The ISO Development Environment: User's Manual

Volume 3: Applications

Marshall T. Rose
Performance Systems International, Inc.

Julian P. Onions
X-Tel Services Ltd

Colin J. Robbins
X-Tel Services Ltd

February 22, 1994
(Version 7.0)

Contents

[0]This document (version #6.18) was LaTeXset February 22, 1994 with lplain v2.09-March 25, 1992.

List of Tables

List of Figures

Preface

The software described herein has been developed as a research tool and represents an effort to promote the use of the International Organisation for Standardisation (ISO) interpretation of open systems interconnection (OSI), particularly in the Internet and RARE research communities.

Notice, Disclaimer, and Conditions of Use

In particular, the Northrop Corporation provided the initial sponsorship for the ISODE and the Wollongong Group, Inc., also supported this effort. The ISODE received partial support from the U.S. Defense Advanced Research

Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602–88–C–0016 to NYSER-Net Inc.

Revision Information

This document (version #6.18) and its companion volumes are believed to accurately reflect release v 7.0 of February 22, 1994.

Release Information

If you would like a copy of the release described in this document, there are several avenues available:

- NORTH AMERICA
  For mailings in NORTH AMERICA, send a check for 375 U.S. Dollars to:

Postal address:	University of Pennsylvania Department of Computer and Information Science Moore School Attn: David J. Farber (ISODE Distribution) 200 South 33rd Street Philadelphia, PA 19104-6314 U.S.A.
Telephone:	+1 215 898 8560

  Specify one:

  1. 1600bpi 1/2–inch tape, or
  2. Sun 1/4–inch cartridge tape.

  The tape will be written in *tar* format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- EUROPE
  For mailings in EUROPE, send a cheque or bankers draft and a purchase order for 200 Pounds Sterling to:

Postal address:	Department of Computer Science Attn: Natalie May/Dawn Bailey University College London Gower Street London, WC1E 6BT UK

For information only:

Telephone:	+44 71 380 7214
Fax:	+44 71 387 1397
Telex:	28722
Internet:	`natalie@cs.ucl.ac.uk`
	`dawn@cs.ucl.ac.uk`

Specify one:

1. 1600bpi 1/2–inch tape, or
2. Sun 1/4–inch cartridge tape.

The tape will be written in *tar* format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- EUROPE (tape only)
  Tapes without hardcopy documentation can be obtained via the European Forum for Open Systems (EurOpen, formerly known as EUUG). The ISODE 7.0 distribution is called EurOpenD14.

Postal address:	EurOpen Software Distributions
	c/o Frank Kuiper
	Centrum voor Wiskunde en Informatica
	Kruislann 413
	1098 SJ Amsterdam
	The Netherlands

For information only:

Telephone:	+31 20 5924056
	(or +31 20 5929333)
Telex:	12571 mactr nl
Telefax:	+31 20 5924199
Internet:	`euug-tapes@cwi.nl`

Specify one:

1. 1600bpi 1/2–inch tape: 140 Dutch Guilders.

2. Sun 1/4–inch cartridge tape (QIC-24 format): 200 Dutch Guilders.

If you require DHL this is possible and will be billed through. Note that if you are not a member of EurOpen then there is an additional handling fee of 300 Dutch Guilders (please enclose a copy of your membership or contribution payment form when ordering). Do not send money, cheques, tapes or envelopes, you will be invoiced.

- PACIFIC RIM
  For mailings in the Pacific Rim, send a cheque for 300 dollars Australian to:

  Postal address: Isode Distribution
  (Attn: Andrew Waugh)
  723 Swanston St
  Carlton, VIC 3053
  Australia

  For information only:

Telephone:	+61 3 282 2615
Fax:	+61 3 282 2600
Internet:	`ajw@mel.dit.csiro.au`

  Specify one:

  1. 1600/3200/6250bpi 1/2-inch tape, or
  2. Sun 1/4-inch cartridge tape in either QIC-11, QIC-24 or QIC-150 format.
  3. Exabyte 2.3 Gigabyte or 5 Gigabyte format

  The tape will be written in *tar* format and returned with a documentation set. Do not send tapes or envelopes. Documentation only is the same price.

- Internet
  If you can FTP to the Internet, you can use anonymous FTP to the host `uu.psi.com [136.161.128.3]` to retrieve `isode-7.tar.Z` in BINARY mode from the `isode/` directory. This file is the *tar* image after being run through the compress program and is approximately 4.7MB in size.

- NIFTP
  If you run NIFTP over the public X.25 or over JANET, and are registered in the NRS at Salford, you can use NIFTP with username "guest" and your own name as password, to access `UK.AC.UCL.CS` to retrieve the file `<SRC>isode-7.tar.Z`. This file is the *tar* image after being run through the compress program and is approximately 4.7MB in size.

- FTAM on the JANET, IXI or PSS
  The source code is available by FTAM at the University College London over X.25 using

  JANET (DTE `00000511160013`)

  IXI (DTE `20433450420113`)

  PSS (DTE `23421920030013`)

  Use TSEL `259` (ASCII encoding). Use the "anon" user-identity and retrieve the file `isode-7.tar.Z` from the `src/` directory. This file is the *tar* image after being run through the compress program and is approximately 4.7MB in size.

  The file service is provided by the FTAM implementation in ISODE 6.0 or later (IS FTAM) and is registered in the pilot OSI Directory below

  `bells, Computer Science, University College London, GB`

For distributions via either FTAM or FTP, there are two additional files available for retrieval:

- `isode-7-doc.tar.Z`
  This is the LaTeXsource for the entire documentation set. It is a compressed *tar* image (3.5MB).

- `isode-7-ps.tar.Z`
  This contains the five volume manual in PostScript format. It is a compressed *tar* image (4.2MB).

Discussion Groups

The Internet discussion group `ISODE@NIC.DDN.MIL` is used as a forum to discuss ISODE. Contact the Internet mailbox `ISODE-Request@NIC.DDN.MIL` to be asked to be added to this list.

Support

Although the ISODE is not "supported" per se, it does have a problem reporting address, `Bug-ISODE@XTEL.CO.UK`. Bug reports (and fixes) are welcome by the way.

Commercial Support

A UK company has been set up to provide commercial support for the ISODE and associated packages — X-Tel Services Ltd. This company provides an update service, general assistance and site specific support. Although inclusion of this information should not be considered an endorsement, it should be noted that two of the primary ISODE developers now work at X-Tel Services Ltd.

Postal address:	X-Tel Services Ltd. University Park, Nottingham, NG7 2RD UK
Telephone	+44 602 412648
Fax	+44 602 790278
Internet	support@xtel.co.uk

If other organisations offering formal support for the ISODE wish to be included in future announcements, a suitable index will be organized.

Acknowledgements

Many people have made comments about and contributions to the ISODE which have been most helpful. The following list is by no means complete:

The first three releases of the ISODE were developed at the Northrop Research and Technology Center, and the first version of this manual is referenced as NRTC Technical Paper #702. The initial work was supported in part by Northrop's Independent Research and Development program.

The Wollongong Group supported ISODE for its 4.0 and 5.0 release. They deserve much credit for that. Furthermore, they contributed an implementation of RFC1085, a lightweight presentation protocol for TCP/IP-based internets.

The ISODE development up until version 7.0 was supported by Performance Systems International, Inc. and NYSERNet, Inc. This was a substantial commitment to which the project in indebted. Before their support, ISODE was always an after-hours activity. The NYSERNet effort was partially supported by the U.S. Defense Advanced Research Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602–88–C–0016 to NYSERNet Inc.

Christopher W. Moore of the Wollongong Group has provided much help with ISODE in terms of both policy and implementational matters. He also performed Directory interoperability testing against a different implementation of the OSI Directory.

Dwight E. Cass of the Northrop Research and Technology Center was one of the original architects of *The ISO Development Environment*. His work was critical for the original proof of concept and should not be forgotten. John L. Romine also of the Northrop Research and Technology Center provided many fine comments concerning the presentation of the material herein. This resulted in a much more readable manuscript. Stephen H. Willson, also of the Northrop Research and Technology Center, provided some help in verifying the operation of the software on a system running the AT&T variant of UNIX[1].

The *librosap*(3n) library was heavily influenced by an earlier native-TCP version written by George Michaelson, formerly of University College London, in the United Kingdom. Stephen E. Kille, of University College London,

[1] UNIX is a trademark of AT&T Bell Laboratories.

provided valuable feedback on the *pepy*(1) utility. In addition, both Steve and George provided us with some good comments concerning the *libpsap*(3) library. Steve is also the conceptual architect for the addressing scheme used in the software, and he modified the *librosap*(3n) library to support half-duplex mode when providing ECMA ROS service. George contributed the CAMTEC and ULTRIX[2] X.25 interfaces. Simon Walton, also of University College London, has been very helpful in providing constant feedback on the ISODE during beta-testing.

The INCA project donated the QUIPU Directory implementation to the ISODE. Stephen E. Kille, Colin J. Robbins, and Alan Turland, at the time all of University College London, are the three principals who developed the 6.0 version of the directory software. In addition, Steve Titcombe, also of UCL spent considerable time on the DIrectory SHell (DISH), and Mike Roe formerly of UCL, put a large amount of effort into the security requirements of QUIPU. Development of the current version of QUIPU has been coordinated by Colin J. Robbins now of X-Tel Services Ltd, and designed by Stephen E. Kille.

The UCL work has been partially supported by the commission of the EEC under its ESPRIT program, as a stage in the promotion of OSI standards. Their support has been vital to the UCL activity. In addition, QUIPU is also funded by the UK Joint Network Team (JNT).

Julian P. Onions, of X-Tel Services Ltd is the current *pepy*(1) guru, having brainstormed and implemented the encoding functionality along with Stephen E. Easterbrook formerly of University College London. Julian also contributed the UBC X.25 interface along with the TCP/X.25 TP0 bridge, and has also contributed greatly to *posy*(1). Julian's latest contribution has been a *transport service bridge*. This is used to masterfully solve interworking problems between different OSI stacks (TP0/X.25, TP4/CLNP, RFC1006/TCP, and so on).

John Pavel and Godfrey Cowin of the Department of Trade and Industry/ National Physical Laboratory in the United Kingdom both contributed significant comments during beta-testing. In particular, John gave us a lot of feedback on *pepy*(1) and on the early FTAM DIS implementation. John also contributed the SunLink X.25 interface.

Keith Ruttle of CAMTEC Electronics Limited in the United Kingdom

[2] ULTRIX is a trademark of Digital Equipment Corporation.

contributed both the driver for the new CAMTEC X.25 interface and the CAMTEC CONS interface (X.25 over 802 networks). This latter driver was later removed from the distribution for lack of use.

In addition, Andrew Worsley of the Department of Computer Science at the University of Melbourne in Australia pointed out several problems with the FTAM DIS implementation. He also developed a replacement for *pepy* and *posy* called *pepsy*. After moving to University College London, he improved this system and integrated into the ISODE.

Olivier Dubous of BIM sa in Belgium contributed some fixes to concurrency control in the FTAM initiator to allow better interworking with the VMS[3] implementation of the filestore. He also suggested some changes to allow interworking with the FTAM T1 and A/111 profiles.

Olli Finni of Nokia Telecommunications provided several fixes found when testing interoperability with the TOSI implementation of FTAM.

Mark R. Horton of AT&T Bell Laboratories also provided some help in verifying the operation of the software on a 3B2 system running UNIX System V release 2. In addition, Greg Lavender of NetWorks One, under contract to the U.S. Navy Regional Data Automation Center (NARDAC), provided modifications to allow the software to run on a generic port of UNIX System V release 3.

Steve D. Miller of the University of Maryland provided several fixes to make the software run better on the ULTRIX variant of UNIX.

Jem Taylor of the Computer Science Department at the University of Glasgow provided some comments on the documentation.

Hans-Werner Braun of the University of Michigan provided the inspiration for the initial part of Section 1.2.

A previous release of the software contained an ISO TP4/CLNP package derived from a public-domain implementation developed by the National Institute of Standards and Technology (then called the National Bureau of Standards). The purpose of including the NIST package (and associated support) was to give an example of how one would interface the code to a "generic" TP4 implementation. As the software has now been interfaced to various native TP4 implementations, the NIST package is no longer present in the distribution.

