

# **HP FTAM/9000 Programmer's Guide**

**Edition 5**



**B1033-90014**  
**HP 9000 Networking**  
**E0597**

Printed in: U.S.A.

© Copyright 1997, Hewlett-Packard Company.

---

## Legal Notices

The information in this document is subject to change without notice.

*Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.* Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

**Warranty.** A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

**Restricted Rights Legend.** Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

HEWLETT-PACKARD COMPANY 3000 Hanover Street Palo Alto, California 94304 U.S.A.

Use of this manual and flexible disk(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

**Copyright Notices.** ©copyright 1983-97 Hewlett-Packard Company, all rights reserved.

Reproduction, adaptation, or translation of this document without prior written permission is prohibited, except as allowed under the copyright laws.

©copyright 1979, 1980, 1983, 1985-93 Regents of the University of California

This software is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

©copyright 1980, 1984, 1986 Novell, Inc.

©copyright 1986-1992 Sun Microsystems, Inc.

©copyright 1985-86, 1988 Massachusetts Institute of Technology.

©copyright 1989-93 The Open Software Foundation, Inc.

©copyright 1986 Digital Equipment Corporation.

©copyright 1990 Motorola, Inc.

©copyright 1990, 1991, 1992 Cornell University

©copyright 1989-1991 The University of Maryland

©copyright 1988 Carnegie Mellon University

**Trademark Notices** UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X Window System is a trademark of the Massachusetts Institute of Technology.

MS-DOS and Microsoft are U.S. registered trademarks of Microsoft Corporation.

OSF/Motif is a trademark of the Open Software Foundation, Inc. in the U.S. and other countries.



---

## **Printing History**

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

First Edition: April 1991 (HP-UX Release 8.0)

Second Edition: November 1992 (HP-UX Release 9.0)

Third Edition: January 1995 (HP-UX Release 10.x)

Fourth Edition: March 1997 (HP-UX Release 10.10)

Fifth Edition: May 1997 (HP-UX Release 10.30)



---

## **Preface**

### **Purpose**

The purpose of the *HP FTAM/9000 Programmer's Guide* is to introduce you to the File Transfer, Access and Management (FTAM) concepts you must know to write FTAM applications. Additionally, this guide provides example programs and program fragments to illustrate the use of FTAM services.

### **Audience**

The *HP FTAM/9000 Programmer's Guide* is for application programmers who want to learn to use FTAM and who are familiar with the C programming language.

### **Scope**

The *HP FTAM/9000 Programmer's Guide* explains basic FTAM concepts and thoroughly explains each FTAM data structure. This guide also describes the FTAM functions and their parameters.

HP FTAM/9000 runs on HP OTS/9000, an HP network product that provides a lower-level OSI protocol "stack," in conjunction with an 802.3 network link.

## Organization

- Chapter 1**     **HP FTAM/9000 Overview**This chapter provides an overview of the standards on which HP based this FTAM implementation, and the general concepts you should know before writing FTAM applications. Reading the conceptual information is vital to understanding the terminology used throughout the remainder of the manual.
- Chapter 2**     **Using HP FTAM/9000**This chapter describes how to start and stop the FTAM system and how to use FTAM regimes, functions, parameters, and libraries. Additionally, it provides general recommendations for programming with FTAM.
- Chapter 3**     **HP FTAM/9000 Data Structures**This chapter describes the structures used by FTAM. Additionally, the chapter describes header files, setting bits with defined constants, and basic data types.
- Chapter 4**     **Using Support Functions**This chapter describes the support functions used throughout most applications. For example, some support functions help you manage memory and translate error codes to printable character strings.
- Chapter 5**     **Using High Level, Context Free Functions**This chapter describes high level, context free (HLCF) functions. High level means you can use one function instead of several other functions (low level). Context free means they are not dependent on the use of other functions for their success.
- Chapter 6**     **Managing HP FTAM/9000 Connections**This chapter describes functions used to establish and release FTAM connections.
- Chapter 7**     **Managing and Accessing HP FTAM/9000 Files**This chapter describes the functions used to manage and access FTAM files after you established connections.



---

<b>Chapter 8</b>	<b>Transferring HP FTAM/9000 Data</b> This chapter describes the functions used to transfer data to and from FTAM files
<b>Chapter 9</b>	<b>Handling Errors</b> This chapter describes troubleshooting FTAM, interpreting errors, and logging error information.
<b>Chapter 10</b>	<b>Example Programs</b> This appendix provides complete examples of HLCF functions, creating FTAM connections, managing and accessing FTAM files, and transferring FTAM data. These programs also use the support functions throughout them.
<b>Chapter 11</b>	<b>Document Types and Constraint Sets</b> This appendix contains descriptions of the file representations supported by the various document types and constraint sets.
<b>Chapter 12</b>	<b>Character Sets</b> This appendix describes the available character sets for the HP-UX implementation of FTAM.
<b>Glossary</b>	The glossary defines terms used throughout the manual.
<b>Index</b>	The index lists information by topics so that you locate necessary information.

## Documentation Guide

<b>For More Information</b>	<b>Read</b>
Installing and Configuring HP FTAM/9000	<i>Installing and Administering HP FTAM/9000</i> (B1033-90034)
Troubleshooting HP FTAM/9000	<i>OSI Troubleshooting Guide</i> (32070-90020)
FTAM Programming	<i>HP FTAM/9000 Reference Manual</i> (B1033-90004)

---

FTAM Protocol Specifications	<i>ISO 8571, Information Processing Systems – Open Systems Interconnection – File Transfer, Access and Management</i>
International Standards ISO 8571	<i>ISO 8571, Information Processing Systems – Open Systems Interconnection – File Transfer, Access and Management</i>
MAP 3.0 Interface Specifications	<i>MAP 3.0 Application Interface Specification</i>
NBS Phase III	<i>Implementation Agreements for Open Systems Interconnection Protocols from the NBS Workshop for Implementors of Open Systems Interconnection</i>
ACSE	<i>ISO 8649, Information Processing Systems – Open Systems Interconnection – Service Definition for the Association Control Service Element</i>
ASN.1	<i>ISO 8824, Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)</i>
BER	<i>ISO 8825, Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)</i>

---

# Contents

## 1. HP FTAM/9000 Overview

Chapter Overview . . . . .	27
The HP-UX Implementation of FTAM . . . . .	28
Open Systems Interconnection (OSI) . . . . .	28
ISO/IS 8571 FTAM . . . . .	28
Manufacturing Automation Protocol (MAP) 3.0 . . . . .	29
NBS (NIST) Implementors Agreements . . . . .	29
Protocol Implementation Conformance Statement (PICS) . . . . .	30
Overview of FTAM Concepts . . . . .	31
The Virtual Filestore (VFS) . . . . .	31
FTAM Shadow Files . . . . .	33
Regimes . . . . .	34
FTAM Communication . . . . .	35
Initiators and Responders . . . . .	36
Application Entities (AEs) . . . . .	37
Synchronous and Asynchronous Operations . . . . .	38
File Attributes . . . . .	39
Document Types . . . . .	39
Constraint Sets . . . . .	40
Access Contexts . . . . .	40
FTAM File Structure Model . . . . .	41

## 2. Using HP FTAM/9000

Chapter Overview . . . . .	44
Starting and Stopping the FTAM System . . . . .	45
Generating FTAM Programs . . . . .	46
Header Files . . . . .	46
libmapftam.a Library . . . . .	46
lint(1) Library . . . . .	46

---

## Contents

Using Regimes . . . . .	47
Regime Guidelines . . . . .	47
Using the FTAM Regime . . . . .	48
Using the File Selection Regime . . . . .	49
Using the File Open Regime . . . . .	49
Using the Data Transfer Regime . . . . .	50
Using Functions . . . . .	51
High Level Services (HLS) and Low Level Services (LLS) . . . . .	51
Context Free (CF) and Context Sensitive (CS) Functions . . . . .	52
Available Functions . . . . .	52
Typical Applications . . . . .	59
Using Parameters . . . . .	60
Parameter Order . . . . .	60
Data Control Blocks . . . . .	61
input_dcb . . . . .	61
inout_dcb . . . . .	61
General Recommendations . . . . .	63
Handling Strings, HP-UX Lines, and FTAM-1 Lines . . . . .	64
<b>3. HP FTAM/9000 Data Structures</b>	
Chapter Overview . . . . .	67
Header Files . . . . .	68
Using Defined Constants To Set Bits . . . . .	69
EXAMPLE . . . . .	69
Basic Data Types . . . . .	70
Object_id . . . . .	70
Octet_string . . . . .	71

---

## Contents

Directory Services Data Structures . . . . .	72
Organizing Directory Services . . . . .	72
Using Directory Services . . . . .	74
Ae_dir_name . . . . .	75
Dir_dn . . . . .	76
EXAMPLE . . . . .	76
Dir_rdn . . . . .	77
EXAMPLE . . . . .	77
Dir_ava . . . . .	79
Setting Ae_dir_name Example . . . . .	81
Ae_label . . . . .	83
Ae_title . . . . .	84
Ae_title_option . . . . .	85
Api_rc . . . . .	87
Connection_id . . . . .	89
Ft_access_context . . . . .	90
Ft_access_control . . . . .	92
Ft_access_control_element . . . . .	94
Ft_account . . . . .	97
Ft_attribute_groups . . . . .	98
Ft_attribute_names . . . . .	101
Ft_attributes . . . . .	103
Ft_attribute_names . . . . .	105
Ft_attribute_values . . . . .	106
Ft_concurrency_control . . . . .	109
Ft_file_lock . . . . .	111
Rules for Ft_concurrency_control . . . . .	111

---

## Contents

Ft_contents_type_element.....	113
Ft_contents_type .....	114
Ft_contents_form .....	114
Ft_contents_info .....	115
Ft_document_type .....	115
Ft_data_unit.....	119
Ft_structure_id .....	120
Ft_data_element .....	121
Ft_node_descriptor .....	123
Rules for Ft_data_unit Linked Lists (FTAM-2 Only) .....	125
Ft_dcb_type.....	126
Ft_delete_action.....	128
Ft_delete_overwrite .....	129
Ft_diagnostic .....	130
Ft_diag_type .....	132
Ft_entity_ref .....	132
Ft_fadu_identity.....	134
Ft_fadu_form.....	134
Ft_fadu_info .....	135
Ft_fadu_operation .....	137
Ft_file_actions.....	138
Ft_file_passwords.....	139
Ft_file_status .....	141
Ft_filename.....	143
Ft_functional_units .....	144
Ft_initiator_identity.....	147

