

IEEE Standard for Information Technology, OSI Application Program Interfaces—File Transfer, Access, and Management [C Language]

Sponsor

**Portable Applications Standards Committee
of the
IEEE Computer Society**

Approved December 13, 1994

IEEE Standards Board

Abstract: A C language Application Program Interface (API) to the OSI File Transfer, Access, and Management (FTAM) protocol is defined. The functions of the API implement high-level file transfer and file management operations using the service of an FTAM initiator and service provider that underlie the API. The API uses the services of the OSI Abstract Data Manipulation API (IEEE Std 122-1993) to facilitate passing of data between the API and the user.

Keywords: API, application layer, application portability, C language, file transfer, FTAM, OM, open systems, open systems interconnection, OSI, OSI Abstract Data Manipulation API

The Institute of Electrical and Electronics Engineers, Inc.

345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1995 by the Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1995.

Printed in the United States of America.

ISBN 1-55937-495-0

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE Standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Introduction

(This introduction is not a normative part of IEEE Std 1238.1-1994, IEEE Standard for Information Technology, OSI Application Program Interfaces—File Transfer, Access, and Management [C Language], but is included for information only.)

Purpose

This standard defines a C language Application Program Interface (API) to the 7 services of the OSI File Transfer, Access, and Management (FTAM) application-service-element. It enables application programmers to create portable FTAM file transfer and management applications. By defining a high-level API, this standard makes the services of OSI FTAM available to application programmers that do not need access to the full generality of the FTAM service and may not have the skills to use those services directly.

The FTAM API relies upon an implementation of the OSI Abstract Data Manipulation API, as specified in IEEE Std 1224-1993 {14}, to support the passing of control information between the API user and the FTAM API.

Related Standards Activities

There are no related standards activities at this time.¹

Extensions are approved as “amendments” or “revisions” to this document, following the IEEE Standards procedures.

Approved amendments are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such amendments are incorporated into their proper positions.

If you have any question about the completeness of your version, you may contact the IEEE Standards Department [+1 (908) 562-3800] to determine what amendments have been published.

If you have an interest in participating in the working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, USA, and ask to have this forwarded to the chairperson of the appropriate PASC working group.

¹A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, USA; Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

This standard was prepared by the P1238 Working Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the P1238 Working Group was as follows:

Portable Applications Standards Committee (PASC)

Chair:	Lowell Johnson
Vice Chairs:	Barry Needham
	Lorraine Kevra
	Andrew Josey
Treasurer:	Pete Meier
Secretary:	Charles Severance

P1238 Working Group Officials

Chair:	Tim Boland
Editor/Secretary:	Graham Jack

Working Group

Reed Adams	Steve Greismer	Curtis Royster
Dave Cannon	Petr Janecek	Larry Spieler
	Ed Kelly	

The following persons were members of the P1238 Balloting Group that approved the standard for submission to the IEEE Standards Board:

Reed Adams	Michel Gien	Thomas M. Kurihara
Robert Barned	Michael Hannah	Kevin Lewis
Jason Behm	Chris Harding	Paul Rabin
Andy Bihain	Barry Hedquist	Curtis Royster
Tim Boland	Petr Janecek	James Tanner
Steve Case	Judy Kerner	Mark-Rene Uchida
Hu Cheng	Lawrence Kilgallen	Keith S. Weir
Philip Enslow, Jr.	Martin J. Kirk	Oren Yuen

The IEEE Standards Board approved this standard on December 13, 1994 with the following membership:

Wallace S. Read, *Chair*
Donald C. Loughry, *Vice Chair*
Andrew G. Salem, *Secretary*

Gilles A. Baril
Bruce B. Barrow
Jose A. Berrios de la Paz
Clyde R. Camp
James Costantino
Stephen L. Diamond
Donald C. Fleckenstein
Jay Forster*
Ramiro Garcia

Donald N. Heirman
Richard J. Holleman
Jim Isaak
Ben C. Johnson
Sonny Kasturi
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight
Joseph L. Koepfinger*

D. N. "Jim" Logothetis
L. Bruce McClung
Marco W. Migliaro
Mary Lou Padgett
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal

James Beall
Richard B. Engelman

Robert E. Hebner

Rachel A. Meisel
IEEE Standards Project Editor

CLAUSE	PAGE
1. General	1
1.1 Scope	1
1.2 Normative References	2
1.3 Conformance	3
2. Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	5
3. Overview (Informative)	8
3.1 FTAM API Model	8
3.2 Feature Summary	9
3.3 Using the FTAM API	11
4. FTAM API Base Package OM Class Definitions	19
4.1 Package Definition	19
4.2 Class Definitions	20
4.3 Declaration Summary	35
5. Service Interface	39
5.1 Datatypes	39
5.2 Functions	39
5.3 Return Codes	72
5.4 C Identifier List	77
Annex A (Informative) Overview of OM	79
Annex B (Informative) Overview of FTAM	90
Annex C (Informative) Bibliography	95

IEEE Standard for Information Technology, OSI Application Program Interfaces—File Transfer, Access, and Management [C Language]

1. General

1.1 Scope

This standard defines an application program interface (API) to the services of the OSI File Transfer, Access, and Management (FTAM) application-service-element. The API provides access to a set of high-level operations. That is, API functions map on to a sequence of FTAM primitives that implement the required operation. Apart from association management functions, functions that map directly onto FTAM service primitives are not provided.

The API provides file transfer and file management operations. File access and filestore management operations are not provided. The API supports the FTAM initiator role. The FTAM responder role is not supported.

The API supports a context-free mode of operation (where the operation is performed using an FTAM association that is created specifically for the operation and destroyed once the operation is completed) and a context-sensitive mode of operation (where the FTAM association is created and destroyed independently, and operations are performed within the context of this independent association). In order to support the context-sensitive mode of operation, the API defines a set of association management functions that create and destroy FTAM associations.

An implementation conforming to this standard requires support of an implementation of the services defined in IEEE Std 1224-1993 {14}.

