,
    OM_object       attribute_names ,
    OM_object       ftam_in ,
    OM_object       api_in ,

    OM_private_object    *directory_list ,
    OM_private_object    *ftam_out ,
    OM_private_object    *api_out
) ;
```

DESCRIPTION

The *ft_frdir* function reads the contents of the directory *pathname* in the file-store identified by *p_address*. A list of zero or more *FTAM-Attributes* objects is returned, one for each file in the remote directory. The order of files in this list is determined by the source file-store.

The *ft_frdir* function uses the NBS-9 document type to retrieve directory information from the remote file-store. The function may fail if the remote file-store does not support this document type.

Two factors may constrain the list of attributes returned for each file:

- The FTAM file attribute groups negotiated for the connection with the FTAM responder may limit the list of attributes that are *requested* for each file.
- The attributes that the remote FTAM responder supports for the NBS-9 file type may limit the list of attributes *returned* for each file. The definition of the NBS-9 document type requires only that the Filename attribute is returned for each file.

ARGUMENTS

session (Private Object (Session))

This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

p_address (Object (Presentation-Address))

This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

pathname (String(Octet))
The name of the directory file for which a file list is to be retrieved.

attribute_names (Object(FTAM_Attribute_names))
This optional parameter is a handle for an object of class FTAM-Attribute-Names which indiacates which file attributes from the kernel or negotiated FTAM attribute group to read. If not specified, attribute_names defaults to the attribute filename only.

The attributes which may be read are:
FILENAME
PERMITTED_ACTIONS
CONTENTS_TYPE
CREATE_DATE_TIME
MOD_DATE_TIME
READ_DATE_TIME
ATT_MOD_DATE_TIME
ID_OF_CREATOR
ID_OF_MODIFIER
ID_OF_READER
ID_OF_ATT_MOD
STORAGE_ACCOUNT
FILE_AVAILABILITY
FILESIZE
FUTURE_FILESIZE
ACCESS_CONTROL
LEGAL_QUAL
PRIVATE_USE

If the underlaying FTAM service provider fails to negotiate the use of an FTAM attribute group required for one of the specified attributes, the function returns an error. A request to read attributes other than those listed here also results in an error. " A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

ftam_in (Object (FTAM-Input-Parameters))
This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions. For the *ft_frdir*( ) the following attributes of this object may be set:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present), this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case, charges for this operation are returned upon completion.

  When not present, the account identified when *ft_connect*( ) created the association is used for any charges, and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **File-Passwords**.
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the *read* password if *filename* contains an access control element which

specifies a password for this action.

- **Concurrency-Control**.
  This parameter is used to specify concurrency locks for the FTAM file actions to be performed. Set the *read* concurrency key if *filename* contains an access control element which specifies a lock for this action.

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))
This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

directory_list (Private Object (Directory-List))
If sucessful, this parameter is a handle for a private object of class *Directory-List* which contains a list of zero or more *FTAM-Attribute* objects, one for each file in the directory accessed. Only values of attributes from groups supported by responder for the *NBS-9* document type are returned.

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

For synchronous calls:
ft_frdir( ) returns either SUCCESS or one of the values listed below in ERRORS. The function return code and the *Return-Code* OM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:
ft_frdir( ) returns either SUCCESS or one of the values in the *API Error Codes* list of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out*

and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than [FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes
- FTE_FTAM_CANCEL
- FTE_FTAM_PERMANENT
- FTE_PROVIDER_ABORT
- FTE_USER_ABORT

Operation Error Codes
- FTE_ATTR_GRP_NOT_NEGOTIATED
- FTE_SERV_CLS_NOT_NEGOTIATED
- FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes
- FTE_CANCEL
- FTE_NO_RESOURCES
- FTE_VENDOR
- FTE_NOTSUP_ASYNC
- FTE_INV_PADDRESS
- FTE_SESSION
- FTE_TOO_MANY_OPS
- FTE_INV_ASSOC
- FTE_CONTEXT_MISMATCH

WARNINGS

The *Access-Passwords* attribute of the *Attributes* class is never returned.

NAME

ft_freceive - receive a file from an FTAM filestore

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_receive(
    OM_private_object    session ,
    OM_object       p_address ,
    OM_string       *src_filename ,
    OM_enum         src_effect ,
    OM_string       *dest_filename ,
    OM_enum         dest_effect ,
    OM_object       src_attributes ,
    OM_object       ftam_in ,
    OM_object       api_in ,

    OM_private_object    *return_attributes ,
    OM_private_object    *ftam_out ,
    OM_private_object    *api_out
) ;
```

DESCRIPTION

The *ft_freceive*( ) function copies the *src_filename* in the FTAM filestore identified by *p-address* to the local file *dest_filename*. If *dest_filename* is NULL, the function attempts to use the source filename as the name of the destination file, this may result in a modified filename being created or the transfer may fail.

Src_effect allows the source file to be deleted if the copy completes successfully. If the API user does not establish sufficient authorisation in the remote FTAM VFS to delete the source file, the file transfer succeeds but the function returns an error indicating that the source file was not deleted.

Dest_effect specifies the action to be taken if the destination file already exists in the remote file system. The transfer fails if *fail* is specified, or if *overwrite* or *extend* is specified and the API user does not have permission to perform the required action in the local filestore.

The setting of the file access permissions of the resulting destination file are implementation dependant.

ARGUMENTS

session (Private Object (Session))

This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

p_address (Object (Presentation-Address))

This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

src_filename (String(Graphic))
The name of the source file. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

src_effect (Enum(Copy-Move))
This parameter is an enumeration which specifies the effect of the file transfer upon the source file. Possible values are:

- *FT_COPY_FILE*, meaning that the source file is to be left in place when the file transfer is complete (i.e. the transfer is a file copy). The option uses the *read* FTAM file action.
- *FT_MOVE_FILE*, meaning that the source file is to be deleted once the file transfer is complete (i.e. the transfer is a file move). If an error occurs which means that the file tranfer does not complete, source file is left in place. The option uses the *read* and *delete* FTAM file actions.

The transfer fails if the required file actions are not allowed by the file's *permitted actions* attribute, or if concurrency locks or file passwords are required for the actions and are not correctly specified in the *File-Passwords* and *Concurrency-Control* OM attributes of the *FTAM-Input-Parameters* object.

dest_filename (String(Graphic))
The name of the local destination file. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

A NULL value indicates that *src_filename* should be used as the filename in the destination filestore. The source filename may be modified by XFTAM to conform to local file naming conventions, or the transfer may fail if the source filename cannot be used. The actual name of the file created is returned in the function's *return_attributes* output parameter.

dest_effect (Enum (Receive-Effect))
This parameter is an enumeration which defines the action to be taken if the destination file *dest_filename* exists. It takes one of the following values:

- *FT_OVERWRITE* - delete the existing file and replace with the file received. The transfer fails if the user does not have permission to delete the file.
- *FT_APPEND* - the remote file is appended to the existing local file. The transfer fails if the user does not have write permission for the file.
- *FT_FAIL* - the file transfer fails.

src_attributes (Object (FTAM-Attributes))
This parameter is a handle to an object of class *FTAM-Attributes*. Only the **Content-Type** OM attribute is significant for this call. Other OM attributes are ignored. If this parameter is NULL, or if no Document-Type OM attribute is included, XFTAM does not specify what document type it is expecting to receive. In this case the transfer may fail if the actual document type or parameter combination specified by the source file's file attributes is not supported by the local XFTAM implementation.