---

## Contents

Ft_processing_mode	148
Ft_qos	149
Ft_service_class	150
Ft_single_file_pw	155
inout_dcb	156
Ft_output	156
Ft_xxx_out_dcb	157
Memory Allocation for inout_dcbs	158
input_dcb	160
Local_event_name	162
P_address	164
<b>4. Using Support Functions</b>	
Chapter Overview	169
Managing Memory	170
ft_didcb()	170
ft_didcb() Parameters	171
ft_dfdbc()	171
ft_dfdbc() Parameters	172
em_fdmemory()	172
em_fdmemory() Parameters	173
ft_fdmemory()	173
ft_fdmemory() Parameters	174
Responding to Asynchronous Calls	175
When to Use em_wait()	175
When to Use em_hp_select()	176
When to Use em_hp_sigio()	176
Using em_wait()	177

---

## Contents

Using em_hp_select() .....	180
Waiting On FTAM and Non-FTAM Events .....	181
Handling Signal Interrupts.....	185
Using em_hp_sigio() .....	188
Translating Error Messages .....	196
em_gperror() .....	196
em_gperror() Parameters .....	197
ft_gperror() .....	198
ft_gperror() Parameters.....	199
Determining Available Resources .....	200
ft_nwcleared() .....	200
ft_nwcleared() Parameters .....	201
<b>5. Using High Level, Context Free Functions</b>	
Chapter Overview .....	205
Copying and Moving FTAM Files (HLCF) .....	206
ft_fcopy() .....	206
Ft_fcopy_in_dcb .....	207
Ft_fcopy_out_dcb .....	208
ft_fcopy() Parameters.....	208
ft_fmove().....	211
Ft_fmove_in_dcb .....	212
Ft_fmove_out_dcb .....	212
ft_fmove() Parameters .....	213
ft_fcopy_aet .....	216
Ft_fcopy_in_dcb .....	217
Ft_fcopy_out_dcb .....	218
ft_fcopy_aet() Parameters .....	218



---

## Contents

ft_fmove_aet() . . . . .	221
Ft_fmove_in_dcb . . . . .	222
Ft_fmove_out_dcb . . . . .	223
ft_fmove_aet() Parameters . . . . .	223
Reading and Changing Attributes (HLCF) . . . . .	226
ft_fattributes() . . . . .	226
Ft_fattributes_in_dcb . . . . .	227
Ft_fattributes_out_dcb . . . . .	227
ft_fattributes() Parameters . . . . .	228
ft_fcattributes() . . . . .	230
Ft_fcattributes_in_dcb . . . . .	231
Ft_fcattributes_out_dcb . . . . .	231
ft_fcattributes() Parameters . . . . .	231
ft_fattributes_aet() . . . . .	234
Ft_fattributes_in_dcb . . . . .	235
Ft_fattributes_out_dcb . . . . .	235
ft_fattributes_aet() Parameters . . . . .	235
ft_fcattributes_aet() . . . . .	238
Ft_fcattributes_in_dcb . . . . .	239
Ft_fcattributes_out_dcb . . . . .	239
ft_fcattributes_aet() Parameters . . . . .	240
Deleting Files (HLCF) . . . . .	242
ft_fdelete() . . . . .	242
Ft_fdelete_in_dcb . . . . .	242
Ft_fdelete_out_dcb . . . . .	242
ft_fdelete() Parameters . . . . .	243
ft_fdelete_aet() . . . . .	245
Ft_fdelete_in_dcb . . . . .	245
Ft_fdelete_out_dcb . . . . .	245
ft_fdelete_aet() Parameters . . . . .	246

---

# Contents

## 6. Managing HP FTAM/9000 Connections

Chapter Overview .....	251
Connection Establishment Process .....	252
Starting and Stopping Application Entities .....	254
ft_aeactivation() .....	254
Ft_aeactivation_in_dcb .....	254
Ft_output .....	255
ft_aeactivation() Parameters .....	255
ft_aedeactivation() .....	256
Ft_output .....	256
ft_aedeactivation() Parameters .....	256
Establishing and Removing Connections .....	257
ft_connect() .....	257
Ft_connect_in_dcb .....	258
Ft_connect_out_dcb .....	259
ft_connect() Parameters .....	259
ft_rrequest() .....	263
Ft_relreq_in_dcb .....	263
Ft_relreq_out_dcb .....	263
ft_rrequest() Parameters .....	264
ft_aereset() .....	264
Ft_output .....	264
ft_aereset() Parameters .....	265
Aborting Connections .....	266
ft_abort() .....	266
Ft_abort_in_dcb .....	266
Ft_output .....	266
ft_abort() Parameters .....	267

---

## Contents

ft_ireceive() .....	267
Ft_indication_out_dcb.....	268
ft_ireceive() Parameters .....	269

### 7. Managing and Accessing HP FTAM/9000 Files

Chapter Overview .....	273
Gaining Access to FTAM Files .....	274
ft_create() .....	274
Ft_create_in_dcb .....	275
Ft_create_out_dcb .....	275
ft_create() Parameters .....	276
ft_select().....	277
Ft_select_in_dcb .....	278
Ft_select_out_dcb .....	278
ft_select() Parameters.....	278
ft_open() (LLCS) .....	280
Ft_open_in_dcb .....	280
Ft_open_out_dcb .....	281
ft_open() Parameters.....	281
ft_close() (LLCS) .....	282
Ft_close_out_dcb .....	283
ft_close() Parameters.....	283
ft_delete() (LLCS) .....	284
Ft_delete_out_dcb .....	284
ft_delete() Parameters.....	285
ft_deselect().....	286
Ft_deselect_out_dcb .....	286
ft_deselect() Parameters .....	286

---

## Contents

Opening and Closing Files (HLCS) .....	288
ft_fopen() (HLCS) .....	288
Ft_fopen_in_dcb .....	289
Ft_fopen_out_dcb .....	290
ft_fopen() Parameters .....	290
ft_fclose() (HLCS) .....	292
Ft_fclose_out_dcb .....	292
ft_fclose() Parameters .....	293
Reading and Changing Attributes (LLCS) .....	294
ft_rattributes() (LLCS) .....	294
Ft_rattributes_in_dcb .....	295
Ft_rattributes_out_dcb .....	295
ft_rattributes() Parameters .....	295
ft_cattributes() (LLCS) .....	296
Ft_cattributes_in_dcb .....	297
Ft_cattributes_out_dcb .....	297
ft_cattributes() Parameters .....	298
Locating and Erasing FTAM Files .....	299
ft_locate() .....	299
Ft_locate_out_dcb .....	299
ft_locate() Parameters .....	300
ft_erase() .....	301
Ft_erase_out_dcb .....	301
ft_erase() Parameters .....	302
Grouping LLCS FTAM Functions .....	303
Allowable Groupings .....	303
ft_bgroup() .....	305
Ft_bgroup_out_dcb .....	305
ft_bgroup() Parameters .....	305

---

## Contents

ft_egroup() . . . . .	306
Ft_egroup_out_dcb . . . . .	306
ft_egroup() Parameters . . . . .	307

### 8. Transferring HP FTAM/9000 Data

Chapter Overview . . . . .	311
Sequence of Transferring Data . . . . .	312
Reading Data . . . . .	313
ft_read() . . . . .	313
Ft_read_out_dcb . . . . .	314
ft_read() Parameters . . . . .	314
ft_rdata() . . . . .	315
Ft_rdata_out_dcb . . . . .	315
ft_rdata() Parameters . . . . .	316
ft_rdataqos() . . . . .	317
Ft_rdataqos_out_dcb . . . . .	318
ft_rdataqos() Parameters . . . . .	318
Writing Data . . . . .	319
ft_write() . . . . .	319
Ft_write_out_dcb . . . . .	320
ft_write() Parameters . . . . .	321
ft_sdata() . . . . .	321
Ft_sdata_out_dcb . . . . .	322
ft_sdata() Parameters . . . . .	323
Ending Data Transfer . . . . .	324
ft_edata() . . . . .	324
Ft_edata_in_dcb . . . . .	324
Ft_edata_out_dcb . . . . .	324
ft_edata() Parameters . . . . .	325

---

## Contents

ft_ettransfer() . . . . .	326
Ft_ettransfer_out_dcb . . . . .	326
ft_ettransfer() Parameters . . . . .	327
ft_cancel() . . . . .	327
Ft_cancel_in_dcb . . . . .	328
Ft_cancel_out_dcb . . . . .	328
ft_cancel() Parameters . . . . .	329
ft_rcancel() . . . . .	330
Ft_rcancel_in_dcb . . . . .	330
Ft_rcancel_out_dcb . . . . .	330
ft_rcancel() Parameters . . . . .	331
<b>9. Handling Errors</b>	
Chapter Overview . . . . .	335
General Approach to Troubleshooting FTAM. . . . .	336
Interpreting Errors . . . . .	338
Function Return Values . . . . .	338
Output Errors . . . . .	339
MAP 3.0 FTAM Errors . . . . .	339
HP-UX-Specific Errors. . . . .	340
FPM and VFS Errors. . . . .	341
Example Program Checking for Errors. . . . .	344
Logging Errors . . . . .	346
Example Program Containing Error Information . . . . .	347
Using API Tracing . . . . .	349
API Tracing Variables. . . . .	349
Enabling API Tracing . . . . .	349
Interpreting the Trace File. . . . .	350

---

# Contents

## 10. Example Programs

Former Introduction ... need to change .....	354
Using HLCF Functions Example .....	355
Managing FTAM Connections Example.....	357
Using LLCs Functions Example .....	360
Setting Ae_dir_dn and P_address Utility Example.....	373
Checking for Errors Example .....	380
Common Code Example .....	382

## 11. Document Types and Constraint Sets

Document Types .....	430
FTAM-1 Document Type .....	431
FTAM-1 Document Semantics .....	432
FTAM-2 Document Type .....	433
FTAM-2 Document Semantics .....	435
FTAM-3 Document Type .....	436
FTAM-3 Document Semantics .....	437
INTAP-1 Document Type.....	438
INTAP-1 Document Semantics.....	439
NBS-9 Document Type.....	439
NBS-9 Document Semantics.....	440
Constraint Sets .....	442
Access Contexts .....	443
Unstructured Constraint Set.....	444
Sequential Flat Constraint Set .....	445

## 12. Character Sets

Former Intro ... Change.....	448
------------------------------	-----

---

## **Contents**





File Transfer, Access and Management (FTAM) is an international standard for transfer, access, and management of files in an open network. HP FTAM/9000 is an HP implementation of this standard; it offers the C language programmatic interface defined for MAP 3.0 FTAM, which in turn, provides you with the following capabilities.