John A. Scott of the MITRE Corporation contributed the SunLink OSI

[3]VMS is a trademark of Digital Equipment Corporation.

interface for TP4. He also wrote the FTAM/FTP gateway which the MITRE Corporation has generously donated to this package.

Philip B. Jelfs of the Wollongong Group upgraded the FTAM/FTP gateway to the "IS-level" (International Standard) FTAM.

Rick Wilder and Don Chirieleison of the MITRE Corporation contributed the VT implementation which the MITRE Corporation has generously donated to this package.

Jacob Rekhter of the T. J. Watson Research Center, IBM Corporation, provided some suggestions as to how the system should be ported to the IBM RT/PC running either AIX or 4.3BSD. He also fixed the incompatibilities of the FTAM/FTP gateway when running on 4.3BSD systems.

Ashar Aziz and Peter Vanderbilt, both of of Sun Microsystems Inc., provided some very useful information on modifying the SunLink OSI interface for TP4.

Later on, elements of the SunNet 7.0 Development Team (Hemma Prafullchandra, Raj Srinivasan, Daniel Weller, and Erik Nordmark) made numerous enhancements and fixes to the system.

John Brezak of Apollo Computer, Incorporated ported the ISODE to the Apollo workstation. Don Preuss, also of Apollo, contributed several enhancements and minor fixes.

Ole-Jorgen Jacobsen of Advanced Computing Environments provided some suggestions on the presentation of the material herein.

Nandor Horvath of the Computer and Automation Institute of the Hungarian Academy of Sciences while a guest researcher at the DFN/GMD in Darmstadt, FRG, provided several fixes to the FTAM implementation and documentation.

George Pavlou and Graham Knight of University College London contributed some management instrumentation to the *libtsap*(3n) library.

Juha Heinänen of Tampere University of Technology provided many valuable comments and fixes on the ISODE.

Paul Keogh of the Nixdorf Research and Development Center, in Dublin, Ireland, provided some fixes to the FTAM implementation.

Oliver Wenzel of GMD Berlin contributed the RFA system.

L. McLoughlin of Imperial College contributed Kerberos support for the FTAM responder.

Kevin E. Jordan of CDC provided many enhancements to the G3FAX library.

John A. Reinart of Cray Research contributed many performance enhancements.

Ed Pring of the T. J. Watson Research Center, IBM Corporation, provided several fixes to the SMUX implementation in ISODE's SNMP agent.

Uli Betzler of University of Karlsruhe donated the HP X.25 interface and other HP-UX modifications.

Alan Young of Concurrent Computer Corporation contributed many helpful changes to the transport code.

James Gosling, author of the superb Emacs screen editor for UNIX, and Leslie Lamport, author of the excellent LaTeX document preparation system both deserve much praise for such winning software. Of course, the whole crew at U.C. Berkeley also deserves tremendous praise for their wonderful work on their variant of UNIX.

Finally, full credit for this release must be given to Marshall T. Rose. Marshall has for so long master-minded the whole ISODE effort. Although he has not been involved in the last month or so leading upto this release, the vast majority of the effort getting ISODE-7.0 into shape came from Marshall.

jpo & cjr

Nottingham, England
February, 1994

Part I

Introduction

Chapter 1

Overview

This document describes a non-proprietary implementation of some of the protocols defined by the International Organisation for Standardisation and International Electrotechnical Commission (ISO/IEC), the International Telegraph and Telephone Consultative Committee (CCITT), and the European Computer Manufacturer Association (ECMA).[1]

The purpose of making this software openly available is to accelerate the process of the development of applications in the OSI protocol suite. Experience indicates that the development of application level protocols takes as long as or significantly longer than the lower level protocols. By producing a non-proprietary implementation of the OSI protocol stack, it is hoped that researchers in the academic, public, and commercial arenas may begin working on applications immediately. Another motivation for this work is to foster the development of OSI protocols both in the European RARE and the U.S. Internet communities. The Internet community is widely known as having pioneered computer-communications since the early 1970's. This community is rich in knowledge in the field, but currently is not actively experimenting with the OSI protocols. By producing an openly available implementation, it is hoped that the OSI protocols will become quickly widespread in the Internet, and that a productive (and *painless*) transition in the defense data network (DDN) might be promoted. The RARE community is the set of corresponding European academic and research organisations. While they do not have the same long implementation experience as the Internet

[1] In the interests of brevity, unless otherwise noted, the term "OSI" is used to denote these parallel protocol suites.

community, they have a deep commitment to international standards. It is intended that this release gives vital early access to prototype facilities.

1.1 Fanatics Need Not Read Further

This software can support several different network services below the transport service access point (TSAP). One of these network services is the DoD transmission control protocol (TCP)[Pos81].[2] This permits the development of the higher level protocols in a robust and mature internet environment, while providing us the luxury of not having to recode anything when moving to a network where the OSI transport protocol (TP) is used to provide the TSAP. However, the software also operates over pure OSI lower levels of software. It is mainly used in that fashion — outside of the United States.

Of course, there will always be "zealots of the pure faith" making claims to the effect that:

> *TCP/IP is dead! Any work involving TCP/IP simply dilutes the momentum of OSI.*

or, from the opposite end of the spectrum, that

> *The OSI protocols will never work!*

Both of these statements, from diametrically opposing protocol camps are, of course, completely unfounded and largely inflammatory. TCP/IP is here, works well, and enjoys a tremendous base of support. OSI is coming, and will work well, and when it eventually comes of age, it will enjoy an even larger base of support.

The role of ISODE, in this maelstrom that generates much heat and little light, is to provide a useful transition path between the two protocol suites in which complementary efforts can occur. The ISODE approach is to use the strengths of both the DDN and OSI protocol suites in a cooperative and positive manner. For a more detailed exposition of these ideas, kindly refer to [Ros90] or the earlier work [RC86].

[2] Although the TCP corresponds most closely to offering a transport service in the OSI model, the TCP is used as a connection-oriented network protocol (i.e., as coservice to X.25).

1.2 The Name of the Game

The name of the software is the ISODE. The official pronunciation of the ISODE, takes four syllables: *I-SO-D-E*. This choice is mandated by fiat, not by usage, in order to avoid undue confusion.

Please, as a courtesy, do not spell ISODE any other way. For example, terms such as ISO/DE or ISO-DE do not refer to the software! Similarly, do not try to spell out ISODE in such a way as to imply an affiliation with the International Organisation for Standardization. There is no such relationship. The *ISO* in ISODE is not an acronym for this organisation. In fact, the *ISO* in ISODE doesn't really meaning anything at all. It just has a catchy two syllable sound.

1.3 Operating Environments

This release is coded entirely in the *C* programming language[KR78], and is known to run under the following operating systems (without any kernel modifications):

Berkeley UNIX The software should run on any faithful port of 4.2BSD, 4.3BSD, or 4.4BSD UNIX. Sites have reported the software running: on the Sun-3 workstation running Sun UNIX 4.2 release 3.2 and later; on the Sun Microsystems workstation (Sun-3, Sun-4, and Sun-386i) running SunOS release 4.0 and later; on Digital Equipment Corporation ULTRIX machines (including X.25); the Integrated Solutions workstation; and on the RT/PC running 4.3BSD.[3] In addition to using the native TCP facilities of Berkeley UNIX, the software has also be interfaced to versions 4.0 through 7.0 of the SunLink/SunNet X.25 and OSI packages (although Sun may have to supply you with some modified `sgtty` and `ioctl` include files if you are using an earlier version of SunLink X.25). The optional SunLink Communications Processor running

[3] Do not however, attempt to compile the software with the SunPro *make* program! It is not, contrary to any claims, compatible with the standard *make* facility. Furthermore, note that if you are running a version of SunOS 4.0 prior to release 4.0.3, then you may need to use the *make* program found in */usr/old/*, if the standard *make* you are using is the SunPro *make*. In this case, you will need to put the old, standard *make* in */usr/bin/*, and you can keep the SunPro *make* in */bin/*.

DCP 3.0 software has also been tested with the software.

AT&T UNIX The software should run on any faithful port of System V UNIX. The SVR4 port used an ICL DRS6000 as a testbed. One of the other systems tested was running with an Excelan EXOS[4] 8044 TCP/IP card. The Excelan package implements the networking semantics of the 4.1aBSD UNIX kernel. As a consequence, the software should run on any faithful port of 4.1aBSD UNIX, with only a minor amount of difficultly. As of this writing, however, this speculation has not been verified. The particular system used was a Silicon Graphics IRIS workstation.[5]

Another system was running the WIN TCP/IP networking package. The WIN package implements the networking semantics of the 4.2BSD UNIX kernel. The particular system used was a 3B2 running System V release 2.0.4, with WIN/3B2 version 1.0.

Another system was also running the WIN TCP/IP networking package but under System V release 3.0. The WIN package on SVR3 systems emulates the networking semantics of the 4.2BSD UNIX kernel but uses STREAMS and TLI to do so.

Note: The System V port does not include VT or SNMP.

AIX The software should run on the IBM AIX Operating System which is a UNIX-based derivative of AT&T's System V. The particular system used was a RT/PC system running version 2.1.2 of AIX.

HP-UX The software should run on HP's UNIX-like operating system, HP-UX. The particular system used was an Indigo 9000/840 system running version A.B1.01 of HP-UX. The system was also reported to have run on an HP 9000/350 system under version 6.2 of HP-UX. Support for HP's X.25 has been included.

ROS The software should run on the Ridge Operating system, ROS. The particular system used was a Ridge-32 running version 3.4.1 of ROS.

[4] EXOS is a trademark of Excelan, Incorporated.

[5] This test was made with an earlier release of this software, and access to an SGI workstation was not available when the current version of the software tested. However, the networking interface is still believed to be correct for the Excelan package.

Pyramid OsX The software should run on a Pyramid computer running OsX. The particular system used was a Pyramid 98xe running version 4.0 of OsX.

Since a Berkeley UNIX system is the primary development platform for ISODE, this documentation is somewhat slanted toward that environment.

1.4 Organisation of the Release

A strict layering approach has been taken in the organisation of the release. The documentation mimics this relationship approximately: the first two volumes describe, in top-down fashion, the services available at each layer along with the databases used by those services; the third volume describes some applications built using these facilities; the fourth volume describes a facility for building applications based on a programming language, rather that network-based, model; and, the fifth volume describes a complete implementation of the OSI directory.

In *Volume One*, the "raw" facilities available to applications are described, namely four libraries:

- The *libacsap*(3n) library, which implements the OSI association control service (ACS).
- The *librosap*(3n) library, which implements different styles of the OSI remote operations service (ROS).
- The *librtsap*(3n) library, which implements the OSI reliable transfer service (RTS). and,
- The *libpsap*(3) library, which implements the OSI abstract syntax and transfer mechanisms.

In *Volume Two*, the services upon which the application facilities are built are described, namely three libraries:

- The *libpsap2*(3n) library, which implements the OSI presentation service.
- The *libssap*(3n) library, which implements the OSI session service. and,

- The *libtsap*(3n) library, which implements an OSI transport service access point.

In addition, there is a replacement for the *libpsap2*(3n) library called the *libpsap2-lpp*(3n) library. This implements the lightweight presentation protocol for TCP/IP-based internets as specified in RFC-1085.

In addition, *Volume Two* contains information on how to configure the ISODE for your network.

In *Volume Three*, some application programs written using this release are described, including:

- An implementation of the ISO FTAM which runs on Berkeley or AT&T UNIX. FTAM (file transfer, access and management), is the OSI file service. The implementation provided is fairly complete in the context of the particular file services it offers. It is a minimal implementation inasmuch as it offers only four core services: transfer of text files, transfer of binary files, directory listings, and file management.

- An implementation of an FTAM/FTP gateway, which runs on Berkeley UNIX.

- An implementation of the ISO VT which runs on Berkeley UNIX. VT (Virtual Terminal) is the OSI terminal service. The implementation consists of a basic class, TELNET profile implementation.

- An implementation of the "little services" often used for debugging and amusement.

- An implementation of a simple image database service.

In *Volume Four*, a "cooked" interface for applications using remote operations is described, which consists of three programs and a library:

- The *rosy*(1) compiler, which is a stub-generator for specifications of remote operations.

- The *posy*(1) compiler, which is a structure-generator for ASN.1 specifications.