The API user may set the **Content-Type** OM attribute to restrict the type of file read from the responder. This may case the request to fail if the specified *content type* does not match that of the source file or a valid simplification/relaxation of it according to the rules specified by FTAM. It may also fail if the local implementation does not support the specified document type or combination of parameters.

ftam_in (Object (FTAM-Input-Parameters))

This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions.

The *ft_freceive*( ) function has the following specific requirements these parameters:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present), this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case, charges for this operation are returned upon completion.

  When not present, the account identified when *ft_connect*( ) created the association is used for any charges, and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **File-Passwords**.
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the password for the required action if *src_filename* contains an access control element which specifies a password for this action. (See the discussion of the *src_effect* parameter above for a description of the FTAM file actions that this function may perform.)

- **Concurrency-Control**.
  This parameter is used to specify concurrency locks for the FTAM file actions to be performed. Set the concurrency key for the required action if *src_filename* contains an access control element which specifies a lock for this action. (See the discussion of the *src_effect* parameter above for a description of the FTAM file actions that this function may perform.)

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))

This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

return-Attributes (Private Object (FTAM-Attributes))

If sucessful, this parameter is a handle for a private object of class *FTAM-Attributes* which returns the attributes of the file as received. Attribute values are returned for the actual filename (which may differ from that specified in the request) and the actual content type of the file accessed.

The *content type* attribute is either that of the source file, or a valid simplification if one was requested using the *source-attribute* input parameter. The contents of the file received may be processed by the local FTAM initiator according to the *content type* received:

- **FTAM-1**. This is an unstructured text file. The contents of the file are filtered to convert FTAM format effectors (in particular end of line) into the equivalent local representation. An XFTAM implementation must support a string length of at least 134 characters for FTAM-1 files received (132 characters plus <CR> <LF>). Implementations may support larger or unlimited string lengths.

- **FTAM-2**. This is a sequential text file. Support for this document type is optional. XFTAM supports the transfer of entire files only. No mechanism is provided for transferring individual records from such a file.

  The strings of the file are processed as for the FTAM-1 document type. The mechanism by which the record boundaries of the FTAM-2 document are preserved in the local filestore is outside of the scope of XFTAM.

- **FTAM-3**. This is an unstructured binary file. The contents are not interpreted or changed in any way when it is received. An XFTAM implementation must support a string length of at least 512 octets for FTAM-3 files received. Implementations may support larger or unlimited string lengths.

The [FTE_INV_STRING_LENGTH] error is returned if the user attempts to receive a document which exceeds the maximum supported value for the *maximum-string-length* document type parameter.

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

For synchronous calls:
ft_freceive( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS. The function return code and the *Return-Code* OM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:
ft_receive( ) returns either [FTE_SUCCESS] or one of the values in the *API Error Codes* list of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out* and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than

[FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes
FTE_FTAM_CANCEL
FTE_FTAM_PERMANENT
FTE_PROVIDER_ABORT
FTE_USER_ABORT

Operation Error Codes
FTE_FILE_EXISTS
FTE_INV_DOC_RCVD
FTE_INV_DOC_SPEC
FTE_INV_STRING_LENGTH
FTE_LOCAL_FILE_ERROR
FTE_LOCAL_PERMISSION
FTE_ATTR_GRP_NOT_NEGOTIATED
FTE_SERV_CLS_NOT_NEGOTIATED
FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes
FTE_CANCEL
FTE_NO_RESOURCES
FTE_VENDOR
FTE_NOTSUP_ASYNC
FTE_NOTSUP_FQOS
FTE_NOTSUP_FTAM2
FTE_INV_PADDRESS
FTE_SESSION
FTE_TOO_MANY_OPS
FTE_INV_ASSOC
FTE_CONTEXT_MISMATCH

NAME

ft_fsend - send a file to an FTAM filestore

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_fsend(
    OM_private_object    session ,
    OM_string       *src_filename ,
    OM_enum         src_effect ,
    OM_object       p_address ,
    OM_string       *dest_filename ,
    OM_enum         dest_effect ,
    OM_object       initial_attributes ,
    OM_object       ftam_in ,
    OM_object       api_in