- Multivendor connectivity
- Text and binary file transfers
- File access (record level locate and read)
- File management, including the following:
  - File creation and deletion
  - File attribute manipulation (read and change)
  - Security mechanisms such as file locking, filestore passwords, and concurrency control, and file access control

## Chapter Overview

This chapter is an overview of the HP-UX implementation of FTAM. The chapter discusses both the standards on which HP FTAM/9000 is based and the high-level concepts you should know before writing FTAM applications (user programs). The major topics are as follows.

- The HP-UX Implementation of FTAM
  - Open Systems Interconnection (OSI)
  - ISO/IS 8571 FTAM
  - Manufacturing Automation Protocol (MAP) 3.0
  - Implementors Agreements
  - Protocol Implementation Conformance Statement (PICS)
- Conceptual FTAM Overview
  - Virtual Filestore (VFS)
  - Regimes
  - FTAM Communication
    - Initiators and Responders
    - Application Entities (AEs)
    - Synchronous and Asynchronous Calls
  - File Attributes
  - Document Types
    - Constraints Sets
    - Access Contexts
  - FTAM File Structure Model

## **The HP-UX Implementation of FTAM**

The HP-UX implementation of FTAM is compliant with the following standards and specifications.

- Open Systems Interconnection (OSI) Model
- ISO/IS 8571 FTAM
- Manufacturing Automation Protocol (MAP) 3.0 Application Interface Specification
- NBS (NIST) Phase III Implementors Agreements

Additionally, HP implemented T1.3, T2.3, A1.3, and M1.3 profiles from the NBS (NIST) Phase III Implementors Agreements.

### **Open Systems Interconnection (OSI)**

The OSI network model defines a standardized network architecture that supports connection-oriented communication between devices for all vendors. Users of an OSI network can accurately anticipate how other network nodes behave. Once a connection is established with a remote system, exchange of data is guaranteed.

The International Standards Organization (ISO) created the OSI model to define the actions and services of seven communication functions known as the seven layers. Layer seven of this model is the Application layer (the layer on which your application program sits). The interface to the Application layer describes your view of the network; FTAM resides in this seventh layer. The activities below layer seven concern the internal workings of the OSI protocols and therefore, are of little concern to an application programmer.

### **ISO/IS 8571 FTAM**

The international standard (IS) ISO/IS 8571 defines FTAM. This standard specifies a virtual filestore, a set of services to manipulate that filestore, and a set of rules (protocols) that define how to use the services. The formal title of ISO/IS 8571 is “Information Processing Systems — Open Systems Interconnection File Transfer, Access and Management.”

## **Manufacturing Automation Protocol (MAP) 3.0**

The Manufacturing Automation Protocol (MAP) 3.0 Specification identifies standard protocols for each of the seven OSI layers. The MAP 3.0 Application Interface Specification defines the C-language programmatic interface to each Application service, including the HP-UX FTAM service.

### **NBS (NIST) Implementors Agreements**

The NBS (NIST) Implementors Agreements further define implementation limitations on ISO/IS 8571 FTAM. HP-UX FTAM supports the NBS Phase III minimum requirements, and implements many of the NBS options, based on perceived value and compatibility with HP-UX.

---

**NOTE**

There are a number of attributes that HP-UX FTAM does not implement. HP-UX FTAM accepts values for these options as input, but makes no local use of them. When these options are output, HP-UX FTAM provides zero (for integers) or “no value available” (for strings). For these options, this document often states that there is “no value available” or that HP-UX FTAM responders accept, but ignore the value.

---

If another implementation is not NBS Phase II or Phase III compliant, you may still be able to communicate with that implementation. Refer to the vendor's Protocol Implementation Conformance Statement (PICS) to determine how to set parameters.

## Protocol Implementation Conformance Statement (PICS)

The HP-UX FTAM Protocol Implementation Conformance Statement (PICS) defines the capabilities of HP-UX FTAM responders. You can use PIC Statements to determine if two implementations can communicate. Refer to the *HP FTAM/9000 Reference Manual* on how to obtain the HP-UX FTAM PICS.

HP implements the profiles A1, M1, T1, and T2. (The following references to document types, constraint sets, file access data units (FADUs), initiators, responders, and Virtual Filestore are described in the “Conceptual FTAM Overview” section.)

- A1** Provides for the transferring and accessing of files with Unstructured or Flat constraint sets. FTAM-1, FTAM-2, and FTAM-3 document types are supported. A1 supports reading from and writing to a file or a single FADU, locating within files, and erasing files.
- M1** Provides for an initiator's management of files within the Virtual Filestore, to which access is provided by the responder. Management includes creating and deleting a file, and reading and changing file attributes.
- T1** Provides for the transferring of entire files with an Unstructured constraint set. Both FTAM-1 and FTAM-3 document types are supported. T1 supports reading from or writing to an entire file.
- T2** Provides for the transferring of entire files with an Unstructured or Sequential Flat constraint set. FTAM-1, FTAM-2, and FTAM-3 document types are supported. T2 supports reading from or writing to a file or a single FADU.

## Overview of FTAM Concepts

This section takes a high-level approach to explaining concepts you should know before writing FTAM applications. This section also refers to specific FTAM functions. For detailed information regarding these concepts and functions, refer to the Table of Contents for specific chapters and sections.

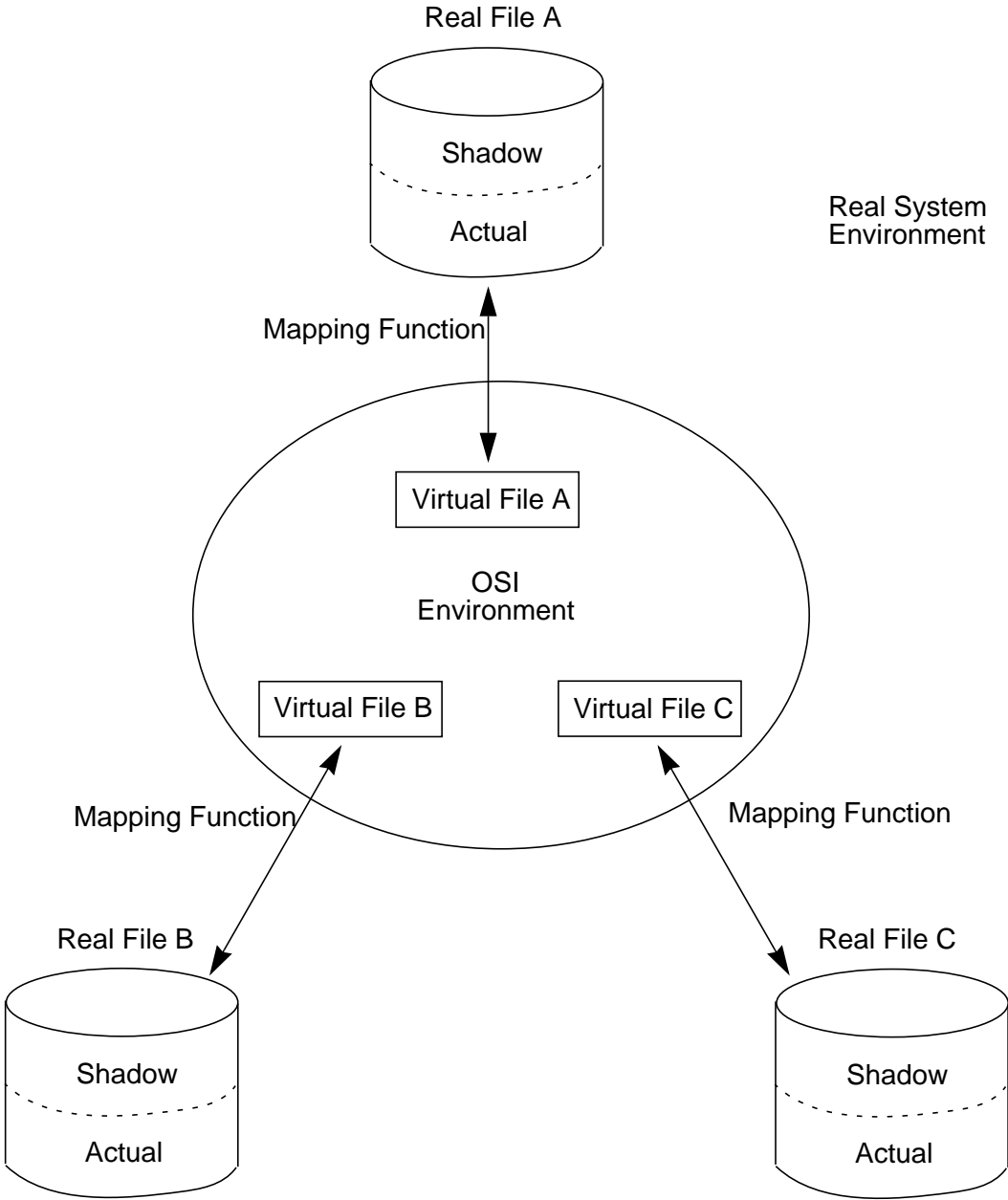
### The Virtual Filestore (VFS)

A real filestore uses a variety of mechanisms to structure, store, and retrieve real files. For example, files on HP-UX systems are stored and accessed as a stream of bytes; on other systems files might be stored and accessed as fixed- or variable-length records, or in other ways.

To conceal these storage and access differences, the FTAM specification defines a **Virtual Filestore**, or **VFS**. The VFS provides a uniform interface to all real file systems. It uses a system-independent model for describing files and their attributes. Though it represents real files, it masks the system-dependent differences in style and structure. (Refer to Figure 1-1.)