This standard is a “thick” C binding (that is, it defines both the semantics of the operations and the C language binding to those semantics).

An understanding of FTAM is assumed in the remaining parts of this specification. Those unfamiliar with FTAM may wish to read Annex B, Overview of FTAM (and perhaps Annex B.4.2, Overview of OM) before proceeding with the remainder of this standard.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

- [1] ISO 7498 :1984,¹ *Information processing systems—Open Systems Interconnection—Basic reference model.*
- [2] ISO 8571-1 :1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management—Part 1: General Introduction.*
- [3] ISO 8571-2 :1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management—Part 2: Virtual Filestore Definition.*
- [4] ISO 8571-3 :1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management—Part 3: File Service Definition.*
- [5] ISO 8571-4 :1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management—Part 4: File Protocol Specification.*
- [6] ISO 8571-5 :1990, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management—Part 5: Protocol Implementation Conformance Statement Proforma.*
- [7] ISO 8649 :1988, *Information processing systems—Open Systems Interconnection—Service definition for the association control service element.*
- [8] ISO 8824 :1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN. 1)*
- [9] ISO/IEC 9899 :1990, *Programming Languages — C.*
- [10] ISO/IEC ISP 10607-1:1990, *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.*
- [11] ISO/IEC ISP 10607-2:1990, *Information technology—International Standardized Profiles AFTnn—File Transfer, Access and Management—Part 2: Definition of document types, constraint sets and syntaxes.*
- [12] ISO/IEC ISP 10607-3:1990, *Information technology—International Standardized Profiles AFTnn—File Transfer, Access and Management—Part 3: AFT11—Simple File Transfer Service (unstructured).*
- [13] ISO/IEC ISP 10607-6:1991, *Information technology—International Standardized Profiles AFTnn—File Transfer, Access and Management—Part 6: AFT3—File Management Service.*
- [14] IEEE Std 1224-1993,² *IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation—Application Program Interface (API) [Language Independent]*
- [15] IEEE Std 1224.2-1993, *IEEE Standard for Information Technology—Directory Services—Application Program Interface (API) [Language Independent].*

¹ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, CH-1211, Geneve 20, Switzerland/Suisse.

²IEEE documents can be obtained from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ, 08855-1331, USA.

[16] IEEE Std 1327-1993, *IEEE Standard for Information Technology—Open Systems Interconnection (OSI) Abstract Data Manipulation C Language Interfaces—Binding for Application Program Interface (API)*.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

A conforming implementation for this standard shall meet all of the following criteria:

- 1) The implementation shall support all required behavior defined in this standard.
- 2) The implementation shall support all required interfaces defined in this standard.
- 3) The implementation may provide additional functions or facilities not required by this standard. Each such nonstandard extension shall be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of functions or facilities defined by this standard. The conformance document shall define an environment in which an application can be run with the behavior specified by this standard. In no case shall such an environment require modification of a strictly conforming application.
- 4) An implementation that complies with this specification shall also comply with the following International Standardized Profiles (ISPs):
 - ISO/IEC ISP 10607-1:1990 {10}
 - ISO/IEC ISP 10607-2:1990 {11}
 - ISO/IEC ISP 10607-3:1990 {12}
 - ISO/IEC ISP 10607-6:1991 {13}

The conformance document shall include a completed FTAM PICS (see ISO 8571-5 :1990 {6}), showing that the requirements of the ISPs are met by the implementation. The FTAM ISPs reference the ISO/IEC Session, Presentation, and ACSE protocols. When PICS Proforma for these protocols are available, the conformance document shall include completed PICS for these protocols showing that the relevant requirements of the ISPs are met by the implementation.

The following points should be noted:

- The service provider shall 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 shall accept values for all attributes but is not required to store those attributes that do not map to an equivalent local filestore file attribute.
- The service provider shall support the FTAM file attributes within the minimum attribute range defined in ISO 8571-2 :1988 {3}, section 15.
- Implementation of the error recovery procedures (and the associated RESTART and RECOVER functional units) is optional. Where the FTAM API user requests a level of FTAM quality of service (FQoS) that is not supported by the service provider, the requested FTAM API operation shall fail with the error code [FTE_NOTSUP_FQOS]. Where the requested FQoS is reduced during negotiation of the FTAM association, the operation shall fail with error code [FTE_FQOS_NOT_NEGOTIATED].

1.3.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to this standard. It shall have the same structure as this standard, with the information presented in the appropriately numbered sections, clauses, and subclauses. The conformance document shall not contain information about extended features or capabilities outside the scope of this standard.

The conformance document shall contain a statement that indicates the full names, numbers, and dates of the standards that apply.

The conformance document shall state which of the optional features defined in this standard are supported by the implementation.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this standard. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where this standard states that implementations may vary or where features are identified as undefined or unspecified.

No specifications other than those specified by this standard shall be present in the conformance document.

The phrases “shall document” or “shall be documented” in this standard mean that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

1.3.1.3 Conforming Implementation Options

The following features of the FTAM API 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.

- 1) *ft_abandon()* is supported if the implementation supports asynchronous operations.
- 2) *ft_rcvresult()* is supported if the implementation supports asynchronous operations.
- 3) **Asynchronous** OM Attribute is supported if the implementation supports asynchronous operations.
- 4) **Invoke-ID** OM Attribute is supported if the implementation supports asynchronous operations.
- 5) **Session-Handle** OM Attribute is supported if the implementation supports the FTAM API instances in the local event handling mechanisms.
- 6) *Source_attributes* or *return_attributes* parameter of *ft_freceive()* specifying **Document-Type-FTAM-2** is supported if the implementation supports the FTAM-2 document type.
- 7) *Initial_attributes* or *return_attributes* parameter of *ft_fsend()* specifying **Document-Type-FTAM-2** is supported if the implementation supports the FTAM-2 document type.
- 8) **FQoS** enum values other than **no-recovery** are supported if the implementation provides the corresponding level of recovery support.
- 9) *Contents type lookup service* is supported if the implementation is capable of determining a file contents type by *ft_fsend()* function when no contents type is specified by the FTAM API user.