    OM_private_object    *return_attributes ,
    OM_private_object    *ftam_out ,
    OM_private_object    *api_out
) ;
```

DESCRIPTION

The *ft_fsend*( ) function copies the local *src_filename* to *dest_filename* on the FTAM filestore identified by p_address. If *dest_filename* is NULL, the function attempts to use the source filename as the name of the destination file. The transfer fails if the source filename is not compatible with the destination filestore.

Whether it is derived from *src_filename* or *dest_filename*, the actual filename created in the remote filestore may be modified by the remote responder.

Dest_effect specifies the action to be taken if the destination file already exists in the remote filestore. The transfer fails if the destination file exists and one of the *dest_effect* options is chosen which is not permitted for the file or for which the API user does not supply the appropriate file passwords or concurrency locks.

The *initial_attributes* input parameter allows the user to override the values of some of the FTAM file attributes in the file created. If this parameter is not provided, some of the file's FTAM attributes are determined by XFTAM and others by the receiving FTAM responder. The *return_attributes* parameter returns the actual file attributes of the file created by the destination responder (including the actual filename created.

ARGUMENTS

session (Private Object (Session))
This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

src_filename (String(Graphic))
The name of the local source file. A *mandatory* parameter, given in the syntax used by the real filestore containing the file.

src_effect (Enum(Copy-Move))
This parameter is an enumeration which specifies the effect of the file transfer upon the

source file. The value is one of:

- *FT_COPY_FILE*, meaning that the source file is to be left in place when the file transfer is complete (i.e. the transfer is a file copy).
- *FT_MOVE_FILE*, meaning that the source file is to be deleted once the file transfer is complete (i.e. the transfer is a file move). If an error occurs which means that the file tranfer does not complete, source file is left in place. If the user does not have permission to delete the source file, the file transfer succeeds but the function returns an error indicating that the source file has not been deleted.

p_address (Object (Presentation-Address))
This parameter is a handle for an object of class **Presentation Address**. If present, the Association-Id attribute of API-Input-Parameters shall be absent as the operation is being carried out in context free mode. When present, this attribute identifies the FTAM responder which serves the remote filestore.

If not present, the *Association-Id* attribute of API-Input-Parameters shall be present as the operation is being carried out in context sensitive mode.

If both *P-address* and *Association-Id* are present, the function returns an error code [FT_CONTEXT_MISMATCH].

dest_filename (String(Graphic))
The name of the destination file. Given in the syntax used by the real destination file system. A NULL value indicates that *src_filename* should be used as the filename in the destination filestore.

Whether it is derived from *src_filename* or *dest_filename*, the created filename may be modified by the responder to conform to the file naming conventions of the remote filestore. The transfer fails if the filename is not compatible with these conventions. The actual name of the file created is returned in the function's *return_attributes* output parameter.

dest_effect (Enum (FTAM-Override))
This parameter maps onto the FTAM *override* parameter to the F-CREATE request primitive and specifies the action to be taken by the responder in the case where *dest_filename* exists. The parameter can take the following parameters:

- FT_CREATE_FAILURE - the file transfer fails.
- FT_SELECT_OLD_FILE - the local file is appended to the existing file on the destination file store. For FTAM-1 and FTAM-3 document types, this option uses the FTAM extend file action; for FTAM-2, it uses insert.
- FT_DELETE_AND_CREATE_WITH_OLD_ATTRIBUTES - the local file overwrites the destination file, preserving the attributes of the destination. This option uses the FTAM delete file action. In addition, for FTAM-1 and FTAM-3 document types, this option uses the FTAM replace file action; for FTAM-2, it uses insert.
- FT_DELETE_AND_CREATE_WITH_NEW_ATTRIBUTES - the local file overwrites the destination file, the new file's attributes are as specified by the FTAM Initial Attributes parameter. This option uses the FTAM delete file action. In addition, for FTAM-1 and FTAM-3 document types, this option uses the FTAM replace file action; for FTAM-2, it uses insert.

Note: As the FTAM file attributes of the new file are established as part of the create operation, no password need be specified in the File-Passwords attribute of FTAM-Input-Parameters in order to perform the replace/insert action in this case.

The transfer fails if one of the required file actions is not allowed by the file's *permitted actions* attribute, or if concurrency locks or file password is required for the action and are not correctly specified in the *File-Passwords* and *Concurrency-Control* OM attributes of the *FTAM-Input-Parameters* object.

initial-attributes (Object (FTAM-Attributes))

This *optional* parameter is a handle for an object of class *FTAM-Attributes* specifying the file attributes which are to be set for the destination file in the *initial attributes* FTAM parameter to the *F-CREATE* request primitive. A value for an attribute specified here overrides any default value that the local XFTAM implementation may provide. In turn, the receiving responder may modify certain attributes when creating the file, or set attributes other than those in the kernel group to ''no value available''. The user may specify the following FTAM file attributes in a call to the *ft_send*( ) function:

PERMITTED_ACTIONS
CONTENTS_TYPE
STORAGE_ACCOUNT,
FILE_AVAILABILITY
FUTURE_FILESIZE
ACCESS_CONTROL
LEGAL_QUAL
PRIVATE_USE

If the underlying FTAM service provider fails to negotiate the use of an FTAM attribute group required for one of the specified attibutes, the function returns an error. A request to specify an intial value for an attribute other than those listed here also results in an error.

The following specific points should be noted:

- **Filename**
  This parameter is set by XFTAM from the *src_filename* and *dest_filename* function parameters as noted above. The FTAM responder may modify the value of this attribute for the created file, the modified value is returned in the function's *return_attributes* output parameter.

- **Permitted actions**
  The FTAM responder may modify the value of this attribute for the created file, the modified value is returned in the *return_attributes* function output parameter.

- **Contents type**
  If the user does not specify a value for this FTAM attribute, XFTAM defaults it to the defined FTAM-1 set or optionally determines the file contents type by an implementation defined lookup service. If the implementation defined service is unable to determine the file contents type, it defaults it to the FTAM-1 set. The implementation of the lookup service is beyond the scope of this specification.

  Note: When the user application is able to determine the contents type, it is recommended to specify a value for this FTAM attribute. Relying on the setting of the default value may be inappropriate. For example, transferring a binary file (FTAM-3) as a text file (FTAM-1) may fail due to ''new line'' manipulation.

  The FTAM-1 default contents type:

  — Document type = **FTAM-1** (unstructured text)

  — Universal class = **General String** (text may contain format effectors)

— Maximum string length = *unlimited*.

— String significance = *non-significant* (lines from the source file are delimited by the FTAM format effectors <CR> <LF>).

The source file is assumed to be an unstructured file, containing a stream of bytes. If the file contains text such as screen or print images, the document type may be specified as FTAM-1 with additional parameters to specify the appropriate universal class number (for example IA5-String).

If the file to be transferred contains binary information, the API user may specify an alternative document type (FTAM-3), to avoid unwanted interpretation of the source file's contents.

If an implementation supports the optional FTAM-2 document type, the API user may specify that the file is to be transferred as a sequential text file. XFTAM supports the transfer of entire files only, no mechanism is provided for transferring individual records from such a file. The default parameters for such a file are as for the FTAM-1 document type. The mechanism by which the record boundaries of the FTAM-2 document are identified in the local filestore is outside of the scope of XFTAM.

The transfer may fail if a document type or parameter combination not supported by the local implementation is specified.

The *minimum* value for the *maximum-string-length* document type parameter that an XFTAM implementation must support is 134 characters for the FTAM-1 and FTAM-2 document types, and 512 octets for FTAM-3. The [FTE_INV_STRING_LENGTH] error is returned if the user attempts to send a document which exceeds the maximum supported value for this parameter.

ftam_in (Object (FTAM-Input-Parameters))
This parameter is a handle for an object of class **FTAM-Input-Parameters**, specifying general FTAM parameters for use in this function. The parameter is *optional*. However, failure to specify some of its OM attributes may result in the remote responder rejecting the requested file actions. The *ft_fsend*( ) function has the following specific requirements for the input object:

- **Account**.
  If context-sensitive processing mode is in use for this operation (*Association-Id* is present), this parameter is optional.

  When present, for the duration of this file transfer or file management function, it overrides the current identified account to which charges are made (as defined when *ft_connect*( ) created the association). In this case, charges for this operation are returned upon completion.

  When not present, the account identified when *ft_connect*( ) created the association is used for any charges, and no charging information is returned when this function completes, all charging information being returned when the association is destroyed by *ft_disconnect*( ).

- **Create-Password**
  Some responders may require this FTAM attribute to be specified when a file is to be created in the destination filestore.

- **File-Passwords**
  This parameter is used to specify file passwords for the FTAM file actions to be performed. Set the password for the required action if *dest_filename* already exists and

contains an access control element which specifies a password for this action. (See the discussion of the *dest_effect* parameter above for a description of the FTAM file actions that this function may perform).

- **Concurrency-Control**
  This parameter is used to specify concurrency locks for the FTAM file actions to be performed. Set the concurrency key for the required action if *dest_filename* already exists and contains an access control element which specifies a lock for this action. (See the discussion of the *dest_effect* parameter above for a description of the FTAM file actions that this function may perform).

If context-sensitive processing mode is in use for this function call (*Association-Id* is present), the following parameters should not be present as they have already been provided when the association was created. In this case, if any of these are present, the function returns an error code [FT_CONTEXT_MISMATCH].

— Initiator-Identity

— Filestore-Password

— FQoS.

api_in (Object (API-Input-Parameters))
This *optional* parameter is the handle of an object of class **API-Input-Parameters**, which may contain API-specific parameters for use in this function call.

If context-sensitive processing mode is in use, this parameter contains the *Association-Id* for an existing association. If the *Association-Id* provided does not represent an active association within the FTAM instance identified by Session, the function returns an error code [FTE_INV_ASSOC].

return_attributes (Private Object (FTAM-Attributes))
If sucessful, attribute values are returned for the actual filename of the file created, and its permitted actions, either of which may have been modified by the remote responder when creating the file in the virtual filestore.

ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is a handle for a private object of class **FTAM-Output-Parameters**, and is returned only if there are relevant FTAM output parameters to be returned as a result of the FTAM actions performed.

If context-sensitive processing mode is in use, the following specific parameter use applies:

- **Charging-List**.
  If an override account was provided (in the account attribute within *ftam_in*), any charges returned are those for this function only and do not include connection changes. The charges returned here are not included in the charges returned when the association is destroyed with *ft_disconnect*( ).

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUE

For synchronous calls:
ft_fsend( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS. The function return code and the *Return-Code* OM attribute of the *API-Output-parameters* output object are identical for synchronous calls.

For asynchronous calls:

ft_fsend( ) returns either [FTE_SUCCESS] or one of the values in the *API Error Codes* list of the ERRORS section below. If the call returns [FTE_SUCCESS] the contents of *ftam_out*, *api_out* and any other output parameters that this function returns are undefined (these parameters are returned by a subsequent call to *ft_rcv_result*( )). For return codes other than [FTE_SUCCESS] the function return code and the *Return-Code* XOM attribute of the *API-Output-Parameters* output object are identical.

ERRORS

FTAM Error Codes

- FTE_FTAM_CANCEL
- FTE_FTAM_PERMANENT
- FTE_PROVIDER_ABORT
- FTE_USER_ABORT

Operation Error Codes

- FTE_INV_ATTRIBUTE
- FTE_INV_DOC_SPEC
- FTE_INV_STRING_LENGTH
- FTE_LOCAL_FILE_ERROR
- FTE_LOCAL_PERMISSION
- FTE_NO_SRC_FILE
- FTE_ATTR_GRP_NOT_NEGOTIATED
- FTE_SERV_CLS_NOT_NEGOTIATED
- FTE_FUNCT_UNIT_NOT_NEGOTIATED

API Error Codes

- FTE_CANCEL
- FTE_NO_RESOURCES
- FTE_VENDOR
- FTE_NOTSUP_ASYNC
- FTE_NOTSUP_FQOS
- FTE_NOTSUP_FTAM2
- FTE_INV_PADDRESS
- FTE_SESSION
- FTE_TOO_MANY_OPS
- FTE_INV_ASSOC
- FTE_CONTEXT_MISMATCH

NAME

ft_gperror - translate an FTAM error to a printable string

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_gperror(
    OM_private_object       api_out_in ,
    OM_string               **return_string ,
    OM_string               **vendor_string ,
    OM_private_object       *api_out
) ;
```