FTAM implementations map the VFS onto a real filestore. The mapping function (from the open system to the real system) absorbs the style and specification differences. You do not need to know the vendor, operating system, or file system with which you want to communicate. You need only know a file's complete name and where it is located. (However, you may need to read the other vendor's PIC statement.)

**Figure 1-1**      **HP-UX FTAM Virtual Filestore (VFS)**





## FTAM Shadow Files

Because the VFS is generalized, certain VFS concepts have no direct correspondence to the HP-UX file system. For example, a VFS file might have fixed-length records. Such records are not defined for the HP-UX file system.

To provide HP-UX files with FTAM VFS attributes, HP-UX uses two files: an actual file and an optional shadow file.

- The **actual file** contains the data for the file. The ownership and permissions for the file are related to, but distinct from, the FTAM attributes for the file.
- The **shadow file** contains only the FTAM VFS attributes for the file.

The actual and shadow files taken together make an “FTAM file.” The shadow file for an HP-UX file has the same name as the HP-UX file, prefixed by “\_.” For example, the FTAM file named myfile has a corresponding shadow file named `._myfile`. Like all files that begin with a period, FTAM shadow files can be listed with the following command:

```
$ ls -a
```

Certain FTAM attributes—such as document type—exist only in the shadow file. Other FTAM attributes—such as file name—are mapped to normal HP-UX file attributes. Some FTAM attributes—such as access control—include a combination of HP-UX file attributes and shadow file information.

The shadow file is not always necessary to read or delete a file. For these actions, FTAM uses default attributes. However, whenever FTAM creates or modifies a file, it always creates a corresponding shadow file.

---

**NOTE**

---

Other FTAM products from other vendors may implement the VFS using methods other than shadow files.

**Shadow File Cautions.** Use caution when employing HP-UX utilities with FTAM-related files. The following examples illustrate the kinds of issues that can arise:

- If you use `mv` to rename or relocate an FTAM-related file, the corresponding shadow file will not be rename or relocated. Subsequent FTAM access to the file will use default VFS attributes, which may or may not apply.

## Overview of FTAM Concepts

- If you use `cp` to create a new copy of an FTAM-related file, a corresponding shadow file will not be created. Subsequent FTAM access to the copy will use default VFS attributes, which may or may not apply.
- If you use `chmod` or `chown` to change the HP-UX permissions or ownership of an FTAM-related file, the new attributes may conflict with the attributes stored in the corresponding shadow file. Subsequent FTAM access to the file may have unexpected results or cause inconsistencies.
- If you use `rm` to remove an FTAM-related file, the corresponding shadow file will not be removed.

Shadow files that are inadvertently severed from their corresponding actual files remain as clutter. Furthermore, they could potentially interfere with future FTAM operations on files that have the same name as the original actual file.

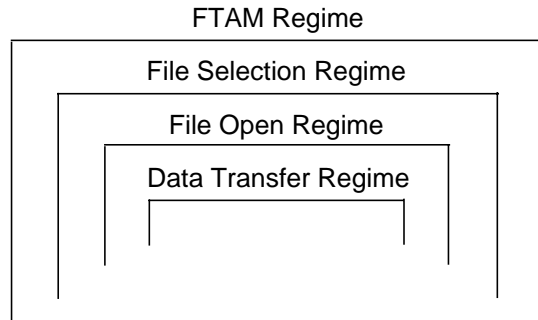
## Regimes

An FTAM transaction is built by calling FTAM functions in steps (or nested state sequences) called regimes. A regime is the period during which you can issue a specific set of FTAM functions. The regime determines the activities that can occur at a given time and therefore, creates the rules (protocols) for the FTAM state machine.

Regimes are nested, in the following order. Applications must pass through outer regimes to gain access to the inner ones. They must also exit the regimes in proper order, leaving the inner regimes to gain access to the outer ones.

- FTAM Regime
- File Selection Regime
- File Open Regime
- Data Transfer Regime

**Figure 1-2 FTAM Regime Order**



<b>FTAM Regime</b>	The FTAM regime requests FTAM services for the connection. The FTAM regime, also called the FTAM Association or FTAM Initialization regime, is the outermost regime.
<b>File Selection Regime</b>	The File Selection regime references (selects) a specific file. You can change and read attributes during this regime.
<b>File Open Regime</b>	During the File Open regime, a file's contents are available for access: reading, writing, locating, and erasing.
<b>Data Transfer Regime</b>	The Data Transfer regime is the innermost regime; it supports the transfer of actual data.

## **FTAM Communication**

The following sections describe the basic means of FTAM communication: initiators, responders, application entities, synchronous calls, asynchronous calls, high and low level functions, and context free and context sensitive functions.

## Initiators and Responders

As noted earlier, you write a program (user program) to access the FTAM interface using a library of functions. FTAM uses the initiator to make requests and the responder to service them. (Refer to Figure 1-3.)

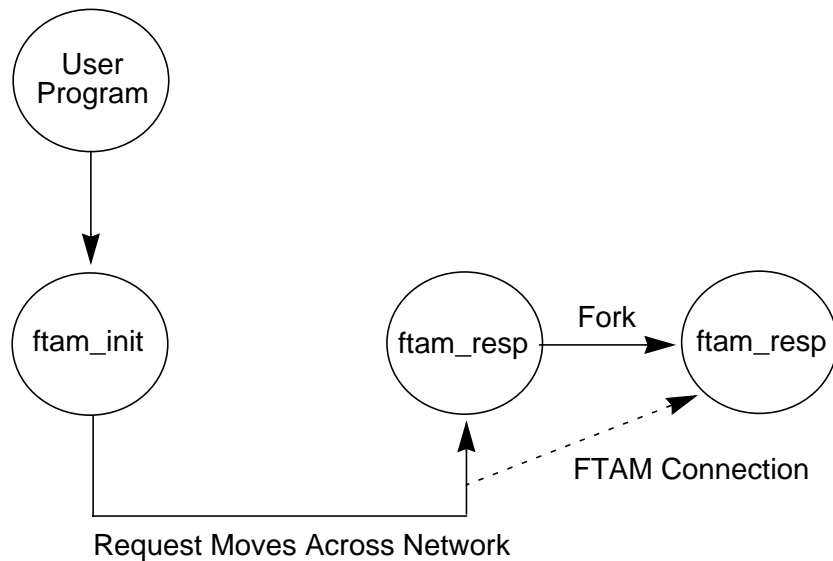
**Initiator** The initiator submits requests on behalf of you. When you submit a request (i.e., call a function), the initiator sends your request to a responder.

- The executable name for an HP-UX initiator is `ftam_init` (located in `/opt/ftam/sbin`). To use an FTAM function, you must link your application with the `libmapftam.a` library. The `ftam_init` process is also known as the initiator server provider process (SPP). Refer to the “Using FTAM” chapter for information on linking FTAM libraries.
- The `ftam_init` is invoked when you call `ft_aeactivation()`, and serves also as a responder for local FTAM requests. For information on `ft_aeactivation()`, refer to the “Managing FTAM Connections” chapter.
- The `ftam_init` is invoked when you call a high level function with the default `ae_label` (i.e., pointer to a NULL location). For information on high level functions, refer to the “Using FTAM” chapter.

**Responder** The responder receives and services an initiator's request. If applicable, it returns a response.

- The executable name for an HP-UX responder is `ftam_resp` (located in `/opt/ftam/sbin`). The `ftam_resp` is a daemon process that provides the interface for the HP-UX file system (real filestore) to the FTAM VFS. Note that the `ftam_resp` is also known as the responder SPP.
- One `ftam_resp` process exists for each active, remote FTAM connection. Connections to a local responder do not require an invocation of `ftam_resp`.
- An `ftam_resp` daemon process must be active to receive remote connection requests. After receiving a request, `ftam_resp` forks the process to create another `ftam_resp` process, which then processes all requests associated with that connection.

**Figure 1-3 FTAM Communication Process**



### **Application Entities (AEs)**

An **application entity (AE)** is a uniquely identified component of your application that associates it with the OSI communications network. An AE may be an FTAM initiator or responder.

Before locating, accessing, or manipulating FTAM files, you must establish communication (initialize an FTAM regime) with the AE (FTAM responder) that services the desired filestore.

Identify the AE by specifying either the entity's presentation address (P\_address) or its directory distinguished name (Ae\_dir\_name). These addresses are defined during system configuration and when nodes are added to the network. Consult your system administrator for the address information.

---

**NOTE**

---

Refer to the “FTAM Data Structures” chapter for information on the P\_address and Dir\_dn structures. Refer to the *Installing and Administering HP FTAM/9000* manual and the *OSI Troubleshooting Guide* for information on the presentation address and directory distinguished name configuration.

Once you establish the FTAM regime, you can access and manipulate FTAM files in that filestore. You can perform two types of file actions on the VFS: actions on complete files and file access actions. For example, you can create and delete a complete file or access a portion of a file's contents.

### **Synchronous and Asynchronous Operations**

FTAM functions use two modes of operation: synchronous and asynchronous:

**Synchronous Operations**

Synchronous function calls return only after the request is completely processed. Therefore, all output information is available when the call returns. Make synchronous function calls when the results must be processed in a specific sequence.

**Asynchronous Operations**

Asynchronous function calls return as soon as ftam\_init validates the request. Processing continues on the request. To determine whether the request returns successfully from the responder, you must call em\_wait(). Output information is available only after the em\_wait() function indicates that processing of the request has completed. Use asynchronous function calls when making multiple requests and when the order in which results are processed does not matter. You can perform other operations while an asynchronous request is being processed.

---

**NOTE**

---

Refer to the “Handling Errors” chapter for information on checking for errors on synchronous and asynchronous calls. Refer to the “Using Support Functions” chapter for information on using `em_wait()`.

## File Attributes

An FTAM file has two distinct parts: its attributes and its contents. Attributes describe the file (e.g., size, creator), and are either file attributes or activity attributes:

<b>File Attributes</b>	Define the file properties that are available to all initiators. File attributes are stored with the file or created with the file. File attributes are an inherent part of the file (e.g., size). Some file attributes identify structural aspects of the file contents.
<b>Activity Attributes</b>	Define the properties that exist only while a specific FTAM connection is being used. These attributes are not part of the file (e.g., <code>file_store_password</code> ).