1.3.2 Application Conformance

All applications claiming conformance to this standard shall fall within one of the following categories.

1.3.2.1 Strictly Conforming Application

A Strictly Conforming Application is an application that requires only the facilities described in this standard, and in the applicable language standard. Such an application shall accept any behavior described in this standard as unspecified or implementation defined, and for symbolic constants shall accept any value in the ranges permitted by this standard.

1.3.2.2 ISO/IEC Conforming Application

An ISO/IEC Conforming Application of this standard is an application that uses only the facilities described in this standard, in the applicable language standards, and in other language binding standards approved by ISO or IEC. Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other ISO or IEC standards used.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions of this standard is an application that differs from a Conforming Application only in that it uses nonstandard facilities that are consistent with this standard. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conforming Application.

2. Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographic Conventions

The `fixed width` font is used for:

- C language source code, when presented in SYNOPSIS sections or in extended code fragments.

The *italic* font is used for:

- C language function names, function argument names, data references, or code fragments, such as *package_list->package*. A function name is indicated by following parentheses, *ft_connect()*.

The **bold** font is used for:

- C language typedef names, such as **FT_package_t**.
- Concrete OM class names, such as **FTAM-Input-Parameters**.
- OM attribute names, such as **Initiator-Identity**.

The ***bold italic*** font is used for:

- Abstract OM class names, such as ***Document-Type-Text***.

Roman (normal) font capitals are used for:

- Symbolic constants defining an implementation limit or indicating support of an optional feature. They are surrounded by curly braces, such as {FT_MAX_ASYNC_OPS}.
- Symbolic constants for an error code returned by a function. They are surrounded by brackets, such as [FTE_NO_WORKSPACE].
- Symbolic constants, other than those covered above. They appear unadorned, such as FT_OVERWRITE.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions apply:

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in 1.3.1.2.

2.2.1.2 implementation defined: An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior [ISO/IEC 9945-1:1990 {B5}].

2.2.1.3 may: An indication of an optional feature [ISO/IEC 9945-1 :1990 {B5}].

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard, but can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used.

2.2.1.4 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate [ISO/IEC 9945-1 :1990 {B5}].

2.2.1.5 should:

- 1) With respect to implementations, an indication of an implementation recommendation, but not a requirement [ISO/IEC 9945-1 :1990 {B5}].
- 2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications [ISO/IEC 9945-1 :1990 {B5}].

2.2.1.6 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified by this standard (except that they do not return the error condition indicated for the unsupported case). If the functionality is not *supported*, the interface shall always return the indication specified for this situation [ISO/IEC 9945-1 :1990 {B5}].

2.2.1.7 system documentation: All documentation provided with an implementation, except the conformance document [ISO/IEC 9945-1 :1990 {B5}].

Electronically distributed documents for an implementation are considered part of the system documentation.

2.2.1.8 undefined: An indication that this standard imposes no portability requirements on the use by an application of an indeterminate value or its behavior with erroneous program constructs or erroneous data [ISO/IEC 9945-1 :1990 {B5}].

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in 1.3.2.3.

2.2.1.9 unspecified: An indication that this standard imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior [ISO/IEC 9945-1 :1990 {B5}].

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in 1.3.2.3.

2.2.2 General Terms

For the purposes of this standard, the following definitions apply:

2.2.2.1 abstract class: A class, instances of which are forbidden unless they belong to one of its concrete subclasses [IEEE Std 1224-1993 {14}].

2.2.2.2 (application-)association: A cooperative relationship between two application-entities, formed by their exchange of application-protocol-control-information through their use of presentation services [ISO 8649 :1988 {7}].

2.2.2.3 application-entity: The aspects of an application process pertinent to OSI [ISO 7498 :1984 {1}].

2.2.2.4 application-service-element: The part of an application-entity that provides an OSI environment, using underlying services when appropriate [ISO 7498 :1984 {1}].

2.2.2.5 attribute: A component of an object, comprised of an integer denoting the type of the attribute and an ordered sequence of one or more attributes values, each accompanied by an integer denoting the syntax of the value [IEEE Std 1224-1993 {14}].

2.2.2.6 class: A category into which objects are placed on the basis of both their purpose and their internal structure [IEEE Std 1224-1993 {14}].

2.2.2.7 concrete class: A class, instances of which are permitted either by direct instantiation or the instantiation of its concrete subclasses [IEEE Std 1224-1993 {14}].

2.2.2.8 context free: The mode of API operation in which the underlying FTAM initiator establishes an FTAM Regime for the sole purpose of executing the requested operation, closing the regime once the operation is complete.

2.2.2.9 context sensitive: The mode of API operation in which the underlying FTAM initiator performs the requested operation, using a pre-existing FTAM Regime that is established and maintained independently of individual operation invocations.

2.2.2.10 data unit: The smallest unit of the contents of a file that the filestore actions can manipulate [ISO 8571-1 :1988 {2}].

2.2.2.11 descriptor: The means by which the client and service exchange an attribute value and the integers that denote its representation, type, and syntax [IEEE Std 1327-1993 {16}].

2.2.2.12 document type: The specification of a class of documents, which states their necessary semantics, abstract syntaxes, and dynamics [ISO 8571-1 :1988 {2}].

2.2.2.13 filestore action: One of the actions specified as part of the definition of the virtual filestore [ISO 8571-1 :1988 {2}].

2.2.2.14 file attributes: The name and other identifiable properties of a file [ISO 8571-1 :1988 {2}].

2.2.2.15 (FTAM) regime: The initial regime negotiated between a pair of FTAM ASEs, within which a series of file selection regimes may be established to select or create individual files.

2.2.2.16 GraphicString: A value of the ASN.1 GraphicString restricted character string type.

2.2.2.17 initiator: The file service user that requests FTAM regime establishment [ISO 8571-1 :1988 {2}].