DESCRIPTION

The *ft_gperror* function translates the result of a previous call to an XFTAM API function into a pair of printable strings, corresponding to the Return-Code and Vendor-Code attributes of the *API-Output-Parameters* object class.

ARGUMENTS

api_out_in (Private Object (API-Output-Parameters))

This *mandatory* parameter is a handle for an object of class *API_Output-Parameters* which contains the API output parameters for a previous another function call for which the printable errors strings are required.

return_string (OM_String(*))

A text string in the natural language of the current locale that represents the *Return-Code* attribute of the *API_out_in* parameter. This is the XFTAM-specified error code and is always returned. The resulting string is formatted for printing as a self-contained unit (for example, in an English language locale, it includes a terminating newline character). The user must pass a pointer to a location of type *OM_string* *. If the pointer in the referenced location is NULL, *ft_gperror*( ) will allocate storage for the string and return a pointer to it into the referenced location. Otherwise, the pointer in the referenced location must point to at least 256 octets of storage, the string will be stored in the locations pointed to.

vendor_string (OM_String (*))

A text string in the natural language of the current locale that represents the *Vendor-Code* attribute of the *API_out_in* parameter. This is an optional implementation-specific error code and is not returned if the *API_out_in* parameter did not contain an equivalent code. The resulting string is formatted for printing as a self-contained unit (for example, in an English language locale, it includes a terminating newline character). The comments regarding allocation of storage for the *return_string* return parameter also apply to this parameter. If no vendor string is available, the returned string will be zero-length (if the user supplies the storage), or the returned pointer will be NULL (if the allocation of storage is left to *ft_gperror*( )).

api_out (Private Object (API-Output-Parameters))

This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN VALUES

ft_gperror( ) returns either [FTE_SUCCESS] or one of the values listed below in ERRORS.

ERRORS

Operation Error Codes
- FTE_INV_RETURN_CODE
- FTE_INV_VENDOR_CODE

API Error Code
- FTE_CANCEL

NAME

ft_open - initialise a session with XFTAM and allocate a workspace

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_open(
    FT_package_t         package_list[ ],
    OM_private_object    *session ,
    OM_workspace         *workspace
);
```

DESCRIPTION

This function creates an XFTAM *instance* and performs any necessary initialisation of the API, including allocation of an XOM workspace for the storage of private objects defined by the *XFTAM Base package*. The function returns a private object of class *Session* which is used to identify the new XFTAM instance to other XFTAM functions.

ft_open must be invoked before any other XFTAM functions. It may be invoked multiple times from within a single program, in which case each call creates a separate XFTAM instance, returning a distinct session and associated workspace.

The *package_list* parameter may be used to negotiate the use of additional *optional* packages with this XFTAM instance. On return, the package list is modified to indicate which of the requested packages are available in the newly created workspace. (Failure to negotiate the use of an optional package does not cause the *ft_open*( ) function to fail - an application must check the returned array to determine if it can proceed without the use of any rejected packages.)

XFTAM does not currently define any optional packages. Additional packages may be defined in the future to provide optional extensions to the base XFTAM functionality. Alternatively, individual XFTAM implementations may define additional packages to support vendor extensions.

ARGUMENTS

package_list (Feature-List)

A sequence of proposed optional XOM *packages* to be included in the workspace created for this XFTAM instance. Each proposed package is represented by an *ft_package* structure. The sequence is terminated by an object identifier having no components (a length of zero and any value of the data pointer).