- The *pepy*(1) compiler, which reads a specification for an application and produces a program fragment that builds or recognises the data structures (APDUs in OSI argot) which are communicated by that application.
- The new *pepsy*(1) compiler which will replace *pepy* and *posy* programs in a future release of the ISODE.
- The *librosy*(3n) library, which is a library for applications using this distributed applications paradigm.

In *Volume Five*, the QUIPU directory is described, which currently consists of several programs and a library:

- The *quipu*(8c) program, which is a directory system agent (DSA);
- The *dish*(1c) family of programs, which are a set of directory shell commands; and,
- The *libdsap*(3n) library, which is a library for applications using the directory.

1.5 A Note on This Implementation

Although the implementation described herein might form the basis for a production environment in the near future, this release is not represented as "production software".

However, throughout the development of the software, every effort has been made to employ good software practices which result in efficient code. In particular, the current implementation avoids excessive copying of bytes as data moves between layers. Some rough initial timings of echo and sink entities at the transport and session layers indicate data transfer rates quite competitive with a raw TCP socket (most differences were lost in the noise). The work involved to achieve this efficiency was not demanding.

Additional work was required so that programs utilising the *libpsap*(3) library could enjoy this level of performance. Although data transfer rates at the reliable transfer and remote operations layers are not as good as raw TCP, they are still quite impressive (on the average, the use of a ROS interface (over presentation, session, and ultimately the TCP) is only 20% slower than a raw TCP interface).

1.6 Changes Since the Last Release

A brief summary of the major changes between v. 6.0 and v. 7.0 are now presented. These are the user-visible changes only; changes of a strictly internal nature are not discussed.

- A new program, *pepsy*, has been developed to replace both *pepy* and *posy*. It is described in *Volume Four*.
- The *dsabuild* program has been removed, in favor of some shell scripts.
- The "higher performance nameservice" has been discontinued in favor of a "user-friendly nameservice". As such, the syntax of the `str2aei` routine has changed. This routine will soon be deprecated, so get in the habit of using the new `str2aeinfo` routine discussed in the ISODE Manual.
- The `na_type` and `na_subnet` fields of the network address structure described in the ISODE Manual have been renamed. For compatibility, macros are provided. These macros will be removed after this release.
- The stub directory facility is now deprecated in favor of an OSI directory based approach. As a result, the *aetbuild* program has been removed.
- An interworking problem in the presentation layer has been fixed in a backwards compatible manner. This should not affect interoperability in this release. The backwards compatibility will be removed after this release.

As a rule, the upgrade procedure is a two-step process: first, attempt to compile your code, keeping in mind the changes summary relevant to the code; and, second, once the code successfully compiles, run the code through *lint*(1) with the supplied lint libraries.

Although every attempt has been made to avoid making changes which would affect previously coded applications, in some cases incompatible changes were required in order to achieve a better overall structure.

Part II

File Transfer, Access and Management

Chapter 2

User Library

The *libftam*(3n) library implements the filestore-independent parts of the International Standard of the OSI file service, FTAM (which stands for File Transfer, Access and Management). Currently supported are: the no-recovery FTAM Quality-of-Service; the transfer, access, management, and transfer and management service classes; the kernel, read, write, access, limited file management, enhanced file management, grouping, and fadu-locking functional units; and, the kernel, storage, security, and private attribute groups.

Unlike most OSI services, FTAM distinguishes between the *initiator* of an FTAM association and the *responder*. However, as with most models of OSI services, an asynchronous environment is assumed. That is, the service provider may generate events for the service user without the latter triggering the actions which led to the event. For example, in a synchronous environment, an indication that data has arrived usually occurs only when the service user asks the service provider to read data; in an asynchronous environment, the service provider may interrupt the service user at any time to announce the arrival of data.

The `ftam` module in this release initially uses a synchronous interface; however once the connection is established, an asynchronous interface may be selected.

All of the routines in the *libftam*(3n) library are integer-valued. They return the manifest constant `OK` on success, or `NOTOK` otherwise.

2.1 Warning

Please read the following important message.

NOTE:	Readers of this chapter should have an intimate understanding of FTAM. It is not the intent of this chapter to present a tutorial on these services, so novice users will suffer greatly if they choose to read further. As previous versions of this software included an implementation of DIS FTAM, and the release contains an IS implementation, users of the *libftam*(3n) library are urged to re-read this chapter.

2.2 Constants

There are several important constants, described below:

2.2.1 FTAM Quality-of-Service

Value	FTAM-QoS
`FQOS_NORECOVERY`	no recovery
`FQOS_CLASS1`	class 1 recovery
`FQOS_CLASS2`	class 2 recovery
`FQOS_CLASS3`	class 3 recovery

Currently, only the no-recovery FTAM-QoS is supported.

2.2.2 Service Classes

Value	Service Class
`FCLASS_TRANSFER`	transfer
`FCLASS_ACCESS`	access
`FCLASS_MANAGE`	management
`FCLASS_TM`	transfer and management
`FCLASS_UNCONS`	unconstrained

Currently, all service classes other than the unconstrained class are supported.

2.2.3 Functional Units

Value	Functional Unit
FUNIT_READ	read
FUNIT_WRITE	write
FUNIT_ACCESS	file access
FUNIT_LIMITED	limited file management
FUNIT_ENHANCED	enhanced file management
FUNIT_GROUPING	grouping
FUNIT_FADULOCK	fadu locking
FUNIT_RECOVERY	recovery
FUNIT_RESTART	restart data transfer

Currently, all functional units other than the recovery and restart units are supported. Further, the grouping unit *must* be specified.

2.2.4 Attribute Groups

Value	Attribute Group
FATTR_STORAGE	storage
FATTR_SECURITY	security
FATTR_PRIVATE	private

Currently, all attribute groups other than the private group are supported.

2.2.5 State Results

Value	State Result
FSTATE_SUCCESS	success
FSTATE_FAILURE	failure

2.2.6 Action Results

Value	Action Result
FACTION_SUCCESS	success
FACTION_TRANS	transient error
FACTION_PERM	permanent error

2.3 Data-Structures

There are several important data structures, described below:

2.3.1 Contents Type

An FTAM document type is represented by the `FTAMcontent` structure.

```
struct FTAMcontent {
    OID     fc_dtn;

    int     fc_id;
    int     fc_result;
};
```

This elements of this structure are:

`fc_dtn`: the document type name, represented as an object identifier (consult Section 5.4.6 of *Volume One*);

`fc_id`: the presentation context identifier (consult Section 2.3.1 of *Volume Two*) associated with the document type; and,

`fc_result`: the status indicator for the presentation context (codes are listed in Table 2.1 of *Volume Two*).

A list of FTAM documents types is represented by the `FTAMcontentlist` structure.

```
struct FTAMcontentlist {
    int     fc_ncontent;

#define NFCONT  (NPCTX - 2)
    struct FTAMcontent fc_contents[NFCONT];
};
```

This elements of this structure are:

`fc_contents/fc_ncontent`: the contents list (and the number of contents in the list).

A limitation of this implementation is that a fixed number of document types may be supported, as denoted by the manifest constant `NFCONT`.

2.3.2 Diagnostics

An FTAM diagnostic is represented by the `FTAMdiagnostic` structure.

```
struct FTAMdiagnostic {
    int     ftd_type;

    int     ftd_identifier;

    int     ftd_observer;
    int     ftd_source;

    int     ftd_delay;

#define FTD_SIZE        512
    int     ftd_cc;
    char    ftd_data[FTD_SIZE];
};
```

This elements of this structure are:

`ftd_type`: the diagnostic type, one of:

Value	Meaning
`DIAG_INFORM`	informative
`DIAG_TRANS`	transient
`DIAG_PERM`	permanent

`ftd_identifier`: the error-identifier (consult `<isode/ftam.h>` for the list, there are far too many to list here);

`ftd_observer/ftd_source`: the observer and source of the error, one of:

Value	Meaning
`EREF_NONE`	no categorization possible
`EREF_IFSU`	initiating file service user
`EREF_IFPM`	initiating file service machine
`EREF_SERV`	service support the file protocol machine
`EREF_RFPM`	responding file protocol machine
`EREF_RFSU`	responding file service user

ftd_delay: the suggested delay that might be honored prior to retrying the operation (use DIAG_NODELAY if a delay is not appropriate); and,

ftd_data/ftd_cc: a diagnostic string (and the length of that string).

A limitation of this implementation is that the routines and data structures which support diagnostics have a fixed limit as to the number of diagnostics permitted. This is currently denoted by the manifest constant NFDIAG.

2.3.3 Charging

A list of FTAM charges is represented by the FTAMcharging structure.

```
struct FTAMcharging {
    int     fc_ncharge;

#define NFCHRG  5
    struct fc_charge {
        char   *fc_resource;
        char   *fc_unit;
        int     fc_value;
    }           fc_charges[NFCHRG];
};
```

This elements of this structure are:

fc_charges/fc_ncharge: the charging list (and the number of charges in the list, which may not exceed the manifest constant NFCHRG). Each charge consists of:

fc_resource: the resource identifier;

fc_unit: the charging unit; and,

fc_value: the charging value.

A limitation of this implementation is that a fixed number of charges may be supported, as denoted by the manifest constant NFCHRG.

2.3.4 Passwords

A list of FTAM passwords is represented by the `FTAMpasswords` structure.

```
struct FTAMpasswords {
    char   *fp_read;
    int     fp_readlen;

    char   *fp_insert;
    int     fp_insertlen;

    char   *fp_replace;
    int     fp_replacelen;

    char   *fp_extend;
    int     fp_extendlen;

    char   *fp_erase;
    int     fp_eraselen;

    char   *fp_readattr;
    int     fp_readattrlen;

    char   *fp_chngattr;
    int     fp_chngattrlen;

    char   *fp_delete;
    int     fp_deletelen;
};
```

This elements of this structure are:

`fp_read/fp_readlen`: the read password (and the length of the password);

`fp_insert/fp_insertlen`: the insert password (and the length of the password);

`fp_replace/fp_replacelen`: the replace password (and the length of the password);

`fp_extend/fp_extendlen`: the extend password (and the length of the password);

`fp_erase/fp_eraselen`: the erase password (and the length of the password);

`fp_readattr/fp_readattrlen`: the read attribute password (and the length of the password);

`fp_chngattr/fp_chngattrlen`: the change attribute password (and the length of the password); and,

`fp_delete/fp_deletelen`: the delete password (and the length of the password).

The `FPFREE` macro can be used to free the storage associated with an `FTAMpasswords` structure (without freeing the structure itself). It behaves as if it was defined as:

```
void    FPFREE (fp)
struct FTAMpasswords *fp;
```

2.3.5 Access Control

An FTAM access control element is represented by the `FTAMacelement` structure.

```
struct FTAMacelement {
    int     fe_actions;

    struct FTAMconcurrency fe_concurrency;

    char   *fe_aet;

    struct FTAMpasswords fe_passwords;

    AEI    fe_aet;

    struct FTAMacelement *fe_next;
};
```

Requested Access	
`FA_PERM_READ`	read fadu
`FA_PERM_INSERT`	insert fadu
`FA_PERM_REPLACE`	replace fadu
`FA_PERM_EXTEND`	extend fadu
`FA_PERM_ERASE`	erase fadu
`FA_PERM_READATTR`	read attribute
`FA_PERM_CHNGATTR`	change attribute
`FA_PERM_DELETE`	delete file

Table 2.1: FTAM Requested Access Values

The elements of this structure are:

`fe_actions`: the action list, containing the inclusive-or of any of the values shown in Table 2.1;

`fe_concurrency`: any concurrency-control constraints;

`fe_aet`: the user identity;

`fe_passwords` the passwords;

`fe_aet`: the application-entity title (defined in Section 2.2.1 on page 17 in *Volume One*); and,

`fe_next`: the next access element in the linked list.