2.2.2.18 instance:

- 1) The collection of state information required to perform FTAM API operations on behalf of the user
- 2) An object in the category represented by a class [IEEE Std 1224-1993 {14}].

2.2.2.19 not present: When describing an argument to a function call, the case where a null pointer is supplied by the caller (as opposed to a pointer to an actual object of the type specified in the declaration).

2.2.2.20 object: Any of the complex information objects created, examined, modified, or destroyed by means of the [OM] interface [IEEE Std 1224-1993 {14}].

2.2.2.21 OCTET STRING: A value of ASN.1 type octetstring.

2.2.2.22 octetstring type: A simple type whose distinguished values are an ordered sequence of zero, one or more octets, each octet being an ordered sequence of 8 bits [ISO/IEC 8824-1990 {8}].

2.2.2.23 operation: A high-level file transfer or management task performed by an FTAM API function.

An operation is implemented by the underlying FTAM service provider, which performs a series of low-level file actions.

2.2.2.24 package: A group of related classes [IEEE Std 1224-1993 {14}].

2.2.2.25 present: When describing an argument to a function call, the case where a pointer to an actual object of the type specified in the declaration is supplied by the caller (as opposed to a null pointer).

2.2.2.26 presentation address: A name unambiguous within the OSI environment that is used to identify a set of presentation service access points.

2.2.2.27 responder: The file service user that accepts an FTAM regime establishment requested by the initiator [ISO 8571-1 :1988 {2}].

2.2.2.28 restricted character string type: A simple type whose values are strings of characters from some defined character set [ISO/IEC 8824-1990 {8}].

2.2.2.29 private object: The internal representation of the language-independent model of an object in a service, and thus unspecified [IEEE Std 1224–1993 {14}].

2.2.2.30 public object: The representation by the client of an object in a particular language binding [IEEE Std 1224–1993 {14}].

2.2.2.31 virtual filestore: An abstract model for describing files and filestores, and the possible actions on them [ISO 8571–1 :1988 {2}].

2.2.2.32 workspace: A repository for instances of classes in the closures of one or more packages associated with the workspace [IEEE Std 1224–1993 {14}].

2.2.3 Abbreviations

The following abbreviations are used in this standard:

2.2.3.1 API: Application Program Interface.

2.2.3.2 ASE: Application Service Element.

2.2.3.3 ISP: International Standardized Profile.

2.2.3.4 FQoS: FTAM Quality of Service.

2.2.3.5 FTAM: File Transfer, Access, and Management.

2.2.3.6 OM: OM API.

2.2.3.7 OM API: OSI Abstract Data Manipulation API.

2.2.3.8 OSI: Open Systems Interconnection.

2.2.3.9 PICS: Protocol Implementation Conformance Statement.

2.2.3.10 VFS: Virtual Filestore.

3. Overview (Informative)

NOTE — This section provides an overview of the FTAM API and is not normative. It does not contain requirements for conforming implementations.

3.1 FTAM API Model

The elements of the FTAM API model are shown in Figure 3.1. It should be noted that while the FTAM initiator and FTAM service provider are shown as separate boxes in the figure, this standard does not require that they be implemented as separate components. Also, the responder elements show a typical arrangement, and do not imply any requirement on the types of responder architecture with which an implementation of the FTAM API may interwork.

The FTAM API provides a set of functions that implement high-level file transfer and management tasks. These are termed *FTAM API operations* in this standard. An operation can be a request to transfer a file from the local filestore to an FTAM filestore, or to read the attributes of a file in an FTAM filestore.

Underlying the API is an FTAM initiator that implements the FTAM API operations using the services of an FTAM service provider. In order to implement an operation, the local initiator establishes an application association with an FTAM responder that serves the FTAM 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 the FTAM API.

In the alternative operating mode, *context sensitive operation*, the FTAM initiator establishes the application association with the FTAM responder (using the INITIALIZE FTAM service to establish an FTAM regime), and this association remains in place to provide a context for subsequent operations (for example, invoking the delete file services). When each operation completes, it returns the result to the API user and the association remains in existence, 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 interacts with the local filestore in order to execute file transfers requested by the FTAM API user. In doing so, it maps between the attributes and contents of the file as defined by the FTAM VFS and those defined by the local filestore. The initiator uses the FTAM service to set up an association with an FTAM responder that provides access to the VFS. 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 FTAM responder implementation.