```
typedef struct {
    OM_object_identifier    package ;
    OM_boolean              included ;
} FT_package_t ;
```

On input, each package is identified by its allocated OSI object identifier. On output, if the result of the function is [FTE_SUCCESS], the list is updated to indicate, via the *included* field, whether each proposed package has been included in the workspace.

session (Private Object (Session))

Upon successful completion, this parameter returns a handle to an object of class *Session* which is used to identify the XFTAM instance created. The contents of this object are not modifiable by the API user. An object of this class must be passed to most other XFTAM functions in order to identify the XFTAM *instance* that is to process the request.

workspace (Workspace)

Upon successful completion, contains a handle to a workspace in which OM objects, defined by the *XFTAM Base Package* and any negotiated packages, can be created and manipulated. Private objects created in this workspace, and only such objects, may be used as arguments to the other XFTAM API functions.

RETURN VALUES

ft_open( ) returns either SUCCESS or one of the values listed below in ERRORS.

ERRORS

FTE_NO_RESOURCES
FT_NO_WORKSPACE

NAME

ft_rcvresult - receive result of an asynchronous operation invocation

SYNOPSIS

```
#include <xftam.h>

FT_return_code ft_rcvresult(
   OM_private_object        session ,

   OM_uint                  *completion_flag ,
   OM_uint32                *op_invoke_id ,
   OM_private_object        *op_api_out ,
   OM_private_object        *op_ftam_out ,
   OM_private_object        *op_result_object ,
   OM_private_object        *api_out
) ;
```

DESCRIPTION

This function is used to retrieve the completed result of a previous asynchronous operation invocation. The *completion flag* parameter indicates if an outstanding invocation has completed, if so the other parameters are used to return the *invocation identifier* and output object handles associated with the completed operation. The result of a completed outstanding invocation will be returned exactly once.

Asynchronous execution mode is an optional feature of XFTAM. If an implementation does not support this feature, *ft_rcvresult*( ) returns an error code of [FTE_NOTSUP_ASYNC].

ARGUMENTS

session (Private Object (Session))

This parameter is a handle for a private object of class **Session** which identifies the particular XFTAM *instance* that is to perform the required XFTAM operation. The session identifies the resources associated with the instance, including the XOM *workspace* that contains all private objects passed to or returned from this XFTAM function call.

completion_flag (Integer)

One of the following values to indicate the status of outstanding asynchronous operations:

- FT_COMPLETED_INVOKATION
  At least one outstanding invocation has completed and its result is made available.
- FT_OUTSTANDING_INVOKATIONS
  There are outstanding invocations, but none has yet completed.
- FT_NO_OUTSTANDING_INVOKATIONS
  There are no outstanding invocations.

Upon successful return with *completion_flag* having the value [FT_COMPLETED_INVOKATION], the *Invoke-ID* and the return object handles associated with the completed invocation are returned as noted below.

op_invoke_id (Integer)
This parameter is used to return the *Invoke-ID* of a completed operation. The value returned can be matched to those returned by oustanding asynchronous operations. It is only valid if the *completion_flag* parameter returned [FT_COMPLETED_INVOKATION].

op_api_out (Private Object (API-Output-Parameters))
This parameter is used to return an object handle for the *API-Output-Parameters* associated with the completed invocation. It is only valid if the *completion_flag* parameter returned [FT_COMPLETED_INVOKATION].

op_ftam_out (Private Object (FTAM-Output-Parameters))
This parameter is used to return an object handle for the *FTAM-Output-Parameters* associated with the completed invocation. It is only valid if the *completion_flag* parameter returned [FT_COMPLETED_INVOKATION] **and** *op_api_out* indicated that the asynchronous operation succeeded.

op_result_object (Private Object (Object))
This parameter is used to return an object handle for a possible output object associated with the completed operation. It is only valid if the *completion_flag* parameter returned [FT_COMPLETED_INVOKATION] and *op_api_out* indicated that the asynchronous operation succeeded and depending on the definition of the XFTAM function which intiated the asynchronous operation.

For example, if the function *ft_frattributes*( ) is invoked in asynchronous mode, this output parameter is used to return a pointer to the resulting *FTAM-Attributes* object returned by the read attributes operation.

api_out (Private Object (API-Output-Parameters))
This parameter is always returned and is a handle for a private object of class **API-Output-Parameters**. It returns API-specific output parameters for this function call.

RETURN CODES

ft_rcvresult( ) returns either [FTE_SUCCESS] or one of the values listed below under ERRORS.

ERRORS

FTE_VENDOR
FTE_NOTSUP_ASYNC
FTE_SESSION

Chapter 5

XFTAM Return Codes

This section lists the return codes defined by XFTAM and listed by the XFTAM functions. Each entry includes a description of the meaning of the code.

FTAM Error Codes

These codes indicate that the underlying FTAM implementation reported an error. The application must examine any FTAM diagnostics objects returned in order to obtain a detailed description for the failure.

[FTE_FTAM_CANCEL]
: This error is returned when XFTAM receives an F-CANCEL indication primitive during data transfer, indicating that the remote responder cannot continue with the data transfer. The responder may provide diagnostic structures to specify the reason for the abort. When XFTAM issues an F-CANCEL request to abandon a data transfer regime, the cause of the error is indicated by an alternative XFTAM return code.

[FTE_FTAM_PERMANENT]
: This type of error will occur every time the user attempts the requested operation. This type of error would be reported when, for example, the create password specified does not match the one held by the responder.

[FTE_PROVIDER_ABORT]
: This error is returned when XFTAM receives an F-P-ABORT indication primitive from the service provider. The primitive may have been initiated by either the initiating service provider (as a result of an state error or a communications error) or the responding service provider (as a result of a state error - of course communications errors detected by the responder cannot be reported to the initiator). The only way to determine which entity caused the abort is to examine the *source* parameters of any diagnostic structures associated with the abort primitive.

[FTE_USER_ABORT]
: This error is returned when XFTAM receives an F-U-ABORT indication primitive from the service provider, as a result of the responder issuing an equivalent request. The responder may provide diagnostic structures to specify the reason for the abort.

Operation Error Codes

These error codes indicate operation-related errors detected by the XFTAM API.

[FTE_ATTR_GRP_NOT_NEGOTIATED]
: Returned by a function when an optional FTAM attribute group, required by the requested operation, has not been negotiated by XFTAM for the FTAM regime (XFTAM always requests support of all attribute groups - this error occurs when the responder refuses to support one or more groups).

  For example, this error would be returned by *ft_fcattributes*( ) when an attempt is made to modify an attribute from the storage attribute group but the responder declines to support that group. Another example is where the API user specifies an values for the FTAM access passwords parameter (via the *Access-Passwords* XOM attribute of the *FTAM-Input-Parameters* object class) but support of the security attribute group is refused by the remote responder.

[FTE_FILE_EXISTS]
This error is returned by *ft_freceive*( ) when the destination file exists and the *dest_effect* function parameter is set to *fail*.

[FTE_FQOS_NOT_NEGOTIATED]
This error is returned when XFTAM is not able to negotiate the requested Quality of Service (specified in the FQoS attribute of the FTAM-Input-Parameters) for the the XFTAM operation. It indicates that the FTAM regime was established with a lower FQoS value than that requested, and the operation was terminated as a consequence. If the requested FQoS is not supported by the underlying FTAM intiator because of an XFTAM implementation restriction, [FTE_NOTSUP_FQOS] is returned instead.