The `FEFREE` macro can be used to free the storage associated with a list of FTAM access elements, including the head element on the list. It behaves as if it was defined as:

```
void    FEFREE (fe)
struct FTAMacelement *fe;
```

2.3.6 Attributes

An FTAM attribute list is represented by the `FTAMattributes` structure.

```
struct FTAMattributes {
    long    fa_present;
    long    fa_novalue;

#define NFFILE  5
    int     fa_nfile;
    char   *fa_files[NFFILE];

    int     fa_permitted;

    OID     fa_contents;
    PE      fa_parameter;

    char   *fa_account;

    struct UTCtime fa_date_create;
    struct UTCtime fa_date_modify;
    struct UTCtime fa_date_read;
    struct UTCtime fa_date_attribute;

    char   *fa_id_create;
    char   *fa_id_modify;
    char   *fa_id_read;
    char   *fa_id_attribute;

    int     fa_availability;

    int     fa_filesize;
    int     fa_futuresize;

    struct FTAMacelement fa_control;
    char   *fa_legal;
};
```

The elements of this structure are:

fa_present: the values present, containing the inclusive-or of any of the values shown in Table 2.2;

fa_novalue: the values which are not available are not available (using the same values listed in Table 2.2, the value of this element is a subset of the value of the `fa_present` element);

fa_files/fa_nfile: the list of filename components (and the number of components, which may not exceed the manifest constant `NFFILE`);

fa_permitted: permitted actions, containing the inclusive-or of any of the values shown in Table 2.1 on page 23, in addition to any of:

Value	Meaning
`FA_PERM_TRAV`	traversal
`FA_PERM_RVTRAV`	reverse-traversal
`FA_PERM_RANDOM`	random-order

fa_contents/fa_parameter: the contents type;

fa_account: the account;

fa_date_create: the date and time of creation;

fa_date_modify: the date and time of last modification;

fa_date_read: the date and time of last read access;

fa_date_attribute: the date and time of last attribute modification;

fa_id_create: the identity of creator;

fa_id_modify: the identity of last modifier;

fa_id_read: the identity of last reader;

fa_id_attribute: the identity of last attribute modifier;

`fa_availability`: file availability, one of:

Value	Meaning
FA_AVAIL_IMMED	immediate
FA_AVAIL_DEFER	deferred

`fa_filesize`: filesize;

`fa_futuresize`: future filesize;

`fa_control`: access control;

`fa_encrypt`: encryption name; and,

`fa_legal`: legal qualification.

A limitation of this implementation is that a fixed number of filename components may be supported, as denoted by the manifest constant `NFFILE`.

The routine `FAFREE` can be used to free the storage associated with an `FTAMattributes` structure (without freeing the structure itself). It is defined as:

```
void    FAFREE (fa)
struct FTAMattributes *fa;
```

2.3.7 Concurrency

An FTAM concurrency list is represented by the `FTAMconcurrency` structure.

```
struct FTAMconcurrency {
#define FLOCK_SHARED    00      /* shared */
#define FLOCK_EXCLUSIVE 01      /* exclusive */
#define FLOCK_NOTREQD   02      /* not-required */
#define FLOCK_NOACCESS  03      /* no-access */

    char    fc_readlock;
    char    fc_insertlock;
    char    fc_replacelock;
    char    fc_extendlock;
```

	Attribute
`FA_FILENAME`	filename
`FA_ACTIONS`	permitted actions
`FA_CONTENTS`	contents type
`FA_ACCOUNT`	account
`FA_DATE_CREATE`	date and time of creation
`FA_DATE_MODIFY`	date and time of last modification
`FA_DATE_READ`	date and time of last read access
`FA_DATE_ATTR`	date and time of last attribute modification
`FA_ID_CREATE`	identity of creator
`FA_ID_MODIFY`	identity of last modifier
`FA_ID_READ`	identity of last reader
`FA_ID_ATTR`	identity of last attribute modifier
`FA_AVAILABILITY`	file availability
`FA_FILESIZE`	filesize
`FA_FUTURESIZE`	future filesize
`FA_CONTROL`	access control
`FA_LEGAL`	legal qualifications
`FA_PRIVATE`	private use

Table 2.2: FTAM Attribute Values

```
        char    fc_eraselock;
        char    fc_readattrlock;
        char    fc_chngattrlock;
        char    fc_deletelock;
    };
```

This elements of this structure are:

fc_readlock: the concurrency constraints for the read operation;

fc_insertlock: the concurrency constraints for the insert operation;

fc_replacelock: the concurrency constraints for the replace operation;

fc_extendlock: the concurrency constraints for the extend operation;

fc_eraselock: the concurrency constraints for the erase operation;

fc_readattrlock: the concurrency constraints for the read attribute operation;

fc_chngattrlock: the concurrency constraints for the change attribute operation; and,

fc_deletelock: the concurrency constraints for the delete operation.

The FCINIT macro can be used to initialize an FTAMconcurrency structure. It behaves as if it was defined as:

```
void    FCINIT (fc)
struct FTAMconcurrency *fc;
```

2.3.8 FADU Identity

An FADU identity is represented by the FADUidentity structure.[1]

[1] FADU stands for File Access Data Unit; if you didn't already know this, you shouldn't be reading this chapter. Read Section 2.1 right now.

```
struct FADUidentity {
    int     fa_type;
#define FA_FIRSTLAST    0       /* first-last */
#define FA_RELATIVE     1       /* relative */
#define FA_BEGINEND     2       /* begin-end */
#define FA_SINGLE       3       /* single-name */
#define FA_NAMELIST     4       /* name-list */
#define FA_FADUNUMBER   5       /* fadu-number */

    union {
        int     fa_un_firstlast;
#define FA_FIRST        0
#define FA_LAST         1

        int     fa_un_relative;
#define FA_PREVIOUS     0
#define FA_CURRENT      1
#define FA_NEXT         2

        int     fa_un_beginend;
#define FA_BEGIN        0
#define FA_END          1

        char   *fa_un_singlename;

#define NANAME  5
        struct {
            char   *fa_un_names[NANAME];
            int     fa_un_nname;
        }       fa_un_list;

        int     fa_un_fadunumber;
    }   fa_un;
#define fa_firstlast    fa_un.fa_un_firstlast
#define fa_relative     fa_un.fa_un_relative
#define fa_beginend     fa_un.fa_un_beginend
#define fa_singlename   fa_un.fa_un_singlename
#define fa_names        fa_un.fa_un_list.fa_un_names
#define fa_nname        fa_un.fa_un_list.fa_un_nname
```

```
#define fa_fadunumber   fa_un.fa_un_fadunumber
};
```

As shown, this structure is really a discriminated union (a structure with a tag element followed by a union). The elements are:

fa_type: indicates what type of FADU identity is represented; Depending on this type, one of the following elements is valid:

- **fa_firstlast**: the location, identified as first or last;
- **fa_relative**: the location, identified as previous, current, or next;
- **fa_beginend**: the location, identified as begin or end;
- **fa_singlename**: the location, as identified by name;
- **fa_names/fa_nname**: the location, as identified by a list of names (and the length of that list, which may not exceed the manifest constant `NANAME`); or,
- **fa_fadunumber**: the location, as identified by number.

A limitation of this implementation is that a fixed number of FADU names may be supported, as denoted by the manifest constant `NANAME`.

The `FUFREE` macro can be used to free the storage associated with an `FADUidentity` structure (without freeing the structure itself). It behaves as if it was defined as:

```
void    FUFREE (fu)
struct FADUidentity *fu;
```

2.4 Association Establishment

The *libftam*(3n) library distinguishes between the entity which started an association, the *initiator*, and the entity which was subsequently bound to the association, the *responder*. We sometimes term these two entities the *client* and the *server*, respectively.

Section 2.2.1 of *Volume One* describes the use of addressing for the application layer. In particular, Figure 2.1 on page 21 of *Volume One* presents an example of how one constructs the address for the File Transfer, Access and Management (FTAM) service on host `RemoteHost`.

2.4.1 Responder

The *tsapd*(8c) daemon, upon accepting a connection from an initiating host, consults the ISO services database to determine which program on the local system implements the desired application context.

Once the program has been ascertained, the daemon runs the program with any arguments listed in the database. In addition, it appends some *magic arguments* to the argument vector. Hence, the very first action performed by the responder is to re-capture the FTAM state contained in the magic arguments. This is done by calling the routine `FInit`, which on a successful return, is equivalent to an F-INITIALIZE.INDICATION event.

```
int     FInit (vecp, vec, fts, tracing, fti)
int     vecp;
char  **vec;
struct FTAMstart *fts;
int   (*tracing) ();
struct FTAMindication *fti;
```

The parameters to this procedure are:

`vecp`: the length of the argument vector;

`vec`: the argument vector;

`fts`: a pointer to an `FTAMstart` structure, which is updated only if the call succeeds;

`tracing`: the address of a tracing routine to be invoked when an FTAM event occurs (consult Section 2.5.3); and,

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

If `FInit` is successful, it returns information in the `fts` parameter, which is a pointer to an `FTAMstart` structure.

```
struct FTAMstart {
    int     fts_sd;

    AEInfo  fts_calledtitle;
```

```
        AEInfo  fts_callingtitle;

        struct PSAPaddr fts_calledaddr;
        struct PSAPaddr fts_callingaddr;

        OID     fts_context;

        int     fts_manage;

        int     fts_class;

        int     fts_units;
        int     fts_attrs;

        PE      fts_sharedASE;

        int     fts_fqos;

        struct FTAMcontentlist fts_contents;

        char   *fts_initiator;
        char   *fts_account;
        char   *fts_password;
        int     fts_passlen;

        int     fts_ssdusize;

        struct QOStype fts_qos;
    };
```

The elements of this structure are:

`fts_sd`: the association-descriptor to be used to reference this association;

`fts_calledtitle`: information on the called application-entity, if any (consult Section 2.2.1 of *Volume One*);

`fts_callingtitle`: information on the the calling application-entity, if any;

`fts_calledaddr`: the called presentation address (consult Section 2.2 of *Volume Two*);

`fts_callingaddr`: the calling presentation address;

`fts_context`: the application context name;

`fts_manage`: a flag indicating if presentation context management is available for use (zero if unavailable);

`fts_class`: the inclusive-or of the service classes offered;

`fts_units`: the functional units;

`fts_attrs`: the attribute groups;

`fts_sharedASE`: shared ASE information;

`fts_fqos`: the FTAM Quality-of-Service;

`fts_contents`: the contents type list;

`fts_initiator`: the user-identity of the initiator, if any;

`fts_account`: the account, if any;

`fts_password/fts_passlen`: the password (and its length), if any;

`fts_ssdusize`: the largest atomic SSDU size that can be used on the underlying connection; and,

`qos`: the quality of service on the connection (see Section 4.6.2 in *Volume Two*).

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

```
void    FTSFREE (fts)
struct FTAMstart *fts;
```

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

If the call to `FInit` is not successful, then an F-P-ABORT.INDICATION event is simulated, and the relevant information is returned in an encoded `FTAMindication` structure.

```
struct FTAMindication {
    int     fti_type;
#define FTI_FINISH      0x00
#define FTI_ABORT       0x01
#define FTI_MANAGEMENT  0x02
#define FTI_BULKBEGIN   0x03
#define FTI_BULKEND     0x04
#define FTI_ACCESS      0x05
#define FTI_READWRITE   0x06
#define FTI_DATA        0x07
#define FTI_DATAEND     0x08
#define FTI_CANCEL      0x09
#define FTI_TRANSEND    0x10

    union {
        struct FTAMfinish   fti_un_finish;
        struct FTAMabort    fti_un_abort;
        struct FTAMgroup    fti_un_group;
        struct FTAMaccess   fti_un_access;
        struct FTAMreadwrite fti_un_readwrite;
        struct PSAPdata     fti_un_data;
        struct FTAMdataend  fti_un_dataend;
        struct FTAMcancel   fti_un_cancel;
        struct FTAMtransend fti_un_transend;
    }   fti_un;
#define fti_finish      fti_un.fti_un_finish
#define fti_abort       fti_un.fti_un_abort
#define fti_group       fti_un.fti_un_group
#define fti_access      fti_un.fti_un_access
#define fti_readwrite   fti_un.fti_un_readwrite
#define fti_data        fti_un.fti_un_data
#define fti_dataend     fti_un.fti_un_dataend
```

```
#define fti_cancel      fti_un.fti_un_cancel
#define fti_transend    fti_un.fti_un_transend
};
```

As shown, this structure is really a discriminated union (a structure with a tag element followed by a union). Hence, on a failure return, one first coerces a pointer to the `FTAMabort` structure contained therein, and then consults the elements of that structure.

```
struct FTAMabort {
    int     fta_peer;

    int     fta_action;

#define NFDIAG  5
    int     fta_ndiag;
    struct FTAMdiagnostic fta_diags[NFDIAG];
};
```

The elements of an `FTAMabort` structure are:

`fta_peer`: if set, indicates that a user-initiated abort occurred (an F-U-ABORT.INDICATION event); if not set, indicates that a provider-initiated abort occurred (an F-P-ABORT.INDICATION event);

`fta_action`: an action result; and,

`fta_diags/fta_ndiag`: any diagnostic information (and the number of diagnostics present).