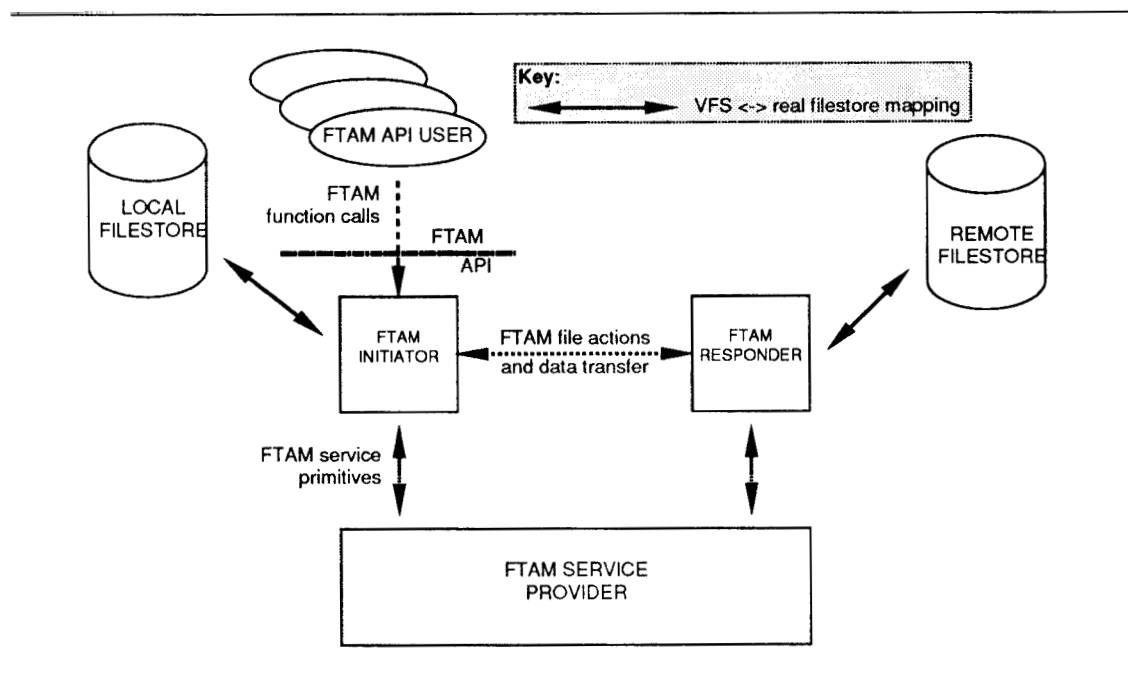


Figure 3.1—FTAM API Model

A user interacts with the API via an *FTAM API instance*, which is the collection of state information required to perform the FTAM API operations on the user's behalf. FTAM API instances are created and deleted by the API user as required. A user may create many independent FTAM API instances within a single program.

3.2 Feature Summary

The FTAM API provides a set of functions to support FTAM file transfer and management operations. That is, the FTAM API allows an application to transfer complete files to and from an FTAM 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 an FTAM filestore (this function relies upon the NBS-9 document type,

a feature of FTAM defined in the FTAM Profile {11}, which may not be supported by the underlying initiator or the responder that provides access to the FTAM filestore).

The FTAM API provides the following functions and features.

3.2.1 API Support Functions

<i>ft_open()</i>	This function creates an FTAM API instance and allocates an OM workspace for the associated package(s).
<i>ft_close()</i>	This function destroys an FTAM API instance and frees the associated workspace.
<i>ft_connect()</i>	This function creates an association with an FTAM responder.
<i>ft_disconnect()</i>	This function destroys an association established by <i>ft_connect()</i> .
<i>ft_abort()</i>	This function aborts an association and any outstanding asynchronous operations.
<i>ft_abandon()</i>	This function abandons an outstanding asynchronous operation (optional).
<i>ft_rcvresult()</i>	This function receives the results of an outstanding asynchronous operation (optional).
<i>ft_gpererror()</i>	This function translates an FTAM API return code into a printable string.

3.2.2 File Transfer Functions

<i>ft_fsend()</i>	This function sends a file from the local filestore to an FTAM VFS.
<i>ft_freceive()</i>	This function receives a file from an FTAM VFS into the local filestore.

3.2.3 File Management Functions

<i>ft_frattributes()</i>	This function reads the attributes of a file in an FTAM VFS.
<i>ft_fcattributes()</i>	This function changes the attributes of a file in an FTAM VFS.
<i>ft_fdelete()</i>	This function deletes a file in an FTAM VFS.

3.2.4 Miscellaneous Functions

ft_frdir() This function returns a list of entries from a directory in an FTAM VFS.

3.2.5 Document Types

The FTAM API 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 profiles referenced by this standard. The FTAM implementation, or the responders with which it interacts, may not support it. The FTAM API does not support file access operations on individual parts of an FTAM-2 document. The FTAM API optionally supports the NBS-9 document type for listing contents of a directory on an FTAM filestore.

3.2.6 FTAM API Base Package

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

3.2.7 Supporting APIs

The FTAM API relies upon an implementation of the OM API (IEEE Std 1224-1993 {14}) to support the passing of control information between the API user and the FTAM API.

Also, optionally, the user of the FTAM API may use an implementation of the Directory Service API (IEEE Std 1224.2-1993 {15}) to obtain addressing information needed to locate an FTAM responder.

3.3 Using the FTAM API

This clause discusses specific aspects of usage that the user must understand in order to use the FTAM API successfully.

3.3.1 Include files

The macros and constant definitions that support the use of the FTAM API are collected in a header file that should be included in each source file that makes use of the FTAM API functions. In addition, the header file for the OM API should 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 from both APIs.

```
#include <xom.h>           /* definitions for the OM API */
#include <xftam.h>          /* definitions for the FTAM API */
```

3.3.2 Library files

Typically, an implementation of the FTAM API provides a library that 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.

3.3.3 FTAM API Instances

Before an API user can invoke file transfer and management operations, an FTAM API instance must be created by a call to *ft_open()*, which initializes the associated resources (such as an OM workspace). The function 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 FTAM API functions.

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

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

3.3.4 Addressing the Remote Filestore

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

The mechanism by which the API user obtains the presentation address identifying a particular FTAM responder is not defined by the FTAM API (for example, it may be by a local look-up service or a distributed directory service). The **Presentation-Address** object class defined in the Directory Service package of IEEE 1224.2-1993 {15} is used by the FTAM API to facilitate the transfer of presentation addresses between the Directory Service API and the FTAM API.

NOTE — The FTAM API does not require the use of the Directory Service API for obtaining presentation addresses.

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

3.3.5 Filenames

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

3.3.6 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 FT_COPY_FILE, where the source file for the transfer is left in place when the transfer is complete; or FT_MOVE_FILE, 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 3.3.15). 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 FTAM API.

3.3.7 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 FT_FAIL, FT_EXTEND, or FT_OVERWRITE.
- For files being sent: The possible values for this enumeration correspond to the values of the FTAM override parameter (that is FT_CREATE_FAILURE FT_SELECT_OLD_FILE, FT_DELETE_AND_CREATE_WITH_OLD_ATTRIBUTES, and FT_DELETE_AND_CREATE_WITH_NEW_ATTRIBUTES. FT_SELECT_OLD_FILE has the effect of appending the contents of the local file to those of the target one. For both FT_SELECT_OLD_FILE and FT_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 that 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 5.2.11). 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 FTAM API.