[FTE_FUNCT_UNIT_NOT_NEGOTIATED]
Returned by a function when a functional unit, required by the requested operation, is not supported by the responder.

For example, this error would be returned by *ft_fcattributes()* when the responder does not support the enhanced file management functional unit.

[FTE_INV_ATTRIBUTE]
Returned by *ft_fcattributes*( ) when an attempt is made to change an attribute that FTAM defines as non-modifiable. Returned by *ft_fsend*( ) when an attempt is made to specify initial FTAM attributes that are not settable by the initiator.

For example, this error would be returned if the user specified a value for the *permitted actions* FTAM attribute (only settable at file creation) in a call to *ft_fcattributes*( ) or a value for *date and time of creation* (only settable by the responder) in a call to *ft_fsend*( ).

[FTE_INV_DOC_RCVD]
This error is returned by *ft_freceive*( ) when the content type of the file being received is not supported by the initiator. It may be possible to transfer the file successfully by specifying a valid simplification or relaxation of the actual file content type (via the *Content-Type* XOM attribute of the *Attributes* object class).

For example, this error is returned where the responder indicates that the file to be accessed is of type *FTAM-2* and the local XFTAM implementation does not support that document type. Another example is the case where the document type is FTAM-1 (which must be supported) but the *universal class* document type parameter is specified to be *TeletextString*, which may not be supported by the local implementation.

[FTE_INV_DOC_SPEC]
This error is returned by *ft_freceive*( ) and *ft_fsend*( ) when the API user specifies a content type for the file to be transferred (via the *Content-Type* XOM attribute of the *Attributes* object class) which is not supported by the local XFTAM implementation. In the send case, the content type refers to the type of file to be created in the reponder's filestore. In the receive case, it is the expected type of the file to be received. See [FTE_INV_DOC_RCVD] for examples.

[FTE_INV_RETURN_CODE]
Returned by *ft_gperror*( ) when the *Return-Code* XOM attribute of the *API-Output-Parameters* object is not a valid error code as defined by this specification.

[FTE_INV_STRING_LENGTH]
The value of the maximum-string-length parameter for FTAM document types may be restricted by an implementation of XFTAM. This error is returned if the user attempts to send or receive a document which exceeds the maximum supported value for this parameter. The error may be returned even if a value is not specified for the parameter, as

the default value is *unbounded*. The *minimum* value for the maximum-string-length that an XFTAM implementation must support is 134 characters for the FTAM-1 and FTAM-2 document types, and 512 octets for FTAM-3.

[FTE_INV_VENDOR_CODE]
Returned by *ft_gperror*( ) when the *Vendor-Code* XOM attribute of the *API-Output-Parameters* object is not a valid error code as defined by the particular implementation.

[FTE_LOCAL_FILE_ERROR]
Returned by *ft_freceive*( ) or *ft_fsend*( ) when XFTAM is unable to perform the required action in the local filestore for a reason other than one for which a specific error code is defined. The precise reason for the failure is indicated by the local operation system error reporting mechanism.

For example, if this error is returned by a call to *ft_freceive*( ) in a CAE-conformant system, the system error variable *errno* contains the CAE-defined error code which is set as a result of the attempt to create or overwrite the local destination file.

[FTE_LOCAL_PERMISSION]
Returned by *ft_freceive*( ) or *ft_fsend*( ) when XFTAM is unable to perform the required action in the local filestore for reasons of access control. In the send case, this may occur when the user specifies a *src_effect* of *move-file* (in which case XFTAM attempts to delete the local file once the transfer is complete - the error indicates that the local file *deletion* failed, though the *transfer* of the file has been completed successfully). In the receive case, the error is returned when the *dest_effect* parameter is set to *delete-file* (in which case XFTAM attempts to delete the local file *if it already exists*).

[FTE_NO_SRC_FILE]
This error is returned by *ft_send*( ) when the local file does not exist.

[FTE_NO_SUCH_INVOKATION]
This error is returned when an attempt is made to abandon an XFTAM operation which is not outstanding, or to cancel a synchronous operation while no such operation has been interrupted.

[FTE_SERV_CLS_NOT_NEGOTIATED]
Returned by a function when a service class, required by the requested operation, is not supported by the responder.

For example, this error would be returned by *ft_fcattributes()* when the responder does not support the file management service class.

API Error Codes

These error codes indicate errors detected by the XFTAM API, other than those directly related to the operation being requested.

[FTE_CANCEL]
This error code is returned by a synchronously-invoked function when *ft_abandon*( ) has been called from within an interrupt handler function to terminate the interrupted XFTAM operation.

[FTE_CONTEXT_MISMATCH]
The error code is returned when there is a mismatch in the use of connection related parameters, and includes the following conditions:

— When neither neither *p_address* or *Association-Id* is provided to identify the FTAM responder. *p_address* is used when context free operation is in use, and *Association-Id* identifies the FTAM association when context-sensitive operation is in use.

— When any connection related parameters are supplied in addition to *Association-Id*. When *Association-Id* is provided, the operation is context-sensitive and these are not provided.

[FTE_INV_ASSOC]
This error is returned when XFTAM is supplied with an *Association-Id* which does not represent an existing association within the XFTAM instance indicated by the session input parameter.

[FTE_INV_PADDRESS]
The *Presentation-Address* object passed to a file transfer or management function does not represent an valid address for the XFTAM operation. This error indicates that the object passed does not have a valid set of XOM attributes:

> A valid instance of the *Presentation-Address* class must contain, as a minimum, one N-Addresses attribute and one of the attributes *P-Selector*, *S-Selector* or *T-Selector*.

Alternatively, it indicates that none of the list of N-Addresses attributes contain a valid NSAP address. The NSAP address formats supported are a feature of the individual XFTAM implementation and the environment in which it is running.

[FTE_NO_RESOURCES]
This error code is returned when a request to execute an XFTAM operation cannot be completed due to a lack of some resource required to perform it. An implementation may use the *Vendor-Code* attribute of the *API-Output-Parameters* object to provide a more specific indication of the resource required to execute the operation.

[FTE_NO_WORKSPACE]
Returned by *ft_open*( ) if an XFTAM instance cannot be created because storage cannot be allocated for the associated XOM workspace.

[FTE_NOTSUP_ASYNC]
This error code is returned in the case where an implementation does not support asynchronous execution mode and the user attempts to invoke an XFTAM operation with the *Asynchronous* XOM attribute in *API-Input-Parameters* set to *TRUE*. If an implementation defines the value of the constant *FT_MAX_ASYNC_OPS* to be zero, asynchronous operations are not supported. If the constant *FT_MAX_ASYNC_OPS* is not defined to be zero, the support of asynchronous operations is mandatory for *ft_fsend*( ) and *ft_freceive*( ) but remains optional for all other functions.

[FTE_NOTSUP_FQOS]
This error code indicates that the level of recovery requested using the FQoS XOM attribute is not supported by this implementation of XFTAM. Support for FQoS values other than *No Recovery* is optional.

[FTE_NOTSUP_FTAM2]
This error indicates that the optional document type FTAM-2 is not supported by this implementation of XFTAM. It may be returned by *ft_freceive*( ) if the specified source file in the remote filestore is of type FTAM-2, or by *ft_fsend*( ) if the user requests that the file created in the remote filestore is to be of type FTAM-2.

[FTE_PENDING_OP]
The error is returned by XFTAM when an *ft_disconnect*( ) function is attempted on an association which is currently carrying active functions. *ft_abandon*( ) should be used to terminate the action, or *ft_abort*( ) used to abort the association.