For each diagnostic present (typically only one) in the `FTAMabort` structure, information regarding the failure is present. In particular, the error code can be found in the the `ftd_identifier` element of each diagnostic.

After examining the information returned by `FInit` on a successful call (and possibly after examining the argument vector), the responder should either accept or reject the association. For either response, the responder should use the `FInitializeResponse` routine (which corresponds to the F-INITIALIZE.RESPONSE action).

```
int     FInitializeResponse (sd, state, action, context,
        respondtitle, respondaddr, manage, class, units,
        attrs, sharedASE, fqos, contents, diag, ndiag, fti)
int     sd;
int     state,
        action,
        manage,
        class,
        units,
        attrs,
        fqos;
OID     context;
AEI     respondtitle;
struct PSAPaddr *respondaddr;
PE      sharedASE;
struct FTAMcontentlist *contents;
struct FTAMdiagnostic diag[];
int     ndiag;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`sd`: the association-descriptor;

`state`: a state result;

`action`: an action result;

`context`: the application context name (use the manifest constant `NULLOID` if the proposed name should be used);

`manage`: the negotiated use of presentation context management;

`respondtitle`: information on the responding application-entity, if any;

`respondaddr`: the responding presentation address, if any;

`manage`: the negotiated service class;

`units`: the negotiated functional units (specify both mandatory and optional functional units for the service class);

`attrs`: the negotiated attribute groups;

`sharedASE`: shared ASE information;

`fqos`: the negotiated FTAM Quality-of-Service;

`contents`: the negotiated contents type list;

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

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

If the call to `FInitializeResponse` is successful, then association establishment has now been completed. Otherwise, if the call failed and the reason is "fatal", then the association is lost.

2.4.2 Initiator

A program wishing to associate itself with a filestore calls the routine `FInitializeRequest`, which corresponds to the user taking the F-INITIALIZE.REQUEST action.

```
int     FInitializeRequest (context, callingtitle, calledtitle,
        callingaddr, calledaddr, manage, class, units, attrs,
        sharedASE, fqos, contents, initiator, account,
        password, passlen, qos, tracing, ftc, fti)
OID     context;
AEI     calledtitle,
        callingtitle;
struct PSAPaddr *calledaddr,
                *callingaddr;
int     manage,
        class,
        units,
        attrs,
        fqos,
        passlen;
PE      sharedASE;
struct FTAMcontentlist *contents;
```

```
char   *initiator,
       *account,
       *password;
struct QOStype *qos;
IFP     tracing;
struct FTAMconnect *ftc;
struct FTAMindication *fti;
```

The parameters to this procedure are:

`context`: the application context name (use the manifest constant `NULLOID` to invoke the default file transfer context);

`callingtitle`: information on the the calling application-entity, if any (consult Section 2.2.1 of *Volume One*);

`calledtitle`: information on the called application-entity, if any;

`callingaddr`: the calling presentation address (consult Section 2.2 of *Volume Two*);

`calledaddr`: the called presentation address;

`manage`: a flag indicating if presentation context management is available for use (zero if unavailable);

`class`: the proposed service classes;

`units`: the proposed functional units (specify both mandatory and optional functional units for the indicated service class);

`attrs`: the proposed attribute groups;

`sharedASE`: shared ASE information;

`fqos`: the proposed FTAM Quality-of-Service;

`contents`: the contents type list;

`initiator`: the user-identity of the initiator, if any;

`account`: the account, if any

`password/passlen`: the password (and its length), if any;

`qos`: the quality of service on the connection (see Section 4.6.2 in *Volume Two*);

`tracing`: the address of a tracing routine to be invoked when an FTAM event occurs (consult Section 2.5.3);

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

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

If the call to `FInitializeRequest` is successful (the F-INITIALIZE.CONFIRMATION event occurs), it returns information in the `ftc` parameter which is a pointer to an `FTAMconnect` structure.

```
struct FTAMconnect {
    int     ftc_sd;

    int     ftc_state;
    int     ftc_action;

    AEInfo  ftc_respondtitle;

    struct PSAPaddr ftc_respondaddr;

    int     ftc_manage;

    int     ftc_units;
    int     ftc_attrs;

    int     ftc_rollback;

    struct FTAMcontentlist ftc_contents;

    int     ftc_ndiag;
    struct FTAMdiagnostic  ftc_diags[NFDIAG];
```

```
        int     ftc_ssdusize;

        struct QOStype ftc_qos;
    };
```

The elements of an FTAMconnect structure are:

ftc_sd: the association-descriptor to be used to reference this association;

ftc_state: a state result

ftc_action: an action result

ftc_respondtitle: the responding application-entity title, if any;

ftc_respondaddr: the responding presentation address, if any;

ftc_manage: the negotiated use of presentation context management;

ftc_units: the negotiated functional units;

ftc_attrs: the negotiated attribute groups;

ftc_rollback: the negotiated rollback availability;

ftc_contents: the negotiated contents type list;

ftc_diags/ftc_ndiag: any diagnostic information (and the number of diagnostics present)

ftc_ssdusize: the largest atomic SSDU size that can be used on the underlying connection; and,

ftc_qos: the quality of service on the connection (see Section 4.6.2 in *Volume Two*).

If the call to FInitializeRequest is successful, and the ftc_state element is set to FSTATE_SUCCESS, then association establishment has completed. If the call is successful, but the acc_result element is not FSTATE_SUCCESS, the the association attempt has been rejected and the diagnostics present indicate the reason for the rejection. Otherwise, if the call fails then the association is not established and the FTAMabort structure contained in the FTAMindication discriminated union has been updated.

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

```
void    FTCFREE (ftc)
struct FTAMconnect *ftc;
```

The macro frees only the data allocated by FInitializeRequest, and not the FTAMconnect structure itself. Further, FTCFREE should be called only if the call to the FInitializeRequest routine returned OK.

2.5 Event Handling

Once the association has been established, an association-descriptor is used to reference the connection. This is usually the first parameter given to any of the remaining routines in the *libftam*(3n) library. Further, the last parameter is usually a pointer to an FTAMindication structure (as described on page 34). If a call to one of these routines fails, then the structure is updated. Consult the FTAMabort element of the FTAMindication structure and look at the diagnostics (typically only one) present. When a diagnostic is reported by the provider, the ftd_type element indicates if the error is "fatal" by having the value DIAG_PERM. Other values, such as DIAG_TRANS indicate non-fatal errors (the association still exists). The most common non-fatal error to occur is FS_GEN_WAITING which indicates that an indication is waiting to be read.

The FWaitRequest routine is used to wait for an event to occur.

```
int     FWaitRequest (sd, secs, fti)
int     sd;
int     secs;
struct FTAMindication *fti;
```

The parameters to this procedure are:

sd: the association-descriptor;

`secs`: the maximum number of seconds to wait for the event (a value of `NOTOK` indicates that the call should block indefinitely, whereas a value of `OK` indicates that the call should not block at all, e.g., a polling action); and,

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

If the call to `FWaitRequest` returns the manifest constant `NOTOK`, then the `FTAMabort` structure contained in the `FTAMindication` parameter `fti` contains the reason for the failure. Similarly, if the manifest constant `OK` is returned, an event is encoded in the `fti` parameter, depending on the value of the `fti_type` element:

Value	Event
FTI_FINISH	F-TERMINATE.INDICATION
FTI_MANAGEMENT	grouped management operation
FTI_BULKBEGIN	begin of grouped file transfer operation
FTI_BULKEND	end of grouped file transfer operation
FTI_ACCESS	F-LOCATE.INDICATION F-LOCATE.CONFIRMATION F-ERASE.INDICATION F-ERASE.CONFIRMATION
FTI_READWRITE	F-READ.INDICATION F-WRITE.INDICATION
FTI_DATA	F-DATA.INDICATION
FTI_DATAEND	F-DATA-END.INDICATION
FTI_CANCEL	F-CANCEL.INDICATION F-CANCEL.CONFIRMATION
FTI_TRANSEND	F-TRANSFER-END.INDICATION F-TRANSFER-END.CONFIRMATION

These are now discussed in turn.

Termination Indication

When an event associated with association release occurs, the `fti_type` field of the `fti` parameter contains the value `FTI_FINISH`. Further, an `FTAMfinish` structure is contained inside the `fti` parameter, indicating an F-TERMINATE.INDICATION event.

```
struct FTAMfinish {
    int     ftf_sharedASE;
};
```

The elements of this structure are:

`ftf_sharedASE`: shared ASE information.

The `FTFFREE` macro can be used to free the storage associated with an `FTAMdfinish` structure (without freeing the structure itself). It behaves as if it was defined as:

```
void    FTFFREE (ftf)
struct FTAMfinish *ftf;
```

Group Indications

Use of the grouping function unit is required in the current implementation, hence when a transfer or management indication occurs, the `fti_type` field of the `fti` parameter contains one of three values:

Value	Event
FTI_MANAGEMENT	grouped management operation
FTI_BULKBEGIN	begin of grouped file transfer operation
FTI_BULKEND	end of grouped file transfer operation

and an `FTAMgroup` structure is contained inside the `fti` parameter.

```
struct FTAMgroup {
    int     ftg_threshold;

    int     ftg_flags;

    union {
        struct FTAMselect   ftg_un1_select;
        struct FTAMcreate   ftg_un1_create;
        struct FTAMclose    ftg_un1_close;
    }                   ftg_un1;
```

```
#define ftg_select      ftg_un1.ftg_un1_select
#define ftg_create      ftg_un1.ftg_un1_create
#define ftg_close       ftg_un1.ftg_un1_close

    struct FTAMreadattr ftg_readattr;

    struct FTAMchngattr ftg_chngattr;

    union {
        struct FTAMdeselect ftg_un2_deselect;
        struct FTAMdelete   ftg_un2_delete;
        struct FTAMopen     ftg_un2_open;
    } ftg_un2;
#define ftg_deselect    ftg_un2.ftg_un2_deselect
#define ftg_delete      ftg_un2.ftg_un2_delete
#define ftg_open        ftg_un2.ftg_un2_open
};
```

The elements of this structure are:

`ftg_threshold`: the grouping threshold;

`ftg_flags`: the indications which are present, containing the inclusive-or of any of the values:

Value	**Meaning**
FTG_BEGIN	F-BEGIN-GROUP
FTG_SELECT	F-SELECT
FTG_CREATE	F-CREATE
FTG_RDATTR	F-READ-ATTRIB
FTG_CHATTR	F-CHANGE-ATTRIB
FTG_OPEN	F-OPEN
FTG_CLOSE	F-CLOSE
FTG_DESELECT	F-DESELECT
FTG_DELETE	F-DELETE
FTG_END	F-END-GROUP

`ftg_select`: an `FTAMselect` structure;

`ftg_create`: an `FTAMcreate` structure;

ftg_close: an FTAMclose structure;

ftg_readattr: an FTAMreadattr structure;

ftg_chngattr: an FTAMchngattr structure;

ftg_deselect: an FTAMdeselect structure;

ftg_delete: an FTAMdelete structure; and,

ftg_open: an FTAMopen structure.

The FTGFREE macro can be used to free the storage associated with an FTAMgroup structure (without freeing the structure itself). It behaves as if it was defined as:

```
void    FTGFREE (ftg)
struct FTAMgroup *ftg;
```

If the flag FTG_SELECT is present, then ftg_select element contains an F-SELECT event encoded in an FTAMselect structure:

```
struct FTAMselect {
    int     ftse_state;
    int     ftse_action;

    struct FTAMattributes ftse_attrs;

    int     ftse_access;
    struct FTAMpasswords ftse_pwds;
    struct FTAMconcurrency ftse_conctl;

    PE      ftse_sharedASE;

    char   *ftse_account;

    int     ftse_ndiag;
    struct FTAMdiagnostic ftse_diags[NFDIAG];
};
```

The elements of this structure are:

`ftse_state`: the state result (.RESPONSE only);

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

`ftse_attrs`: file attributes;

`ftse_access`: requested access, containing the inclusive-or of any of the values found in Table 2.1 (.REQUEST only);

`ftse_pwds`: access passwords (.REQUEST only);

`ftse_conctl`: concurrency control (.REQUEST only);

`ftse_sharedASE`: shared ASE information (.REQUEST only);

`ftse_account`: account (.REQUEST only); and,

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

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

```
void    FTSEFREE (ftse)
struct FTAMselect *ftse;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMselect` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTSEFREE` as appropriate.