3.3.8 Synchronous and Asynchronous Processing Modes

An FTAM API file transfer or management function performs a high-level operation (for example, file transfer to an FTAM filestore) that is implemented by a series of low-level file actions. By default, when an FTAM API function is called, execution control is passed to the API, which only returns control to the caller once the requested operation 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 the FTAM API. When handling an interrupt, 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 value of the *invoke_id* parameter. The interrupted FTAM API function will then return the error code [FTE_CANCEL].

Where the API user is to perform other activities while the FTAM API operation is in progress (for example, an application might initiate several concurrent file transfers), the FTAM API 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 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 the *ft_fsend()* and *ft_freceive()* functions. Support for all other functions remains optional. When asynchronous, the FTAM API operations are permitted and 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 FTAM API function asynchronously causes the function to return the error code [FTE_NOTSUP_ASYNC] if the implementation does not 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 the FTAM API functions. A subsequent call to the *ft_rcvresult()* function returns the result of the FTAM API operation invocation, along with these output objects.

The asynchronous mode of processing is selected by setting the **Asynchronous** attribute of the **API-Input-Parameters** object to FT_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 among the concurrent requests outstanding for this particular FTAM API instance, and can be matched to the invocation identifier returned by a subsequent call to the *ft_rcvresult()* function to indicate that the associated FTAM API 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** value 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 FTAM API operations that return a result object (for example, the **FTAM-Attributes** object returned in the *return_attributes* output parameter of the *ft_fattributes()* function).

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

- 1) Call the required FTAM API function, passing it an **API-Input-Parameters** object with the **Asynchronous** attribute set to FT_TRUE.
- 2) Check the function return code to confirm that the asynchronous operation has been invoked successfully.
- 3) Save the value returned in the **Invoke-ID** attribute of the **API-Output-Parameters** output object to identify the completion of the operation invocation.
- 4) Call the *ft_rcvresult()* function to receive the invocation identifier of a completed operation invocation. The function returns a completion flag that may be FT_COMPLETED_INVOCATIONS to indicate that a completed invocation has been returned.
 NOTE — The **Session-Handle** attribute, optionally returned as part of the **Session** object that identifies an FTAM API instance, can be used in conjunction with the local event notification mechanisms to determine when an asynchronous operation has completed (see 3.3.11).
- 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.

3.3.9 Context-Free and Context-Sensitive Processing Modes

As outlined in the overview, the FTAM API supports two processing modes—context free and context sensitive.

To use the FTAM API for context-free operation, all the parameters needed to carry out identification of the FTAM file store, authentication and negotiation of service levels are provided with each operation. This is the default mode of operation for both synchronous and asynchronous operations.

However, when a number of 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.

In this mode an association is created with the FTAM responder using the *ft_connect()* function (which maps onto the FTAM F-INITIALIZE service). The function takes as input the information needed to identify the responder, complete the authentication, and negotiate service levels. Upon completion, *ft_connect()* returns an association identifier for use in subsequent operations to identify the association created.

To use an existing association, subsequent API functions supply the association identifier as an input parameter, in addition to the parameters needed to complete the file transfer or file management operation. The operation is then completed using the identified association, which remains established after the operation is completed. In the same way, *ft_abandon()* abandons an active file operation that is using the association but leaves 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 this mode of processing is mandatory. However, the number of concurrent FTAM associations that may be created by a single API instance may be limited by an implementation using the {FT_MAX_ASYNC_OPERATIONS}, which limits the number of concurrent asynchronous operations. (A value of zero for {FT_MAX_ASYNC_OPERATIONS} is a special case, having the same meaning as the value one as far as support of FTAM associations is concerned).

3.3.10 Interrupt Handling

With the exceptions of the two functions listed below, the functions of the FTAM API are not required to be re-entrant and should not be called when handling an interrupt.

- *ft_abandon()* may be called to terminate an interrupted synchronous operation as described in 3.3.8. This mechanism enables the API user to install a watchdog timer.
- *ft_close()* may be called for workspace shutdown if the API user intends to terminate the FTAM API application. If control returns to an interrupted synchronous FTAM API function after calling *ft_close()*, the function returns the error code [FTE_SESSION] to indicate that an workspace shutdown occurred during its execution.

3.3.11 Session Handle

Where an implementation supports it, the **Session** object, returned by *ft_open()* to identify the created FTAM API 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 FTAM API 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 implementation defined.

3.3.12 Result Reporting

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

3.3.12.1 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 FTAM API function, or return the reason why it failed. It may indicate a failure detected by the API, the underlying FTAM initiator or service provider, or the FTAM responder or its local filestore. Values are defined by this specification, along with the meaning of the code and a possible action, in 5.3.

The values returned in the vendor code (returned as the **Vendor-Code** attribute of the **API-Output-Parameters** class) are not specified. The system 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.

3.3.12.2 FTAM-level Result Reporting

This level of result reporting is specified by the ISO 8571 series of standards, and is based upon diagnostic structures that may be returned as parameters in FTAM service primitives. The FTAM API 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 service user. Because a single FTAM API function may result in a series of FTAM service primitives being issued, more than one of these structures may be returned from a function.

3.3.13 VFS Mapping

The FTAM API is responsible for accessing the local file system when transferring files to and from an FTAM 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.

3.3.13.1 Mapping Files Received

When receiving a file, the FTAM API 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 contents of the FTAM file into lines of text in the local file, using the local convention for representing the end-of-line. For implementations that support the optional FTAM-2 document type, the mechanism for representing the records of the file in the local filestore is implementation defined. See 5.2.10 for detailed information about mapping received files into the local filestore.

3.3.13.2 Mapping Files Sent

For files sent, by default a file is transferred as an FTAM-1 file with the text lines from the local file delimited by the format effectors <CR> and <LF>. 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 5.2.11 for detailed information about mapping sent files from the local filestore.

3.3.14 FTAM Attributes

The FTAM VFS defines a set of file attributes that are used to describe the characteristics and contents of a file (see B.4). As an FTAM initiator, an implementation of the FTAM API supports all file attribute groups. However, the capabilities of the local filestore and that of the responder may limit the attributes that can be specified or returned for a file (see below for individual cases).