[FTE_SESSION]
The *Session* object passed to a function in the *session* parameter does not identify a valid XFTAM instance. A *Session* object is returned when an XFTAM instance is created by a call to *ft_open*( ), and destroyed when the object is passed to *ft_close*( ).

[FTE_SYSTEM_ERROR]
An error related to the OS has occurred. The user should examine the Vendor-Code XOM attribute of the *API-Output-Parameters* return object to determine the precise error.

[FTE_TOO_MANY_OPS]
This error is returned if the API user attempts to invoke too many simultaneous asynchronous operations. The constant *FT_MAX_ASYNC_OPS* indicates how many such operations may be outstanding at any one time. It is the users responsibility to keep track of asynchronous operations and check against this number. If this error is returned, the user must wait until one of the current operations to complete and use *ft_rcvresult*( ) to return its results before further asynchronous operations may be invoked.

[FTE_VENDOR]
An error related to the specific implementation of XFTAM has occurred. The user should examine the *Vendor-Code* XOM attribute of the *API-Output-Parameters* return object to determine the precise error.

[FTE_XDS_ERROR]
An error related to the XDS has occurred. The user should examine the Vendor-Code XOM attribute of the *API-Output-Parameters* return object to determine the precise error.

Note: This is for XFTAM implementations that use the XDS *bind*( ) function to get a bound session.

[FTE_XOM_ERROR]
An error related to the XOM has occurred. The user should examine the Vendor-Code XOM attribute of the *API-Output-Parameters* return object to determine the precise error.

Appendix A

Summary of XOM

The XFTAM API makes use of facilities provided by the *OSI-Abstract-Data Manipulation API* to support the passing of control information between XFTAM and the API user. This API is fully described in the referenced **XOM** specification. However, a brief outline is presented here for the convenience of the reader of this specification.

Both specifications make use of the term *attribute*. XOM uses it to describe a value type which forms part of an XOM *Object Class*, whilst XFTAM uses it to describe a characteristic of an FTAM file. In this specification, in general, the particular meaning intended is clear from the context of its use. However, where this is not clear, the reference is qualified as *FTAM attribute* or *XOM attribute*. The term *XFTAM attribute* is used to refer to an XOM attribute of an object class defined by the *XFTAM package*. .

The description below introduces the various concepts that are used in object management, starting with the smallest.

A.1 Syntax

A *syntax* is the basis for the classification and representation of values in object management. Examples of syntaxes are Boolean, Integer, String(Octet) and Object.

Syntaxes are defined in the Object Management Specification, and nowhere else, and are themselves represented by integers.

A.2 Value

A *value* is a single datum, or piece of information. Each value belongs to exactly one syntax by which its representation is defined. A value may be as simple as a Boolean value (e.g. True), or as complicated as an entire XOM object (e.g. a Message).

A.3 XOM Attribute

An *XOM attribute type* is an arbitrary category into which a specification places some values.

OM attribute types are represented by integers, which are assigned in individual service specifications, and which are only meaningful within a particular package.

An *XOM attribute* is an XOM attribute type, together with an ordered sequence of one or more values. OM attributes can occur only as parts of an XOM object and the XOM attribute type, and values are constrained by the XOM class specification of that XOM object.

The XOM attribute type can be thought of as the name of the XOM attribute.

There is no general representation for an XOM attribute, but a descriptor represents an XOM attribute type together with a single syntax and value.

A.4 XOM Object

An *XOM object* is a collection of XOM attributes, the values of which can be accessed by means of functions. The particular XOM attribute types that may occur in an XOM object are determined by the XOM class of the XOM object, as are the constraints on those XOM attributes. The XOM class of an XOM object is determined when the XOM object is created, and cannot be changed.

OM objects are represented in the interface by a handle, or opaque pointer. The internal representation of an XOM object is not specified, though there is a defined data structure, called a *descriptor list*, which can also be used directly in a program.

A.5 XOM Class

An *XOM class* is a category of XOM object, set out in a specification. It determines the XOM attributes that may be present in the XOM object, and details the constraints on those XOM attributes.

Each XOM object belongs directly to exactly one XOM class, and is called an *instance* of that XOM class.

The XOM classes of XOM objects form a tree. Each XOM class has exactly one immediate *superclass* (except for the XOM class *Object*, which is the root of the tree), and each XOM class may have an arbitrary number of *subclasses*. The tree structure is also known as the *XOM class hierarchy*. The importance of the XOM class hierarchy stems from the inheritance property discussed below.

Each XOM class of XOM object has a fixed list of XOM attribute types, and every XOM object that is an instance of the XOM class has only these XOM attributes (some XOM attributes may not be present in particular instances, as permitted by the constraints in the XOM class specification). The list of XOM attribute types that may appear in instances of an XOM class has two parts. Each XOM class *inherits* all the XOM attribute types that are permitted in its immediate superclass as legal XOM attribute types. There is also a list of additional XOM attribute types that are permitted in the XOM class. Any subclasses of this XOM class will inherit all of these XOM attribute types, from both lists.

Because of inheritance, an XOM object is also said to be an instance of all its superclasses. It is required that the XOM class constraints of each superclass are met, considering just those XOM attribute types that are permitted in the superclass.

The XOM class hierarchy and the list of XOM attribute types for each XOM class are determined solely by the interface specification and cannot be changed by a program.

The XOM class specification may impose arbitrary constraints on the OM attributes. The most common of these are tabulated in the XOM class specification and are marked with a * below. Frequently encountered cases include constraints as follows:

- to restrict the syntaxes permitted for values of an XOM attribute (often to a single syntax) *
- to restrict the particular values to a subset of those permitted by the syntax
- to require one or more values of the XOM attribute (a *mandatory* XOM attribute) *
- to require either zero or more values of the XOM attribute (an *optional* XOM attribute) *
- to permit multiple values, perhaps up to some limit known as the *value number constraint* *

- to restrict the length of strings, up to a limit known as the *value length constraint* *.

Note: The constraint is expressed in terms of bits, octets or characters according to the kind of string. However, the lengths of strings are stated everywhere else in terms of the number of *elements*, which are either bits or octets. The number of elements in a string with multibyte characters (for example, T.61 Teletext) may thus exceed the value length constraint. (In C, an array with more bytes will be needed to store it.)

The constraints may affect multiple XOM attributes at once, for example a rule that only one of several XOM attributes may be present in any XOM object.

Every XOM object includes the XOM class to which it belongs as the single value of the mandatory XOM attribute type **Class**, which cannot be modified. The value of this XOM attribute is an OSI Object Identifier, which is assigned to the XOM class by the specification.

An *abstract class* is an XOM class of which instances are forbidden. It may be defined as a superclass in order to share XOM attributes between XOM classes, or simply to ensure that the XOM class hierarchy is convenient for the interface definition.

A.6 Package

A *Package* is a set of XOM classes that are grouped together by the specification, because they are functionally related.