## Document Types

An FTAM document type corresponds to a file type on most systems. In real filestores, common file types include stream text files, record-oriented text files, binary files, and directories which contain files. These file types are represented by the following FTAM document types:

- FTAM-1 Unstructured, stream-oriented text files
- FTAM-2 Record-oriented text files
- FTAM-3 Unstructured, stream-oriented binary files
- INTAP-1 Record files
- NBS-9 Directories, which contain files

For more information on document types, refer to the “Document Types and Constraint Sets” chapter. For information on setting document types, refer to the “FTAM Data Structures” chapter.

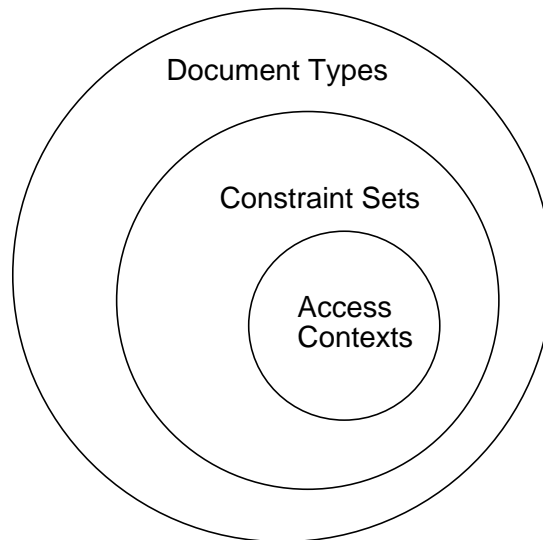
## Constraint Sets

Document types restrict the type of file by specifying **constraint sets**. Constraint sets specify the allowable file structures and the ways in which access actions can modify the structure. Constraint sets are either **Unstructured** or **Sequential Flat**. (Refer to Figure 1-5 and Figure 1-6 for file structure information of these two types.) For more information on constraint sets, refer to the “Document Types and Constraint Sets” chapter.

## Access Contexts

Constraint sets restrict the type of **access contexts** available. Access contexts specify how the information in the file is accessed. You specify access contexts only when reading a file.

**Figure 1-4** FTAM Document Types





## FTAM File Structure Model

The FTAM file model consists of a hierarchical structure (ordered tree) containing various levels.

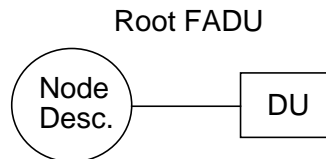
- Each node in the tree contains a **node descriptor**. Node descriptors reference specific parts of a file. However, a node descriptor may be empty.
- Each node in the tree contains a **data unit (DU)**. This data unit contains the actual file contents.

A **file access data unit (FADU)** is the smallest unit you can access in an FTAM file. A FADU is the complete information content of a subtree (i.e., node descriptors and data units). FADUs may be nested and may contain several data units.

The two file structures described by the Unstructured and Sequential Flat constraint sets are identified in the following illustrations.

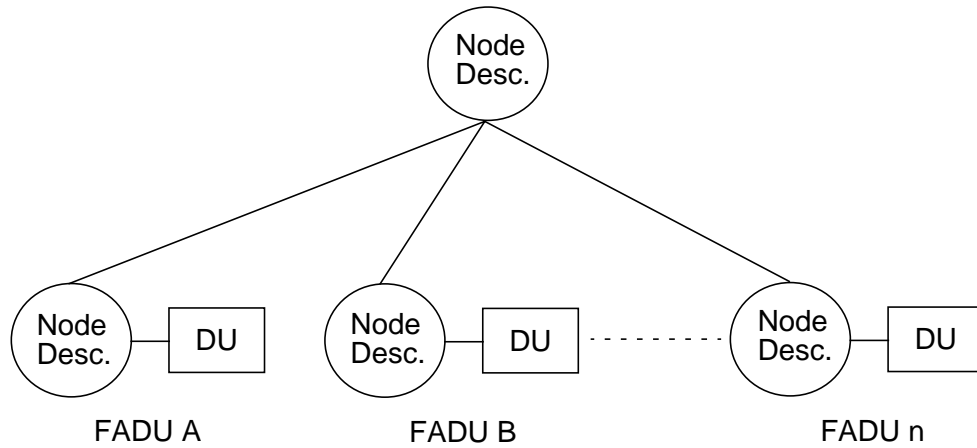
- An Unstructured constraint set contains one FADU consisting of one node descriptor and one data unit.

**Figure 1-5 FADU Defined by Unstructured Constraint Set**



- A Sequential Flat constraint set contains a root FADU with an unlimited number of FADUs one level beneath it.

**Figure 1-6 FADU Defined by Sequential Flat Constraint Set**



---

## **2** **Using HP FTAM/9000**

Before writing HP FTAM/9000 applications, you should understand FTAM libraries, regimes, functions, and parameters.

---

## Chapter Overview

This chapter describes how to start and stop FTAM and explains information you should know before writing applications. The primary sections are as follows.

- Starting and Stopping the FTAM System
- Generating FTAM Programs
- Using Regimes
- Using Functions
- Using Parameters
- General Recommendations

---

### NOTE

This chapter refers to specific functions. For detailed information on a function, refer to the Table of Contents for the correct chapter.

---

---

## Starting and Stopping the FTAM System

The HP OSI Transport Services (OTS/9000) startup procedure automatically starts the `ftam_resp` daemon (HP FTAM responder). You may occasionally need to start or stop only the FTAM portion of the system.

---

### NOTE

Bringing down the FTAM portion of the HP OSI network system is destructive and abruptly terminates any activity using the VFS associated with the local responder.

Using the shutdown option of `ftam_resp` does not kill user programs or `ftam_init` processes, though they will encounter unexpected errors when you bring down the system. You must be superuser to shutdown the FTAM responder, as follows.

```
$ /opt/ftam/lbin/ftam_resp -shutdown
```

To re-start only the FTAM portion of the HP OSI network system, first ensure the system is running. Also, execute the `ps(1)` command to ensure that no FTAM responder daemons (`ftam_resp`) are running; if they are, execute the above shutdown command. You must be superuser to re-start `ftam_resp`, as follows.

```
$ /opt/ftam/lbin/ftam_resp
```

## Generating FTAM Programs

Before using FTAM, you must include header files and link your programs with the `libmapftam.a` and `libmap.a` libraries. Another important library is the `lint(1)` library used for detecting bugs.

### Header Files

In your `*.c` files (source files), you must have the following lines at the beginning of your program to include the appropriate header files.

```
#include %</opt/ftam/include/map.h>  
#include %</opt/ftam/include/mapftam.h>
```

### `libmapftam.a` Library

To access FTAM services, link your application programs with the `libmapftam.a` library. Compile your programs by executing the following command. For information on using the `cc` command, refer to `cc(1)` in the HP-UX Reference Pages.

```
cc [options] -o %<executable> %<files to compile> -lmapftam -lmap -lint1 [additional  
libraries]
```

### `lint(1)` Library

A `lint(1)` library is available for use with the HP-UX programmatic interface to FTAM. Using `lint(1)` saves time by eliminating costly debugging effort. Although `lint(1)` does not detect all bugs, it can find some that general testing may not detect. To invoke `lint(1)`, enter the following command.

```
$ lint -ux filename.c -lmapftam -lmap
```

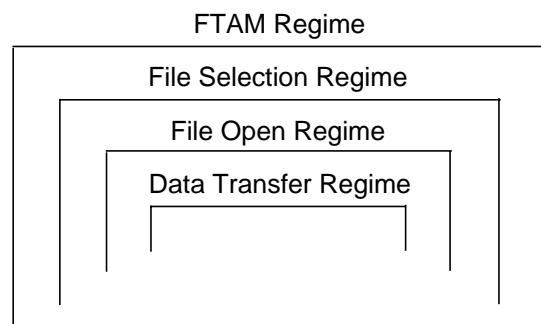
The `-ux` flags are optional; they remove some unnecessary errors about variables that are used, but not defined or some errors that are defined, but not used. For more information, refer to `lint(1)` in the HP-UX Reference Pages.

---

## Using Regimes

Regimes define which functions an FTAM application can use at any time. As an application descends the regimes, you can use specific FTAM functions as described in the following figure and sections. The regime order is as follows.

**Figure 2-1** FTAM Regime Order



- FTAM Regime
- File Selection Regime
- File Open Regime
- Data Transfer Regime

If you use high-level FTAM functions, you do not need to deal with regimes; FTAM takes care of these details.

## Regime Guidelines

Refer to the following guidelines for working in regimes.

- You must enter the outer regimes before entering the inner regimes. For example, you must enter the File Open regime before entering the Data Transfer regime.
- You must exit inner regimes before you can exit outer regimes. For example, you must exit the File Open regime before exiting the File Selection regime.

Using HP FTAM/9000

## Using Regimes

- After exiting an inner regime, you can re-enter the same regime (e.g., you can sequentially select several files in the FTAM regime).
- You can terminate all regimes, no matter how deeply nested, using the abort request (`ft_abort()`).

## Using the FTAM Regime

The FTAM functions requested during this regime are negotiated between the initiator and responder during the protocol exchange. (The initiator supplies information which the responder then verifies.)

Your FTAM application must supply the initiator identity. You may also request (i.e., try to negotiate) certain document types (file types), attribute groups, service classes, and functional units. Refer to the “FTAM Data Structures” chapter for detailed information regarding these negotiated services.

<b>Document Types</b>	File types FTAM-1, FTAM-2, FTAM-3, INTAP-1, and NBS-9. FTAM-1 is a text file, FTAM-2 is a record-oriented text file, FTAM-3 is a binary file, INTAP-1 is a record-oriented binary file, and NBS-9 is a directory.
<b>Attribute Groups</b>	Organize specific qualities or characteristics to a file (e.g., name and size); identify subsets of attributes.
<b>Service Classes</b>	Organize functional units into meaningful groups. Each service class identifies mandatory and optional functional units.
<b>Functional Units</b>	Specify which FTAM functions, within the selected service class, are allowed. The mandatory and optional functional units depend on the service class, which must support all selected functional unit values.