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

FTAM attributes are collected into groups. A particular FTAM responder must support the kernel group of attributes, but support of the other groups is optional. When the FTAM API establishes an FTAM regime with a responder, it negotiates which attribute groups are to be supported for this association. The 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 the value of an attribute from that 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 FTAM API functions either accept or return attribute information. The following description highlights salient aspects of the way that some of the FTAM API functions utilize the **FTAM-Attributes** object.

3.3.14.1 ft_fattributes

The class **New-Attributes** subclass of **FTAM-Attributes** is used to specify the modified file attributes to be set for a file in an FTAM filestore. The **FTAM-Attributes** class is used to return the resulting values that the 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 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 responder as a result of modifying the attributes of the file. These can be interrogated to determine the effect of the request. Return of attribute information as a result of the change attribute file action is optional for the responder.

3.3.14.2 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 that 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.

3.3.14.3 *ft_freceive*

On input, only the **Content-Type** OM 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.

3.3.14.4 *ft_send*

For *ft_send()*, the **FTAM-Attributes** class is used both to specify the initial attribute values proposed for the file being created in the FTAM filestore, and to return the actual values that the 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 responder:

- **Filename.** The filename for the created file is specified using the *dest_filename* parameter to the function. The equivalent OM attribute in the **FTAM-Attributes** class is ignored if present.
- **Content type.** If not specified by the user, the FTAM API either determines the type of the local file, or assumes a default document type of FTAM-1, as specified in the *ft_send()* manual page in 5.2.11.

The output attributes are those returned by the 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.

3.3.14.5 *ft_rdir*

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

3.3.15 Access Control

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

3.3.15.1 File Access Control

The access control file attribute provides a set of conditions that may be attached to each of the FTAM actions permitted for a file. A condition consists of an optional password that may be attached to the action, that 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. The FTAM API provides the **Access-Control** object class to set and retrieve the value of this file attribute. Figure 3.2 shows how a list of such objects can be used to specify a set of access conditions for a file, allowing anyone running the identified application and possessing the appropriate passwords to access its contents or attributes, but limiting file modification actions to user “Captain Cat”.

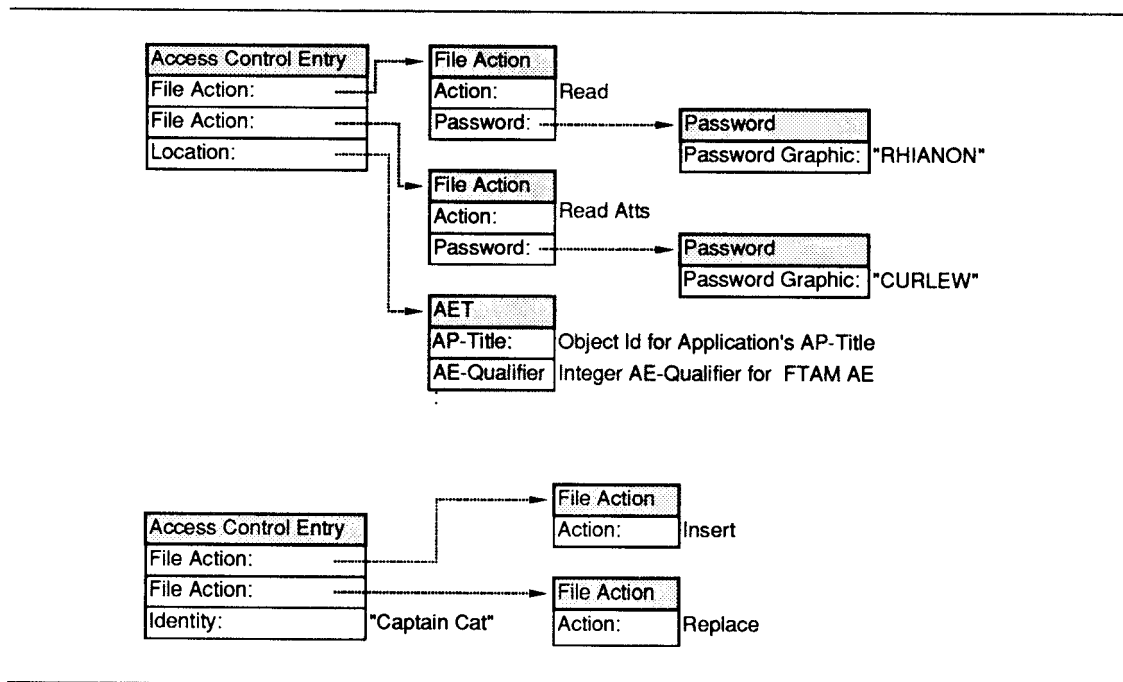


Figure 3.2—Access-Control Attribute OM Objects

3.3.15.2 Filestore Access Control

When an FTAM service provider establishes an FTAM regime with a 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 responder. (For example, a responder on a POSIX.1-conformant system might use the FTAM identity parameter as the login user name, with the associated password matched against the FTAM filestore password parameter.)

The FTAM API 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.

3.3.15.3 Access Passwords

When the FTAM service provider identifies a file in the filestore of the responder 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 FTAM API 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 authorize the requested actions in the case where the access control file attributes of the selected file require a password to be supplied for a particular action.

The FTAM API 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.

4. FTAM API Base Package OM Class Definitions

This section defines the OM classes that make up the FTAM API Base Package. The classes are listed in alphabetical order, in the form defined by IEEE Std 1224-1993 {14}. (summarized in Annex A, Overview of OM). To avoid repetitive detail, an OM class is assumed to be a subclass of the OM class **Object** unless otherwise stated. Similarly, an OM class is assumed to be a concrete class unless otherwise stated.