A package is identified by an OSI Object-Identifier, which is assigned to the package by the specification. Thus the identity of each package is completely unique.

A.7 Package Closure

An XOM class may be defined to have an XOM attribute whose value is an XOM object of an XOM class defined in some other package. This is done to share definitions and to avoid duplication. For example, the *Directory Contents* package (reference **XDS**) defines an XOM class called **Teletex-Terminal-Identifier**. This XOM class has an XOM attribute whose value is an XOM object of XOM class **Teletex-NBPs**, which is defined in another package. An XOM class may also be a subclass of an XOM class in another package. These relationships between packages lead to the concept of a Package-Closure.

A *Package-Closure* is the set of classes which need to be supported in order to be able to create all possible instances of all classes defined in the package. A formal definition is given in the XOM specification.

A.8 Workspace

Details of the representation of XOM objects, and of the implementation of the functions that are used to manipulate them, are not specified because they are not the concern of the application programmer. However, the programmer sometimes needs to be aware of which implementation is being used for a particular XOM object.

The case of the XOM class **Teletex-NBPs** was mentioned above. This XOM class is used in both the Message Transfer Service and in the Directory Service. If an application uses both services, and the two services use different internal representations of XOM objects (perhaps because they are supplied by different vendors), then it is necessary for the application to specify which implementation should create a **Teletex-NBPs** OM object. This is done by means of a workspace.

A *workspace* is one or more Package-Closures, together with an implementation of the object management functions that supports all the XOM classes of XOM objects in the Package-Closures.

The notion of a workspace also includes the storage used to represent OM objects and management of that storage. The interested reader should refer to the relevant part of the OSI Object Management API Specification (reference **XOM**) for more details of how workspaces are implemented.

The application must obtain a workspace that supports an XOM class before it is able to create any XOM objects of that XOM class. The workspaces are returned by functions in the appropriate service. For example, *ft_shutdown*( ) returns a workspace that supports the XFTAM API package, whilst another function in another OSI service API returns a workspace that supports another package.

Some implementations may support additional packages in a workspace. For example, vendors may provide XOM classes for document types not supported by the base XFTAM API specification. The API user may negotiate these packages into a workspace when it is set up. Another important case is where two or more services are supported by the same implementation. In this case, the workspaces returned by *ft_shutdown*( ) and *ds_initialize*( ) are likely to have the same implementation. The application need take no account of this, but may experience improved performance.

A.9 Descriptor

A *descriptor* is a defined data structure that is used to represent an XOM attribute type and a single value. The structure has three components: a type, a syntax and a value.

A *descriptor list* is an ordered sequence of descriptors that is used to represent several XOM attribute types and values.

Where the list contains several descriptors with the same XOM attribute type (representing a multi-valued XOM attribute), the order of the values in the XOM attribute is the same as the order in the list. Such descriptors will always be adjacent.

Where the list contains a descriptor representing the XOM class, this must occur before any others.

A *public object* is a descriptor list that contains all the XOM attribute values of an XOM object, including the XOM class. Public objects are used to simplify application programs by enabling the use of static data structures instead of a sequence of XOM function calls.

A *private object* is an XOM object created in a workspace using the object management functions or the functions in an OSI service. The term is simply used for contrast with a public object.

A.10 Use of Objects

OM objects are used to represent the data collections used in the interface, such as *FTAM_Input_Parameters* defined in Section 3.20 on page 59.

An important feature of the interface is that an instance of a subclass can be used wherever a particular XOM class is needed. This means both that the application can supply a subclass and that the service can return a subclass. For example, the *FTAM-Attributes* class defined in Chapter 3 includes an XOM attribute which is itself an object, of class *Content-Type*. This is an *abstract class* and both the API user and XFTAM itself use it to refer to occurences of concrete sub-classes, such as *Document-Type-FTAM-1* for example.

Because the service may return a subclass of the specified XOM class, applications should always use the *om_instance*( ) function when checking the XOM class of an XOM object, rather than testing the value of the **Class** OM attribute.

When the application supplies a subclass of the specified XOM class as an argument, the service will either recognise them as vendor extensions or will ignore all XOM attribute types that are not permitted in the specified XOM class.

The application can generally supply either a public object or a private object as an argument of the interface functions. There are exceptions, such as where an argument must be a private object - in the interests of efficiency for example. The interface always returns private objects. The application can convert these into public objects by a call to *om_get*( ), if required.

Note that public objects returned by *om_get*( ) are read-only and must not be modified in any way.

Glossary

Some of the definitions listed here are closely based upon definitions from the FTAM specification. Where this is so, the description is preceded by the tag "FTAM". Others are taken from the XOM specification and are tagged "XOM".

abstract class
(XOM) An **XOM Class**, instances of which are forbidden.

attribute
(XOM) A component of an **object**, comprising an integer denoting the attributes's type and an ordered sequence of one or more attributes values, each accompanied by an integer denoting the value's syntax.

class
(XOM) A static grouping of **objects**, within a specification, based on both their semantics and their form.

concrete Class
(XOM) A **class**, instances of which are allowed.

data unit
(FTAM) The smallest unit of a file's contents which the filestore actions can manipulate.

descriptor
(XOM) A 'C' structure - the means by which the client and service exchange an attribute values and the interferes that denote its representation, type and syntax.

document type
(FTAM) The specification of a class of documents, which states their necessary semantics, abstract syntaxes and dynamics.

file[store] action
(FTAM) One of the actions specified as part of the definition of the virtual filestore.

file attributes
(FTAM) The name and other identifiable properties of a file.

initiator
(FTAM) That file service user which requests FTAM regime establishment.

instance
(XFTAM) An API user interacts with the XFTAM via an XFTAM *instance* which is the collection of state information required to perform XFTAM operations on the user's behalf.

instance
(XOM) An **object** in the category represented by an **class**.

object
(XOM) A composite information object comprising zero or more **attributes**.

operation
(XFTAM) A high-level file transfer or management task performed by an XFTAM function. An *operation* is implemented by the underlying FTAM service provider by performing a series of low-level **file actions**.

package
(XOM) A specified group of related **classes**, denoted by an Object Identifier.

responder
(FTAM) That file service user which accepts an FTAM regime establishment requested by the initiator.

private object
(XOM) An **object** that is represented in an unspecified fashion. Such an object is created in a **workspace** using the XOM functions.

public object
(XOM) A list of **descriptors** which contain all the **attributes** of an **object**.

virtual filestore
(FTAM) An abstract model for describing files and filestores, and the possible actions on them.

workspace
(XOM) A space in which **objects** can be created, together with an implementation of the functions which support the related **classes**.

Index