If the flag `FTG_CREATE` is present, then `ftg_create` element contains an F-CREATE event encoded in an `FTAMcreate` structure:

```
struct FTAMcreate {
    int     ftce_state;
    int     ftce_action;

    int     ftce_override;
    char   *ftce_create;
    int     ftce_crelen;
```

```
        struct FTAMattributes ftce_attrs;

        int     ftce_access;
        struct FTAMpasswords ftce_pwds;
        struct FTAMconcurrency ftce_conctl;

        PE      ftce_sharedASE;

        char   *ftce_account;

        int     ftce_ndiag;
        struct FTAMdiagnostic ftce_diags[NFDIAG];
    };
```

The elements of this structure are:

`ftce_state`: the state result (.RESPONSE only);

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

`ftce_override`: the override setting, one of:

Value	Meaning
FOVER_FAIL	fail, if already exists
FOVER_SELECT	select, if already exists
FOVER_WRITE	zero-truncate, if already exists
FOVER_DELETE	delete, if already exists

(.REQUEST only);

`ftce_attrs`: file attributes;

`ftce_create/ftce_crelen`: the create password and its length, if any (.REQUEST only);

`ftce_access`: requested access, containing the inclusive-or of any of the values found in Table 2.1 (.REQUEST only);

`ftce_pwds`: access passwords (.REQUEST only);

`ftce_conctl`: concurrency control (.REQUEST only);

`ftce_sharedASE`: shared ASE information (.REQUEST only);

`ftce_account`: account (.REQUEST only); and,

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

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

```
void    FTCEFREE (ftce)
struct FTAMcreate *ftce;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMcreate` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTCEFREE` as appropriate.

If the flag `FTG_RDATTR` is present, then `ftg_readattr` element contains an F-READ-ATTRIB event encoded in an `FTAMreadattr` structure:

```
struct FTAMreadattr {
    int     ftra_action;

    int     ftra_attrnames;

    struct FTAMattributes ftra_attrs;
    int     ftra_ndiag;
    struct FTAMdiagnostic ftra_diags[NFDIAG];
};
```

The elements of this structure are:

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

`ftra_attrnames`: the attributes to read, containing the inclusive-or of any of the values shown in Table 2.2 on page 27 (.REQUEST only);

`ftra_attrs`: the attribute values (.RESPONSE only); and,

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

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

```
void    FTRAFREE (ftra)
struct FTAMreadattr *ftra;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMreadattr` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTRAFREE` as appropriate.

If the flag `FTG_CHNGATTR` is present, then `ftg_chngattr` element contains an F-CHANGE-ATTRIB event encoded in an `FTAMchngattr` structure:

```
struct FTAMchngattr {
    int ftca_action;

    struct FTAMattributes ftca_attrs;

    int     ftca_ndiag;
    struct FTAMdiagnostic ftca_diags[NFDIAG];
};
```

The elements of this structure are:

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

`ftca_attrs`: the attribute values to change (in a .REQUEST), and possibly the new values (in a .RESPONSE); and,

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

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

```
void    FTCAFREE (ftca)
struct FTAMchngattr *ftca;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMchngattr` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTCAFREE` as appropriate.

If the flag `FTG_OPEN` is present, then `ftg_open` element contains an F-OPEN event encoded in an `FTAMopen` structure:

```
struct FTAMopen {
    int     ftop_state;
    int     ftop_action;

    int     ftop_mode;

    OID     ftop_contents;
    PE      ftop_parameter;
    struct FTAMconcurrency ftop_conctl;
    PE      ftop_sharedASE;

    int     ftop_locking;

    int     ftop_ndiag;
    struct FTAMdiagnostic ftop_diags[NFDIAG];
};
```

The elements of this structure are:

ftop_state: the state result (.RESPONSE only);

ftop_action: the action result (.RESPONSE only);

ftop_mode: the processing mode, containing the inclusive-or of any of the values shown in Table 2.1 on page 23 (.REQUEST only);

ftop_contents/ftop_parameter: the contents type;

ftop_conctl: concurrency control;

ftop_sharedASE: shared ASE information;

ftop_locking: enable FADU locking (.REQUEST only); and,

ftop_diags/ftop_ndiag: any diagnostic information (and the number of diagnostics present, .RESPONSE only).

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

```
void    FTOPFREE (ftop)
struct FTAMopen *ftop;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMopen` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTOPFREE` as appropriate.

If the flag `FTG_CLOSE` is present, then `ftg_close` element contains an F-CLOSE event encoded in an `FTAMclose` structure:

```
struct FTAMclose {
    int     ftcl_action;

    PE      ftcl_sharedASE;

    int     ftcl_ndiag;
    struct FTAMdiagnostic   ftcl_diags[NFDIAG];
};
```

The elements of this structure are:

`ftcl_action`: the action result;

`ftcl_sharedASE`: shared ASE information; and,

`ftcl_diags/ftcl_ndiag`: any diagnostic information (and the number of diagnostics present).

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

```
void    FTCLFREE (ftcl)
struct FTAMclose *ftcl;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMclose` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTCLFREE` as appropriate.

If the flag `FTG_DESELECT` is present, then `ftg_deselect` element contains an F-DESELECT event encoded in an `FTAMdeselect` structure:

```
struct FTAMdeselect {
    int     ftde_action;

    PE      ftde_sharedASE;

    struct FTAMcharging ftde_charges;
    int     ftde_ndiag;
    struct FTAMdiagnostic   ftde_diags[NFDIAG];
};
```

The elements of this structure are:

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

`ftde_sharedASE`: shared ASE information;

`ftde_charges`: any charges (.RESPONSE only); and,

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

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

```
void    FTDEFREE (ftde)
struct FTAMdeselect *ftde;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMdeselect` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTDEFREE` as appropriate.

If the flag `FTG_DELETE` is present, then `ftg_delete` element contains an F-DELETE event encoded in an `FTAMdelete` structure:

```
struct FTAMdelete {
    int     ftxe_action;

    PE      ftxe_sharedASE;

    struct FTAMcharging ftxe_charges;
    int     ftxe_ndiag;
    struct FTAMdiagnostic   ftxe_diags[NFDIAG];
};
```

The elements of this structure are:

ftxe_action: the action result (.RESPONSE only);

ftxe_sharedASE: shared ASE information;

ftxe_charges: any charges (.RESPONSE only); and,

ftxe_diags/ftxe_ndiag: any diagnostic information (and the number of diagnostics present, .RESPONSE only).

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

```
void    FTXEFREE (ftxe)
struct FTAMdeltee *ftxe;
```

The macro frees only the data allocated by `FWaitRequest`, and not the `FTAMdelete` structure itself. Note that the macro `FTGFREE` when applied to the group, will call `FTXEFREE` as appropriate.

Access Indications

When an event associated with file access occurs, the `fti_type` field of the `fti` parameter contains the value `FTI_ACCESS`. Further, an `FTAMaccess` structure is contained inside the `fti` parameter, indicating an F-LOCATE or F-ERASE event.

```
struct FTAMaccess {
    int     ftac_operation;

    int     ftac_action;

    struct FADUidentity ftac_identity;

    int     ftac_locking;

    int     ftac_ndiag;
    struct FTAMdiagnostic   ftac_diags[NFDIAG];
};
```

Access Operations	
FA_OPS_LOCATE	F-LOCATE
FA_OPS_ERASE	F-ERASE

Table 2.3: FTAM Access Operations

The elements of this structure are:

`ftac_operation`: the operation, one of the values shown in Table 2.3;

`ftac_action`: the action result (.CONFIRMATION only);

`ftac_identity`: the FADU identity (either .INDICATION and optionally on F-LOCATE.CONFIRMATION);

`ftac_locking`: FADU locked (F-LOCATE.INDICATION only); and,

`ftac_diags`/`ftac_ndiag`: any diagnostic information (and the number of diagnostics present, .CONFIRMATION only).

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

```
void    FTACFREE (ftac)
struct FTAMaccess *ftac;
```

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

Read/Write Indications

When an event associated with the beginning of bulk data transfer occurs, the `fti_type` field of the `fti` parameter contains the value `FTI_READWRITE`. Further, an `FTAMreadwrite` structure is contained inside the `fti` parameter, indicating an F-READ.INDICATION or F-WRITE.INDICATION event.

```
struct FTAMreadwrite {
    int     ftrw_operation;

    struct FADUidentity ftrw_identity;

    int     ftrw_context;
    int     ftrw_level;

    int     ftrw_locking;
};
```

The elements of this structure are:

`ftrw_operation`: the operation, one of the values shown in Table 2.4;

`ftrw_identity`: the FADU identity;

`ftrw_context/ftrw_level`: the access context, one of the values shown in Table 2.5 (F-READ.INDICATION only); and,

`ftrw_locking`: FADU locked.

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

```
void    FTRWFREE (ftrw)
struct FTAMreadwrite *ftrw;
```

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

Data Indications

When an event associated with user data occurs, the `fti_type` field of the `fti` parameter contains the value `FTI_DATA`, and an `PSAPdata` structure (described in Chapter 2 of *Volume Two*) is contained inside the `fti` parameter, indicating an F-DATA.INDICATION event. To distinguish between structure and contents, FTAM uses different presentation context identifiers. If the `pe_context` field of any presentation element contains the value

Data Operations	
FA_OPS_READ	F-READ
FA_OPS_INSERT	F-WRITE insert
FA_OPS_REPLACE	F-WRITE replace
FA_OPS_EXTEND	F-WRITE extend

Table 2.4: FTAM Data Operations

Access Contexts	
FA_ACC_HA	hierarchical all data units
FA_ACC_HN	hierarchical no data units
FA_ACC_FA	flat all data units
FA_ACC_FL	flat one level data units
FA_ACC_FS	flat single data unit
FA_ACC_UA	unstructured all data units
FA_ACC_US	unstructured single data unit

Table 2.5: FTAM Access Contexts

PE_DFLT_CONTEXT, then the element contains structuring information in the FTAM PCI, and the pe_id element should be consulted to determine the particular structuring information being conveyed:

Value	Meaning
FADU_NODESCR	node descriptor data element
FADU_ENTERTREE	enter subtree data element
FADU_EXITREE	exit subtree data element

Otherwise, the element contains file contents as defined in the PCI for the document type being transferred.

Data End Indication

When an event associated with the end of bulk data transfer occurs, the fti_type field of the fti parameter contains the value FTI_DATAEND. Further, an FTAMdataend structure is contained inside the fti parameter, indicating an F-DATA-END.INDICATION event.

```
struct FTAMdataend {
    int     ftda_action;

    int     ftda_ndiag;
    struct FTAMdiagnostic   ftda_diags[NFDIAG];
};
```

The elements of this structure are:

ftda_action: the action result; and,

ftda_diags/ftda_ndiag: any diagnostic information (and the number of diagnostics present).

Cancel Indications

When an event associated with the cancellation of bulk data transfer occurs, the fti_type field of the fti parameter contains the value FTI_CANCEL. Further, an FTAMcancel structure is contained inside the fti parameter, indicating an F-CANCEL event.

```
struct FTAMcancel {
    int     ftcn_action;

    PE      ftcn_sharedASE;

    int     ftcn_ndiag;
    struct FTAMdiagnostic   ftcn_diags[NFDIAG];
};
```

The elements of this structure are:

- `ftcn_action`: the action result;
- `ftcn_sharedASE`: shared ASE information; and,
- `ftcn_diags/ftcn_ndiag`: any diagnostic information (and the number of diagnostics present).