The responder verifies requests in one of the following ways.

- Returns a positive confirmation that it accepts the request with no changes.



- On `ft_connect()`, returns a positive confirmation that it accepts the request with the given (returned) changes to the initiator's request.
- Returns a negative confirmation that it does not accept the request.

<b>Entering the FTAM Regime</b>	<b>Functions Used Within the FTAM Regime</b>	<b>Exiting the FTAM Regime</b>
Invoke <code>ft_connect()</code> to establish a connection	<code>ft_select()</code> moves you to the File Selection regime and selects a file  <code>ft_create()</code> moves you to the File Selection regime and creates, then selects a file	Invoke <code>ft_rrequest()</code> to release the connection and move you out of the FTAM regime. (Note: You will no longer be in a regime, but <code>ftam_init</code> will still be activated.)

### Using the File Selection Regime

Enter the File Selection regime to reference a specific file. You must specify a filename. You cannot access a file's contents in the File Selection regime; you must enter the File Open regime to do so.

<b>Entering the File Selection Regime</b>	<b>Functions Used Within the File Selection Regime</b>	<b>Exiting the File Selection Regime</b>
Invoke <code>ft_create()</code> to create a new file <b>OR</b> Invoke <code>ft_select()</code> to select an existing file	<code>ft_rattributes()</code> reads file attributes  <code>ft_cattributes ()</code> changes file attributes  <code>ft_open()</code> moves you to the File Open regime and opens a file	Invoke <code>ft_delete()</code> to move to the FTAM regime and permanently remove the file <b>OR</b> Invoke <code>ft_deselect()</code> to move back to the FTAM regime and deselect (but not remove) the file

### Using the File Open Regime

The File Open regime makes the file's contents available for reading and writing.

**Entering the File Open Regime**

Invoke `ft_open()` to access a file's contents

**Functions Used Within the File Open Regime**

`ft_locate()` locates a specific place (FADU) in a file (FTAM-2 files only)  
`ft_erase()` completely erases an entire file  
`ft_read()` enters the Data Transfer regime and requests data transfer from an FTAM file  
`ft_write()` enters the Data Transfer regime and requests data transfer to an FTAM file

**Exiting the File Open Regime**

Invoke `ft_close()` to move back to the File Selection regime and close the file to further access

**Using the Data Transfer Regime**

The Data Transfer regime is the innermost regime; it supports the transfer of actual data.

**Entering the Data Transfer Regime**

Invoke `ft_read()` to request the transfer of data from an FTAM file  
**OR**  
Invoke `ft_write()` to request the transfer of data to an FTAM file

**Functions Used Within the Data Transfer Regime**

`ft_sdata()` sends (writes) data  
`ft_rdata()` receives (reads) data  
`ft_edata()` ends the sending of data

**Exiting the Data Transfer Regime**

Invoke `ft_ettransfer()` to move back to the File Open regime and end data transfer  
**OR**  
Invoke `ft_cancel()` to move back to the File Open regime and cancel the transfer of data  
**OR**  
If you have received a cancellation indication, invoke `ft_rcancel()` to acknowledge the indication and move back to the File Open regime

## Using Functions

FTAM functions may be high level or low level and may be context free or context sensitive. Refer to the following sections for information on these function types and for a list of available functions.

### High Level Services (HLS) and Low Level Services (LLS)

FTAM functions are either a high level service (HLS) or a low level service (LLS). Both HLS and LLS perform the following operations.

- Ensure the input and output data are provided
- Ensure all mandatory parameters have valid values
- Ensure all optionally supplied parameters have valid values
- Provide both synchronous and asynchronous calling modes

**Low Level Service**

An LLS is a function that provides the lowest level of service for the FTAM application interface.

**High Level Service**

An HLS is a function that performs a sequence of LLS functions, thereby eliminating the need to make the LLS calls individually. For example, the HLS `ft_copy()` copies one file to another by performing all the necessary steps to do so. The `ft_copy()` function establishes connections with the source and destination nodes, opens and closes the file, reads and writes data, and terminates the regimes.

## Context Free (CF) and Context Sensitive (CS) Functions

The term context refers to a dependency on the use of other functions.

**Context Free** A function that is context free does not depend on prior use of other FTAM functions. You can call a context free function at any time.

All context free functions are high level services: `ft_fcopy()`, `ft_fmove()`, `ft_frattributes()`, `ft_fcattributes()`, and `ft_fdelete()`.

**Context Sensitive** A function that is context sensitive depends on prior use of other FTAM functions. You must call context sensitive functions in the correct sequence as defined by the FTAM protocol.

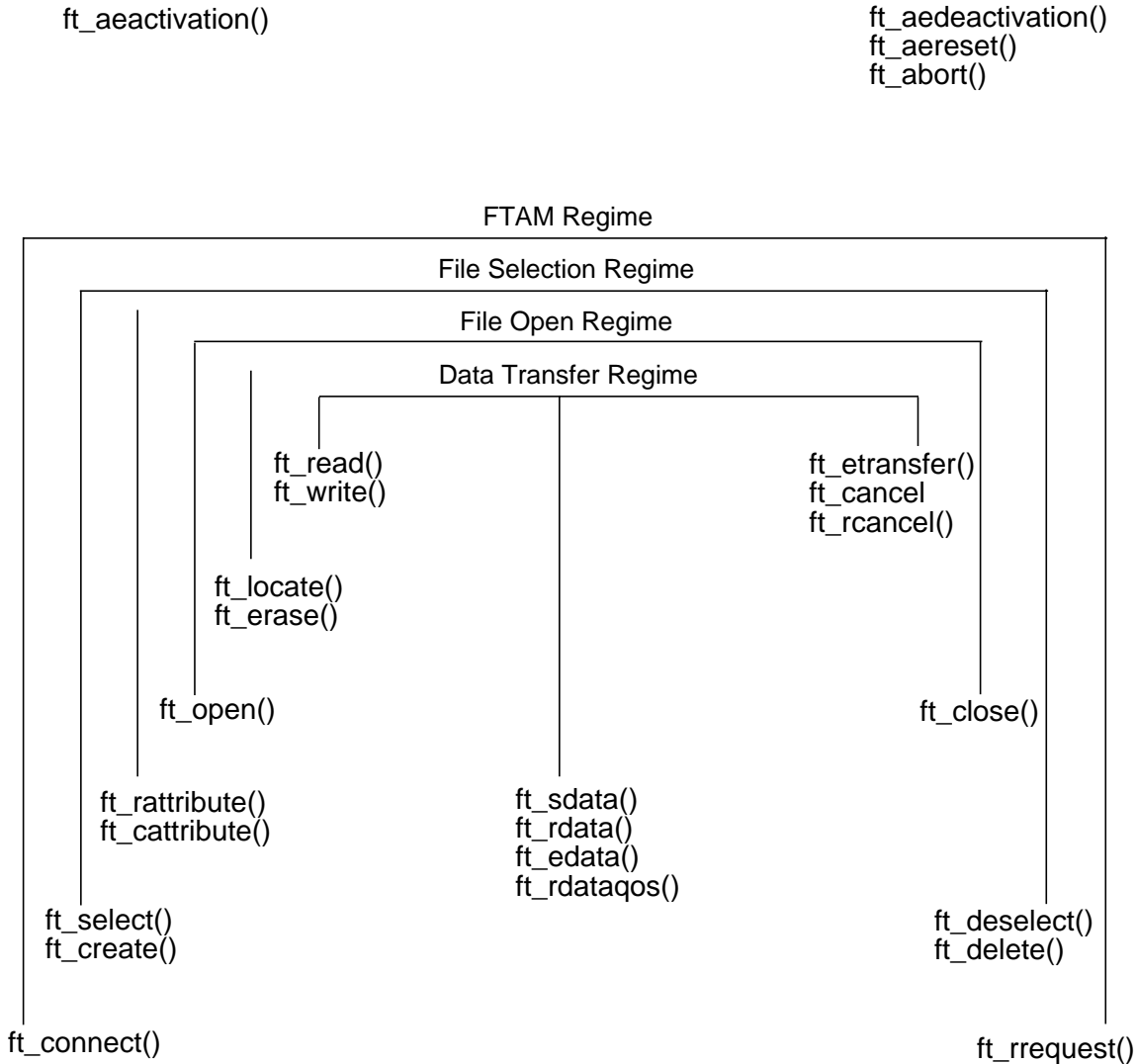
All context sensitive functions are low level services except for `ft_fopen()` and `ft_fclose()`.

## Available Functions

Figure 2-2 shows the low level FTAM functions available per regime. For example, you call `ft_select()` or `ft_create()` in the FTAM regime to move to the File Selection regime, where you can call `ft_rattributes()`, `ft_cattributes()`, and `ft_open()`. To move out of the File Selection regime, you call `ft_deselect()` or `ft_delete()`.

Note that you do not have to be in a regime to call those functions shown outside the regimes.

**Figure 2-2 Available FTAM Functions Per Regime**



**Using Functions**

The following table briefly describes the FTAM functions.