4.1 Package Definition

This standard defines a single OM package that contains all the classes required by the FTAM API. The package is known as the FTAM API Base Package, and is uniquely identified to OM by the object identifier:

ISO(1) National Body Member (2)us (840) IEEE-1238.1 (10026)

The class hierarchy for the FTAM API Base Package is given below, using indentation to show the relationship between classes. Classes listed in bold type are concrete classes, while those in bold/italic type are abstract classes. Those marked “(Read Only)” are used by the FTAM API 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 — The **Presentation-Address** class is defined in the Directory Services (DS) package defined in IEEE Std 1224.2-1993 {15}.

4.2 Class Definitions

4.2.1 Access-Control-Element

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

OM 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	-

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 OM 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.

4.2.2 AE-Title

This OM 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 the FTAM API. 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.

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

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.

4.2.3 API-Input-Parameters

This OM class is used to pass API-specific parameters to an FTAM API function call.

OM Attribute	Value Syntax	Value Length	Value Number	Value Initially
Association-ID	Integer	-	0-1	-
Asynchronous	Boolean	-	0-1	-

Association-ID

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

If this parameter is not present, the API functions operate in context-free mode, creating and destroying an association to perform the requested operation.

If this parameter is present, context-sensitive processing mode is selected and the API functions do not require the *p_address* parameter and the **FTAM-Input-Parameters** attributes that relate to establishing an FTAM regime. If these parameters are supplied, an error code [FTE_CONTEXT_MISMATCH] is returned by the service.

Supplying an invalid value for **Association-ID** causes the error code [FTE_INV_ASSOC] to be returned.

Asynchronous

If set to FT_TRUE the associated FTAM API 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 set to FT_FALSE, the function is executed synchronously.

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. A request that would exceed this number causes the function to return the error code [FTE_TOO_MANY_OPS]. An attempt to invoke an FTAM API function asynchronously causes the function to return the error code [FTE_NOTSUP_ASYNC] if the implementation does not support asynchronous operation (either generally—{FT_MAX_ASYNC_OPS} is defined to be zero—or specific to this function).

4.2.4 API-Output-Parameters

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

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

Return-Code

The result code for an FTAM API function. The values for this OM attribute are listed in 5.3. Each FTAM API function lists its possible return codes.

Vendor-Code

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

Invoke-ID

This attribute is returned if the API user requested that the FTAM API operation associated with the called function is to be executed in asynchronous mode. Its value may be matched against invocation identifier values returned by subsequent calls to *ft_rcvresult()* to determine when the associated FTAM API operation has completed. This attribute is only returned if the **Return-Code** attribute is [FTE_SUCCESS].

4.2.5 Charging

This OM class is mapped from the FTAM charging parameters that may be returned by a number of FTAM confirmation service primitives. It conveys information on the costs incurred during an FTAM regime.

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

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 that the **Charge-Value** attribute measures. The contents of this string are specific to the responder implementation accessed.

Charge-Value

The number of charging units that have been accumulated for use of the identified resource.

4.2.6 Content-Type

This OM class identifies the abstract datatypes used to represent the content of an FTAM file, plus other structuring information necessary to maintain its structure and semantics.

FTAM defines two forms in which this information can be specified. The FTAM API supports only the “document type” form. **Content-Type** is an abstract OM class with no specific OM attributes.

There is only one subclass defined: **Document-Type**.

4.2.7 Directory-List

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

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

Directory-Entry

This OM Attribute is used to return the FTAM file attributes for a single file in the source directory.

4.2.8 Document-Type

This OM class is a subclass of **Content-Type**. It represents the document-type form of the equivalent FTAM file attribute. Subclasses are defined for each supported document type. Two abstract subclasses are currently defined in this specification, representing generic document types that 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 the FTAM API (no information is returned about the values of any parameters that may be associated with an unsupported document type).

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

Type-Name

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

4.2.9 Document-Type-Text

This abstract OM class is a subclass of the **Document-Type** OM class. It defines OM attributes for the parameters common to the FTAM text document types (FTAM-1 and FTAM-2). Subclasses are defined for the concrete classes **Document-Type-FTAM-1** and **Document-Type-FTAM-2**.

OM 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	-

Content-Class

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

- FT_PRINTABLE_STRING (Optional)
- FT_TELETEX_STRING (Optional)
- FT_VIDEOTEX_STRING (Optional)
- FT_IA5_STRING (Mandatory for FTAM-1 files)
- FT_GRAPHIC_STRING (Mandatory)
- FT_VISIBLE_STRING (Optional)
- FT_GENERAL_STRING (Mandatory for FTAM-1 files and Optional for FTAM-2 files)

If this OM attribute is not present, the character set is GeneralString.

Note that for file transfer operations, support of IA5String and GeneralString is mandatory for FTAM-1 files, and support of GeneralString is mandatory for FTAM-2 files. Support of the other classes is optional in each case. That is, an implementation 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 the strings of a file. If this OM attribute is omitted, the string length for this file is unlimited. The *minimum* value that an FTAM API 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 that exceeds the maximum supported value for this parameter.

String-Significance

This parameter defines the significance of the string length for this file. Allowed values are:

- FT_VARIABLE. The length of the character strings is less than or equal to String-Length.
- FT_FIXED. The length of the character strings is exactly equal to String-Length.
- FT_NOT_SIGNIFICANT. The boundaries of the character strings are not necessarily preserved when the file is stored.

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

4.2.10 Document-Type-FTAM-1

This OM class is a subclass 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.

4.2.11 Document-Type-FTAM-2

This OM class is a subclass 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** OM attribute to point to an object of this type may cause an FTAM API function to return the error code [FTE_NOTSUP_FTAM2].

4.2.12 Document-Type-Binary

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

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

String-Length

Maximum length of the strings of a file. If this OM attribute is missing, the string length for this file is unlimited. The *minimum* value for the maximum-string-length that an FTAM API 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 that exceeds the maximum supported value for this parameter.

String-Significance

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

- FT_VARIABLE. The length of the character strings is less than or equal to String-Length.
- FT_FIXED. The length of the character strings is exactly equal to String-Length.
- FT_NOT-SIGNIFICANT. The boundaries of the character strings are not necessarily preserved when the file is stored.

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