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

```
void    FTCNFREE (ftcn)
struct FTAMcancel *ftcn;
```

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

Transfer End Indications

When an event associated with the termination of bulk data transfer occurs, the `fti_type` field of the `fti` parameter contains the value `FTI_TRANSEND`. Further, an `FTAMtransend` structure is contained inside the `fti` parameter, indicating an F-TRANSFER-END event.

```
struct FTAMtransend {
    int     ftre_action;

    PE      ftre_sharedASE;

    int     ftre_ndiag;
    struct FTAMdiagnostic   ftre_diags[NFDIAG];
};
```

The elements of this structure are:

ftre_action: the action result (.RESPONSE only);

ftre_sharedASE: shared ASE information; and,

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

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

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

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

2.5.1 Asynchronous Event Handling

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

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

The parameters to this procedure are:

sd: the association-descriptor;

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

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

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

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

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

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

The parameters are:

`sd`: the association-descriptor; and,

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

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

2.5.2 Synchronous Event Multiplexing

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

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

2.5.3 Tracing

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

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

The parameters are:

`sd`: the association-descriptor;

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

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

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

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

The return value of the tracing routine is ignored.

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

```
LLog *ftam_log;
```

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

2.6 Grouped Operations: File Transfer

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

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

- F-END-GROUP.REQUEST.

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

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

The parameters to this procedure are:

sd: the association-descriptor;

ftg: a pointer to an FTAMgroup structure; and,

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

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

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

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

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

The parameters to this procedure are:

sd: the association-descriptor;

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

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

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

- F-BEGIN-GROUP.REQUEST;
- F-CLOSE.REQUEST;
- F-DESELECT.REQUEST or F-DELETE.REQUEST; and,
- F-END-GROUP.REQUEST.

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

Upon receiving an `FTI_BULKEND` indication, containing the appropriate .INDICATION events, the responder is required to generate the appropriate .RESPONSE events using the `FBulkEndResponse` routine.

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

2.7 File Access

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

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

Upon receiving an F-LOCATE.INDICATION or an F-ERASE.INDICATION event, the user is required to generate the appropriate .RESPONSE action using the `FAccessResponse` routine.

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`action`: the action result;

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

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

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

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

2.8 Data Transfer

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

2.8.1 Read/Write

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

`lock`: lock FADU; and,

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

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

2.8.2 Sending Data

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`action`: the action result;

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

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

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

2.8.3 Canceling Transfer

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`action`: the action result;

`sharedASE`: shared ASE information;

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

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

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

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

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

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

The parameters to this procedure are:

sd: the association-descriptor;

action: the action result;

sharedASE: shared ASE information;

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

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

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

2.8.4 Terminating Transfer

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

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

The parameters to this procedure are:

- `sd`: the association-descriptor;
- `sharedASE`: shared ASE information; and,
- `fti`: a pointer to an `FTAMindication` structure, which is always updated.

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

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

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

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

The parameters to this procedure are:

- `sd`: the association-descriptor;
- `action`: the action result;
- `sharedASE`: shared ASE information;
- `diag/ndiag`: a list of diagnostics (and the number of diagnostics in the list); and,
- `fti`: a pointer to an `FTAMindication` structure, which is updated only if the call fails.

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

2.9 Grouped Operations: File Management

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

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

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

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

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

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

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

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

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

The parameters to this procedure are:

sd: the association-descriptor;

ftg: a pointer to an FTAMgroup structure; and,

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

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

2.10 Association Release

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`sharedASE`: shared ASE information;

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

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

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

```
struct FTAMrelease {
    PE      ftr_sharedASE;

    struct FTAMcharging ftr_charges;
};
```

The elements of this structure are:

`ftr_sharedASE`: shared ASE information; and,

`ftr_charges`: any charges.

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

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

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

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

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`sharedASE`: shared ASE information;

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

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

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

2.11 Association Abort

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

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

The parameters to this procedure are:

`sd`: the association-descriptor;

`action`: an action result;

`diag/ndiag`: a list of diagnostics (and the number of diagnostics in the list); and, `fti`: a pointer to an `FTAMindication` structure, which is updated only if the call fails.

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

2.12 Error Conventions

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

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

2.13 Compiling and Loading

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

2.14 An Example

Read Section 4.1 on page 80.

2.15 For Further Reading

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

Chapter 3

The ISO Documents Database

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

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

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

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

3.1 Accessing the Database

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

```
struct isodocument {
    char   *id_entry;

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

The elements of this structure are:

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

`id_type`: the document type;

`id_constraint`: the constraint set;

`id_abstract`: the abstract syntax; and,

`id_transfer`: the transfer syntax.

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

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

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

The routine `setisodocument` opens and rewinds the database.

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

The parameter to this procedure is:

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

The routine `endisodocument` closes the database.

```
int     endisodocument ()
```

Both of these routines return non-zero on success and zero otherwise.

There are two routines used to fetch a particular entry in the database. The routine `getisodocumentbyentry` maps textual descriptions into the internal form.

```
struct isodocument *getisodocumentbyentry (entry)
char   *entry;
```

The parameter to this procedure is:

`entry`: the descriptor of the object.

and returns the `isodocument` structure describing that document. On failure, the manifest constant `NULL` is returned instead.

The routine `getisodocumentbytype` performs the inverse function.

```
struct isodocument *getisodocumentbytype (tpye)
OID     type;
```

The parameter to this procedure is:

`type`: the identifier of the document.

On a successful return, an `isodocument` structure describing the document is returned.

Chapter 4

UNIX Implementation

The File Transfer, Access, and Management (FTAM) standard is the OSI file service. Included in the release is a fairly complete FTAM implementation in the context of the particular file services it offers. It is a minimal implementation in as much as it offers only four core services: transfer of binary files, transfer of text files, directory listings, and file management. The implementation included has been tested on both Berkeley and AT&T SVR2 and SVR3 UNIX. Both the FTAM initiator and responder programs have UNIX manual entries.

4.1 Implementation

If you have access to the source tree for this release, the directory *ftam2/* contains the code for the responder and initiator.

4.1.1 The Initiator

There is currently one initiator which uses FTAM: *ftam*(1c). Supported are: the no-recovery FTAM-QoS; any of the transfer, management, and transfer and management service classes; the kernel, read, write, limited file management, enhanced file management, and grouping functional units; and, the kernel and storage attribute groups. Only three document types are supported as of this writing: unstructured text files (FTAM-1), unstructured binary files (FTAM-3), and filedirectory files (NIST-9).

The *ftam* program is an interactive FTAM initiator which prompts the user for commands. Generating an interrupt, usually by typing control-C ('`^C`'), at the top-level does nothing, but generating an interrupt twice in a row at the top-level terminates *ftam*; generating an interrupt during additional prompting causes *ftam* to abort the command; typing generating an interrupt during file transfer causes the transfer to be aborted.

Commands

Here are the commands to *ftam*:

append `source destination` Appends to a file in the filestore.

cd `[dir]` Changes the working directory on the virtual filestore. This requires the **realstore** variable to be set appropriately.

chgrp `group file` ... Changes the account attribute of the named files.

close Terminates the association with the virtual filestore.

dir `[file]` Prints a long directory listing.

echo `file` ... Simply echoes any arguments. Useful for seeing how globbed expressions will evaluate.

fdir `stream [file]` Prints a long directory listing to a file or program. If `stream` starts with a vertical bar ('`|`') then the named program is invoked; otherwise the named file is written.

fls `stream [file]` Prints a directory listing to a file or program. If `stream` starts with a vertical bar ('`|`') then the named program is invoked; otherwise the named file is written.

get `source destination` Retrieves a file.

help `[command]` Prints help information. For detailed information, try "`help␣?`".

lcd `[file]` Changes the working directory on the local system.

ls `[file]` Prints a directory listing.

mkdir `dir` ... Creates a directory.

mv `source destination` Renames a file.

open `host user [account]` Associates with the virtual filestore.

put `source destination` Stores a file.

pwd Prints the working directories.

quit Terminates the association with the virtual filestore and exits.

rm `file` ... Deletes a file.

set `variable value` Displays or changes variables. For detailed information, try "`set␣?`".

status Shows the current status.

Variables

Here are the variables which effect *ftam*'s behavior.

bell Rings the bell after each command terminates. Useful for long file transfers when you want to attend to other matters and be notified when you can type another command. Boolean (values: **on** or **off**).

debug This enables voluminous output during file transfers, among other things. Boolean.

glob This enables the expansion of shell meta-characters. Operations which perform globbing require the **realstore** variable to be set appropriately. Boolean.

hash This enables the printing of hash marks during file transfers. Values: `off`, `on`, `total`.

override This sets the creation override mode for files being written to the virtual filestore. If the file being created already exists, then one of four alternatives is taken. Values:

`fail`: the creation operation;

`select`: use the existing file with its old contents and attributes;

`write`: zero-truncate if it already exists, and use the existing file with its old attributes; and,

`delete`: if it already exists, then create a new file with new attributes.

This defaults to `write`.

qualifier This sets the "qualifier" portion of the srevice which *ftam* will associate with. It is needed when using the current implementation of the MITRE FTAM/FTP gateway. This defaults to `filestore`.

query This determines if *ftam* should ask the user to confirm operations involving globbing that expand to more than one filename. Boolean. This defaults to `on`.

realstore Sets the type of remote realstore associated with the virtual filestore. This is used to help *ftam* act friendlier to the user! Values: `unix`, `unknown`.

NOTE:	The concept of a **realstore** is contrary to the notion of open systems as it is an $N * M$ (not $N + M$) method.

trace This enables the tracing of FTAM PDUs. Boolean.

tracefile This defines the file where tracing information is appended.

type This defines the file transfer mode to use. Values: `default`, `binary`, and `text`.

verbose This enables printing of informative diagnostics during operation. Boolean.

watch This enables watch mode, something in between debug mode (too voluminous), and verbose mode (not informative enough). Boolean.

userdn This defines the Distinguished Name to be used when binding to the Directory for AE-lookup. DN-string.

*xyz***sapfile** This defines the file where *xyz*PDU tracing information is appended. Values: any filename, or `-` for the diagnostic output.

*xyz***saplevel** This enables tracing of the *xyz* module.
Values: `none`, `exceptions`, `notice`, `pdus`, `trace`, and `debug`.

Options

Here are the command line options:

-a *acct* Sets the account to be used on the virtual filestore.

-d Sets **debug**.

-f Inhibits reading of the user's *$HOME/.ftamrc* file on startup.

-h Sets **hash**.

-o *mode* Sets **override**.

-t Sets **trace**.

-u *user* Sets the initiator identity to be used on the virtual filestore.

-v Sets **verbose** (default for interactive use).

-w Sets **watch**.

4.1.2 The Responder

The *ftamd*(8c) program implements the file service. It implements *filestore* abstractions directly on the UNIX filesystem. Supported are: the no-recovery FTAM-QoS; any of the transfer, management, and transfer and management service classes; the kernel, read, write, limited file management, enhanced file management, and grouping functional units; and, the kernel and storage attribute groups. Only three document types are supported as of this writing: unstructured text files (FTAM-1), unstructured binary files (FTAM-3), and filedirectory files (NIST-9).

Authentication

An FTAM initiator must be listed in the *passwd*(5) file and have a non-empty password. Further, as with the *ftpd*(8c) daemon, the username must not appear in the *ftamusers* file in the ISODE `ETCDIR` directory or in the */etc/ftpusers* file. (In fact, many of the mechanisms in *ftamd* are based on the *ftpd* program supplied with Berkeley UNIX.)

If the username `ANON` or `ftp` is given, then *ftamd* treats this as a guest access, similar to the "anonymous" facility supported by the *ftpd* daemon. An entry in the */etc/passwd* file for user `ftp` must be present with a non-zero UID. For guest access, a *chroot*(2) to the guest home directory is executed to restrict access to the system.

NOTE:	The anonymous account is inherently dangerous and should be avoided when possible. It is also inherently useful.

The Berkeley UNIX version of this program runs with the effective UID of the FTAM initiator, but also with the real UID of the super-user. This is necessary to change the account attribute on files using *chown*(2). The possible security holes have been extensively considered, but may be incomplete.

The AT&T UNIX version, which lacks kernel support for this technique, acts differently. Immediately upon association establishment, it changes both the real and effective UID to that of the FTAM initiator. To change the account attribute on files, it invokes the *chgrp*(1) program. Similarly, to create or delete directories, it invokes either the *mkdir*(1) program or the *rmdir*(1) program. Finally, it is unable to change the filesize attribute to a non-zero value if this value is smaller than the current filesize.

Finally, on Berkeley UNIX systems, the *wtmp*(5) file is updated as appropriate. (We couldn't figure out how to update *wtmp* under AT&T UNIX using the description in the SVID!)

Virtual Filestore

Here are the file attribute mappings. Most attributes are derived by doing a *stat*(2) on the file and then examining the indicated field in the resulting structure.

filename A single component, relative to the user's *$HOME*. Changing this attribute is equivalent to a *rename*(2).

contents-type Based on the `st_mode` field:

> **NIST-9** for directories;
>
> **FTAM-1** for regular files appearing to be textual; and,
>
> **FTAM-3** for all other regular files.
>
> Files that are neither regular nor directories are inaccessible via this implementation of the VFS (i.e., special files).

account The `st_gid` field according to *group*(5). Changing this attribute is equivalent to a *chgrp*(1).

date-and-time-of-creation The `st_mtime` field.

date-and-time-of-last-modification The `st_mtime` field.

date-and-time-of-last-read-access The `st_atime` field.

date-and-time-of-last-attribute-modification
The `st_ctime` field.

identity-of-creator The `st_uid` field according to *passwd*(5).

identity-of-last-modifier The `st_uid` field according to *passwd*(5) (if the value of the `st_mode` field guarantees uniqueness).

identity-of-last-reader The `st_uid` field according to *passwd*(5) (if the value of the `st_mode` field guarantees uniqueness).

identity-of-last-attribute-modifier The `st_uid` field according to *passwd*(5) (if the value of the `st_mode` field guarantees uniqueness).

file-availability Immediate.

permitted-actions Depends on the `st_mode` the as interpreted by *access*(2): `R_OK` for permission to read; `W_OK` for permission to write; permission is always granted to read attributes; permission is granted to change attributes if the initiator has uid equal to the `st_uid` field; and, permission to delete is based on writability of parent directory.

filesize The `st_size` field.

future-filesize Not available.

access-control Not available.

encryption-name Not available.

legal-qualifications Not available.

private-use Not available.

The activity attribute mappings are straight-forward. The read action corresponds to reading UNIX files. The insert, replace, extend, and erase actions correspond to writing UNIX files. Concurrency control is supported for reading and writing, but not for reading or changing attributes, or for deleting files.

Chapter 5

FTAM-FTP gateway

The FTAM/FTP gateway is an application-gateway for file service. The gateway is actually two programs: one which acts as an FTAM responder and an FTP client, and the other which acts as an FTP server and an FTAM initiator. Note that the gateway currently resides on a different location than the standard FTAM responder and FTP server.

The implementation included runs only on Berkeley UNIX.

5.1 Implementation

If you have access to the source tree for this release, the directories *ftam-ftp/* and *ftp-ftam/* contains the code for the two programs.

5.1.1 The FTAM/FTP side

The FTAM/FTP side of the gateway appears to implement the responder side of the FTAM service, but actually acts as an FTP client in order to provide this service.

The true destination is encoded in the user name (i.e., `user@tcphost`).

Note that the FTAM/FTP side is available on a different presentation address than the FTAM service on the gateway host. To select the FTAM/FTP side, tell your FTAM initiator to associate with the service having "qualifier" `ftpstore` on the gateway host. For example, using *ftam*(1c):