<b>Function</b>	<b>Description</b>
em_fdmemory()	Free dynamic memory allocated by em_gperror()
em_gperror()	Translate em_fdmemory(), em_gperror(), and em_wait() errors (return_codes) to character strings
em_wait()	Notify you that an asynchronous call completed
ft_abort()	Abort a connection Low level, context sensitive
ft_aeactivation()	Activate ftam_init Low level, context free
ft_aedeactivation()	Deactivate ftam_init Low level, context sensitive
ft_aereset()	Reset ftam_init Low level, context sensitive
ft_bgroup()	Designate the beginning of a grouped set of functions Low level, context sensitive
ft_cancel()	Cancel data transfer in progress Low level, context sensitive
ft_cattributes()	Change file attributes Low level, context sensitive
ft_close()	Close a file Low level, context sensitive
ft_connect()	Establish a connection from ftam_init to a responder Low level, context sensitive

<b>Function</b>	<b>Description</b>
ft_create()	Create a file Low level, context sensitive
ft_delete()	Delete the currently selected file Low level, context sensitive
ft_deselect()	Deselect the currently selected file Low level, context sensitive
ft_dfpcb()	Dynamically free FTAM data control block Context sensitive
ft_didcb()	Dynamically initialize FTAM data control block Context free
ft_edata()	End a series of data units Low level, context sensitive
ft_egrp()	Designate the end of a set of grouped functions Low level, context sensitive
ft_erase()	Erase all or part of a file Low level, context sensitive
ft_etransfer()	End the transfer of data Low level, context sensitive
ft_fattributes()	Select and change file attributes, and then deselect the file High level, context free
ft_fattributes_aet()	Select and change file attributes, and then deselect the file High level, context free
ft_fclose()	Close and then deselect or delete a file High level, context sensitive
ft_fcopy()	Copy a file High level, context free
ft_fcopy_aet()	Copy a file High level, context free

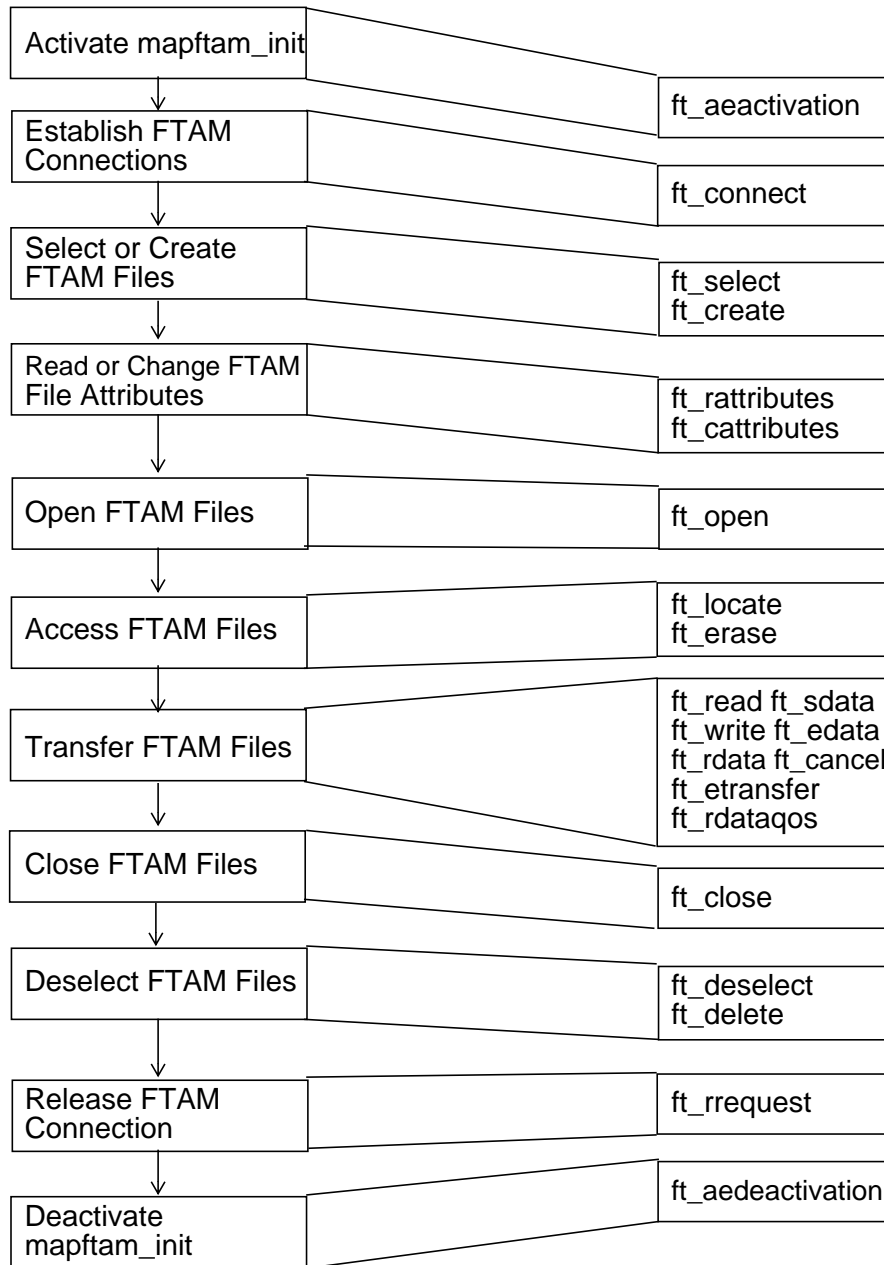
Using HP FTAM/9000  
Using Functions

<b>Function</b>	<b>Description</b>
<code>ft_fdelete()</code>	Select and delete a file High level, context free
<code>ft_fdelete_aet()</code>	Select and delete a file High level, context free
<code>ft_fdmemory()</code>	Free dynamic memory allocated by <code>ft_gperror()</code> Low level, context sensitive
<code>ft_fmove()</code>	Move a file High level, context free
<code>ft_fmove_aet()</code>	Move a file High level, context free
<code>ft_fopen()</code>	Select or create a file, and then open it High level, context sensitive
<code>ft_fattributes()</code>	Select and read file attributes, and then deselect the file High level, context free
<code>ft_fattributes_aet()</code>	Select and read file attributes, and then deselect the file High level, context free
<code>ft_gperror()</code>	Translate returned FTAM error codes to character strings Low level, context sensitive
<code>ft_ireceive()</code>	Receive an abort indication Low level, context sensitive
<code>ft_locate()</code>	Locate a specific part of an FTAM-2 file Low level, context sensitive
<code>ft_nwcleared()</code>	Note when an outstanding resource is cleared Low level, context sensitive
<code>ft_open()</code>	Open a file Low level, context sensitive
<code>ft_rattributes()</code>	Read file attributes Low level, context sensitive



<b>Function</b>	<b>Description</b>
ft_rcancel()	Either acknowledge the end of a data transfer in progress <b>OR</b> Respond to a cancel indication Low level, context sensitive
ft_rdata()	Receive a block of data Low level, context sensitive
ft_rdataqos()	Request transfer of data from a file Low level, context sensitive
ft_read()	Request transfer of data from a file Low level, context sensitive
ft_rrequest()	Release a connection Low level, context sensitive
ft_sdata()	Send a block of data Low level, context sensitive
ft_select()	Select a file Low level, context sensitive
ft_write()	Request transfer of data to a file Low level, context sensitive

**Figure 2-3 Example Use of Low Level FTAM Functions**



## Typical Applications

Figure 2-3 on the previous page is an example series of low level FTAM functions used within an application.

- The first column contains major segments (tasks) of a typical FTAM application.
- The second column contains typical low level functions used to perform the application tasks. You will probably not use all the functions in each box for every application.

## Using Parameters

This section discusses parameters to FTAM functions, focussing on the `input_dcb` and `inout_dcb` parameters. For information on specific parameters, determine the function using the parameter; then use the Table of Contents to locate the chapter explaining that function.

Parameters are the mechanisms for exchanging information between your application and the FTAM interface. Parameters may be any of the following types:

- Mandatory or Optional
- Input to the function, Output from the function
- Exposed (in the function's parameter list) or Internal (as a component of a data structure called a DCB in the parameter list)

## Parameter Order

Parameters to FTAM functions are organized as follows:

- Input parameters precede output parameters.
- Exposed input parameters precede input DCB parameters.
- Exposed output parameters follow output DCB parameters.

---

**NOTE**

Exposed parameters take precedence over the `input_dcb` parameters. For example, if you specify both the exposed parameter `filename` and the `input_dcb->attributes.values.filename` on the `ft_select()` function, the exposed `filename` is the one used to select the file.

---

## Data Control Blocks

Data control blocks (DCBs) are C structures used as function parameters to pass information between your application and the FTAM interface.

- `input_dcb` Contains only function input parameters.
- `inout_dcb` Contains parameters used for both input and output, and parameters used for output only.

Most FTAM functions use both an `input_dcb` and an `inout_dcb`.

### **input\_dcb**

The values used within `input_dcb` depend on the function requirements. If any of the `input_dcb` parameters are mandatory, you must pass a non-NULL address of an `input_dcb`; otherwise, you can pass a NULL value.

You can allocate memory for the `input_dcb` two ways.

- Allocate the memory yourself (e.g., using `malloc()` or by using variables).
- Before the request, invoke `ft_didcb()` to allocate memory for `input_dcb`. After the request completes, invoke `ft_dfdcb()` to free the memory used by `input_dcb`.

After the function call returns, you can immediately reclaim or reuse memory occupied by `input_dcb`.

### **inout\_dcb**

Use the `inout_dcb` to obtain output information in the user program. When passing the `inout_dcb` to the interface, you must pass an address that references either a non-NULL or NULL value.

Except for size, all parameters in the `inout_dcb` are output parameters. Note, however, some output parameters are exposed rather than being part of the `inout_dcb` (e.g., `connection_id` on the `ft_connect()` function call).

The size and result fields are always present in the `inout_dcb`. Other `inout_dcb` fields may be present, depending on the function.

### Using Parameters

For asynchronous calls, you should not access the `inout_dcb` memory from the time the asynchronous call returns `SUCCESS` until `em_wait()` verifies completion of the request; otherwise, the memory occupied by the `inout_dcb` contains random data.

You can allocate memory for `inout_dcb` in three ways.

1. The recommended method is to have the `inout_dcb` address reference a `NULL` value when first making the call.
  - By doing so, the FTAM interface allocates the memory necessary; thus, you avoid having to anticipate how much memory is required.
  - After the FTAM function returns (for asynchronous calls, after `em_wait()` returns `SUCCESS`), you must call `ft_dfdbc()` to free the memory.

---

**NOTE**

The next two methods described require you to manually allocate memory, and they carry some risk. It can be hard to predict how much memory to allocate, and failure to allocate enough can cause a run-time error. Consequently, HP encourages programmers to allow the FTAM interface to allocate DCB memory. Use one of these last two methods only if you have high performance requirements that are seriously compromised by the allocation and deallocation time in the first method.

2. Allocate the memory yourself (e.g., using `malloc()` or using variables). *Ensure the size is large enough to hold the `inout_dcb` structure and all data it may reference.* This memory should include a diagnostic list that may contain from 0 to 12 elements. You can then pass the address of the allocated space to the interface.
3. Before the request, invoke `ft_didcb()` to allocate memory for `inout_dcb`. Ensure the `additional_size` parameter is large enough to hold the `inout_dcb` structure and all data it may reference. After `em_wait()` returns `SUCCESS`, invoke `ft_dfdbc()` to free the memory used by `inout_dcb`.