```
% ftam
```

```
ftam> set qualifier ftpstore
ftam> open gateway
user (gateway:user): user@tcphost
password (gateway:user@tcphost):
```

Limitations

File information is limited to file names. All file access rights are assumed until access is attempted; the FTP server of the utlimate destination grants or denies action permission at the time of file access.

Empty directories may not be recognized depending on the FTP server of the destination machine. This bug manifests itself when trying to remove an empty directory.

5.1.2 The FTP/FTAM side

The FTP/FTAM side of the gateways appears to be an FTP server, but actually acts as an FTAM initiator in order to provide this service.

The true destination is encoded in the user name (i.e., `user@osihost`), or by using the FTP SITE command. If further accounting information is required by the true destnation, the FTP ACCT command is used seperately and the SITE command must be used to specify the destination.

Note that the FTP/FTAM side is available on a different port than the FTP server on the gateway host. To select the FTP/FTAM side, tell your FTP client to connect to port 531 on the gateway host. For example, using *ftp*(1c):

```
% ftp
ftp> open gateway 531
ftp> open gateway
Name (gateway:user): user@osihost
Password:
```

Limitations

The FTP CD and PWD commands are not supported by the gateway (there is no equivalent in the FTAM service and it is too difficult to emulate at the gateway).

Part III

Virtual Terminal

Chapter 6

UNIX Implementation

The Virtual Terminal (VT) standard is the OSI terminal service. Included in the release is an implementation of VT that is roughly comparable to an average *telnet* implementation.

The implementation included runs only on Berkeley UNIX.

6.1 Implementation

If you have access to the source tree for this release, the directory *vt/* contains the code for the responder and initiator.

6.1.1 The Initiator

There is currently one initiator which uses VT: *vt*(1c). Supported is the VT TELNET profile from the NIST OSI Implementors Workshop Agreements.

The *vt* program is an interactive VT initiator which prompts the user for commands. Command mode is entered by typing an escape character ("`^]`" by default).

Commands

Here are the commands to *vt*:

ayt Sends an "are you there" message to the remote login server.

break Flushes data queued in both directions and interrupts the remote process.

close Terminates the association with the terminal service.

escape Set the "escape character" used to enter command mode. Control characters may be specified as "`^`" followed by a single letter (e.g., "control-X" is "`^X`").

help `[command]` Prints help information. For detailed information, try "`help␣?`".

open `host user [account]` Associates with the temrinal service.

quit Terminates the association with the terminal service and exits.

set `variable value` Displays or changes variables. For detailed information, try "`set␣?`".

status Shows the current status.

suspend Suspends *vt*. This works only if the program was invoked under a shell with job control (e.g., *csh*).

Variables

Here are the variables which effect *vt*'s behavior.

crmod This enables the mapping of CR characters received from the remote host into CR-LF pairs. This does not affect those characters typed by the user, only those received. Boolean (values: **on** or **off**).

debug This enables voluminous output during file transfers, among other things. Boolean.

echo Determines whether echoing is done locally or remotely. Values: `local`, `remote`.

escape Sets the escape character. Value: any single character or "`^`" followed by a character.

options Determines if option processing is shown. Boolean.

repetoire Determines which character set (repetoire) shall be used. Values: `ascii`, `transparent` (binary).

tracelevel This enables the tracing of VT. Values: `none`, `exceptions`, `notice`, `pdus`, `trace`, and `debug`.

tracefile This defines the file where tracing information is appended. Values: any filename, or `-` for the diagnostic output.

verbose This enables printing of informative diagnostics during operation. Boolean.

*xyz***sapfile** This defines the file where *xyz*PDU tracing information is appended. Values: any filename, or `-` for the diagnostic output.

*xyz***saplevel** This enables tracing of the *xyz* module. Values: `none`, `exceptions`, `notice`, `pdus`, `trace`, and `debug`.

Options

Here are the command line options:

-B Do not negotiate use of the VT BREAK functional unit.

-D Use the VT asynchronous DEFAULT profile rather than the TELNET profile.

-F *logfile* Sets the logging file to be used.

-g Use only the G0 character set for the *ascii* repetoire (graphics only).

-f Inhibits reading of the user's *$HOME/.vtrc* file on startup.

6.1.2 The Responder

The *vtd*(8c) program implements the terminal service. Supported is the VT TELNET profile from the NIST OSI Implementors Workshop Agreements.

Options

Here are the command line options:

-F *logfile* Sets the logging file to be used.

-d *level* Sets the debug level from `0` (none) to `7` (verbose).

Part IV

Miscellaneous Applications

Chapter 7

The ISODE Little Services

The *ISODE Little Services* are examples of a few simple applications written using *The ISO Development Environment*. The programs described herein should work on all systems on which the software runs. All of these programs have UNIX manual entries.

The little services are based on the protocols of the same name found in the DoD TCP/IP protocol suite. There are several services:

utcTime: the universal time

genTime: the generalized time

timeOfDay: the current date and time since the UNIX epoch

users: the users logged in on the system

charGen: a character generation pattern

qotd: a quote of the data

finger: "fingers" users logged in

pwdGen: six pseudo-randomly generated (allegedly mnemonic) passwords

tellUser: sends a message to a remote user

ping: ping test for performance measurement

sink: sink test for performance measurement

echo: echo test for performance measurement

Only the *finger* service takes any arguments, the individuals on whom to report.

7.1 Implementation

If you have access to the source tree for this release, the directory *imisc/* contains the code for the responder and initiator.

7.1.1 The Initiator

The *imisc*(1c) program is the initiator which requests the little services. If invoked with arguments, it executes that exact operation and terminates. Otherwise, it enters interactive mode, prompting for each operation and argument until end-of-file is found.

In addition, the pseudo-operations `help` and `quit` do the obvious things.

7.1.2 The Responder

The *imiscd*(8c) program is the responder which implements the little services.

As shown in Figure 7.1 on the following page, the ROS-based definition is very simple.

```
IMISC DEFINITIONS ::=

BEGIN

-- operations

utcTime OPERATION
        RESULT       UTCResult
        ERRORS       { congested, unableToDetermineTime }
        ::=          0

timeOfDay OPERATION
        RESULT       TimeResult
        ERRORS       { congested, unableToDetermineTime }
        ::=          1

users OPERATION
        RESULT       IA5List
        ERRORS       { congested, unableToOpenFile }
        ::=          2

charGen OPERATION
        RESULT       IA5List
        ERRORS       { congested }
        ::=          3

qotd    OPERATION
        RESULT       IA5List
        ERRORS       { congested, unableToAccessFile, unableToPipe,
                       unableToFork, errorReading }
        ::=          4

finger  OPERATION
        ARGUMENT     IA5List
        RESULT       IA5List
        ERRORS       { congested, unableToAccessFile, unableToPipe,
                       unableToFork, errorReading }
        ::=          5
```

```
pwdGen OPERATION
        RESULT      IA5List
        ERRORS      { congested }
        ::=         6

genTime OPERATION
        RESULT      GenResult
        ERRORS      { congested, unableToDetermineTime }
        ::=         7

tellUser OPERATION
        ARGUMENT    IA5List
        RESULT      NULL
        ERRORS      { congested, unableToOpenFile, userNotLoggedIn }
        ::=         8

ping OPERATION
        ARGUMENT    Empty
        RESULT      Empty
        ERRORS      { congested }
        ::=         9

sink OPERATION
        ARGUMENT    Data
        RESULT      Empty
        ERRORS      { congested }
        ::=         10

echo OPERATION
        ARGUMENT    Data
        RESULT      Data
        ERRORS      { congested }
        ::=         11

-- errors

congested
        ERROR
```

```
        ::=         0

unableToDetermineTime
        ERROR
        ::=         1

unableToOpenFile
        ERROR
        PARAMETER   IA5List
        ::=         2

unableToAccessFile
        ERROR
        PARAMETER   IA5List
        ::=         3

unableToPipe
        ERROR
        PARAMETER   IA5List
        ::=         4

unableToFork
        ERROR
        PARAMETER   IA5List
        ::=         5

errorReading
        ERROR
        PARAMETER   IA5List
        ::=         6

userNotLoggedIn
        ERROR
        ::=         7

-- types

IA5List ::=
        SEQUENCE OF IA5String
```

UTCResult ::=
    UniversalTime

TimeResult ::=
    INTEGER

GenResult ::=
    GeneralizedTime

END

Figure 7.1: ROS definition of ISODE Little Services

Bibliography

[ISO88] Information Processing Systems — File Transfer, Access, and Management. International Organization for Standardization and International Electrotechnical Committee, April 1988. Final Text of Draft International Standard 8571.

[KR78] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Software Series. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[Pos81] Jon B. Postel. Transmission Control Protocol. Request for Comments 793, DDN Network Information Center, SRI International, September 1981. See also MIL-STD 1778.

[RC86] Marshall T. Rose and Dwight E. Cass. OSI Transport Services on top of the TCP. *Computer Networks and ISDN Systems*, 12(3), 1986. Also available as NRTC Technical Paper #700.

[Ros90] Marshall T. Rose. *The Open Book: A Practical Perspective on Open Systems Interconnection*. Prentice-hall, 1990. ISBN 0–13–643016–3.

Index