## General Recommendations

Observe the following recommendations for programming with FTAM:

- FTAM functions are not re-entrant. Do not call any FTAM function within a signal handler.
- The `inout_dcb` pointer should reference a NULL value on input. This way, you do not have to anticipate the necessary output memory; the FTAM interface will allocate it for you. (Remember to later free the memory using `ft_dfcb()`.)
- You are guaranteed a minimum of 10 local to local (i.e., loopback) connections. You might have a maximum of 50 local to local connections, depending on the packet sizes.
- To make troubleshooting easier, always check for errors and record the following information:
  - Loop counters and other program-state information.
  - Specific function call that failed
  - Input parameters passed to the function
  - Function return value
  - `inout_dcb->result.return_code`
  - `inout_dcb->result.vendor_code`
  - Log instance (upper 16 bits of `vendor_code`)
  - `inout_dcb->diagnostic.error_id`
  - `inout_dcb->diagnostic.further_details`
  - `return_string` and `vendor_string` (from `ft_gperror()` or `em_gperror()`)

## Handling Strings, HP-UX Lines, and FTAM-1 Lines

---

**NOTE**

---

This section applies to FTAM-1 files only and to strings as they are used within low level FTAM calls.

This section clarifies the meaning of a **string**, **HP-UX line**, and **FTAM-1 line**. When transferring data with low level calls, you must understand FTAM-1 lines. If you are reading or writing local HP-UX files using low level calls, you must understand how to convert between HP-UX lines and FTAM lines.

String	A string is data within or pointed to by the primitive field of type struct Ft_data_element. A string may contain multiple or partial FTAM lines.
HP-UX Line	HP-UX lines are terminated by LINE FEED %<LF> characters (%<LF> is '\n' or '\012').
FTAM-1 Line	FTAM-1 lines are terminated by a combination of CARRIAGE-RETURN/LINE-FEED %<CR,LF> (%<CR> is '\r' or '\015').

Since HP-UX uses %<LF> to represent new lines, you must convert each %<LF> to the combination %<CR,LF> when writing to FTAM-1 files. For example, if your application reads an HP-UX file and subsequently sends the data as an FTAM-1 file (using low level FTAM calls), you must convert all instances of %<LF> in the data to %<CR,LF> before sending it.

- Any FTAM data element you receive uses %<CR,LF> to indicate a new line. Consequently, to write such data to an HP-UX file without using FTAM, you must write a %<LF> for every %<CR,LF> combination.
- Any FTAM data element you send must use %<CR,LF> to indicate a new line. A single %<LF> character is treated as data.

If you did not receive data directly from an FTAM function call and if you are using FTAM low level calls (e.g., ft\_sdata()), you must insert %<CR,LF> characters to delimit lines; you will receive them back in ft\_rdata() output.



---

## **3 HP FTAM/9000 Data Structures**

HP FTAM/9000 functions use data structures to handle information consistently and to pass information between cooperating applications.

---

**NOTE**

---

Your understanding of FTAM data structures is vital to your success in creating FTAM applications.

## Chapter Overview

This chapter describes the structures used for Directory Services, Event Management functions (e.g., `em_wait()`, `em_gperror()`, and `em_fdmemory()`), and FTAM. Additionally, the chapter describes header files, setting bits with defined constants, and basic data types.

## Header Files

Two header files, `map.h` and `mapftam.h`, contain the necessary FTAM structures, constants, and type definitions. Complete header files are available online in `/opt/ftam/include`.

`map.h` This header file contains definitions for the constants and structures shared by MAP-based OSI services and those used the Event Management functions `em_wait()`, `em_fdmemory()`, and `em_gperror()`.

To communicate with remote systems, you must use the `map.h` structures to specify either directory distinguished names (`Ae_dir_name`) or presentation addresses (`P_address`). Refer to the Directory Services section for structure information.

`mapftam.h` This header file contains the definitions for constants, structures, and enumerations specific to FTAM.

---

## Using Defined Constants To Set Bits

Certain structures require that you use defined constants to set bits to request specific items like attributes, actions, and concurrency control. You request an item by setting a bit to ON (1). Refer to the following table for a list of defined constants and their corresponding structures.

Defined Constants	Corresponding Structures
FT_FA_XXX	Ft_file_actions
FT_AG_XXX	Ft_attribute_groups
FT_AN_XXX	Ft_attribute_names
FT_AC_XXX	Ft_access_control_element
FT_PA_XXX	Ft_permitted_actions
FT_FT_XXX	Ft_functional_units
FT_PM_XXX	Ft_processing_mode
FT_SC_XXX	Ft_service_class

- Use the defined constant to set the bits to either ON (1) or OFF (0). Setting bits other than those listed results in an error.
- If needed, set the bit by first defining a variable. Use the appropriate defined constant to set the bit you want.

### EXAMPLE

This example sets the filename and contents\_type attributes in the variable called attribute\_names.

```
Ft_attribute_names      attribute_names;  
attribute_names = (FT_AN_FILENAME | FT_AN_CONTENT_TYPE);
```

## Basic Data Types

The FTAM interface uses the following data types. These basic types are unambiguous typedef synonyms for well-known intrinsic C data types.

Data Type	Description	Corresponding C Data Type
Uint8	8-bit unsigned integer	unsigned character
Unit16	16-bit unsigned integer	unsigned character
Uint32	32-bit unsigned integer	unsigned long
Sint8	8-bit signed integer	char
Sint16	16-bit signed integer	short
Sint32	32-bit signed integer	long
Bool	Logical binary	unsigned character
Octet	8 bits of unformatted data	unsigned character

## Object\_id

```
struct Object_id
{
    Uint16    length;
    Sint32    *element; /* List of integers */
};
```

The `Object_id` is a widely used `map.h` structure containing the following fields.

<code>length</code>	Number of integers in the <code>*element</code> array.
<code>*element</code>	Pointer to an array of integers.

## Octet\_string

```
struct Octet_string
{
    Uint32      length;
    Octet_pointer pointer; /* List of Octets */
};
```

The `Octet_string` is a widely used `map.h` structure containing the following fields.

<code>length</code>	Number of Octets in the pointer array.
<code>pointer</code>	Pointer to a array of Octets.

## Directory Services Data Structures

Directory Services holds and provides access to information about objects (generally existing in telecommunications and information processing). Directory Services stores information in an area collectively known as the Directory Information Base (DIB).

FTAM uses Directory Services to identify responders. One analogy is that Directory Services is like a telephone directory that lists responders.

For communicating with other systems and processes, you need to know how to identify yourself and the systems and processes with which you want to communicate. FTAM uses Directory Services to obtain the presentation addresses (called `presentation_address`) for such identification purposes. To obtain these addresses, Directory Services uses the directory distinguished names (called `dir_name`, `source_dirname`, `destination_dirname`, and `dirname`) that you provide.

For information on configuring your directory distinguished names and presentation addresses for the OTS/9000 IEEE 802.3 link, refer to the *OSI Troubleshooting Guide*.

## Organizing Directory Services

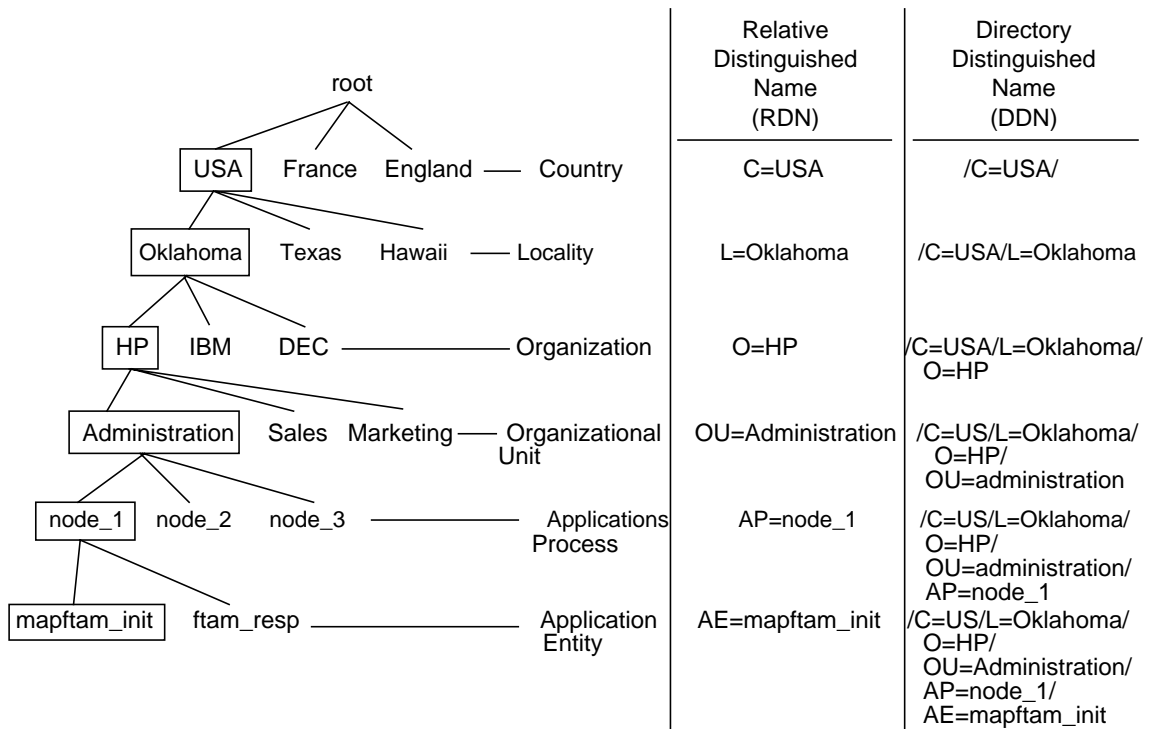
The Directory Service organization starts at a root entry that is the base for all entries. The following list provides possible entries in the required order.

- Country
- Locality
- Organization
- Organizational Unit
- Application Process (AP)
- Application Entity (AE)

Figure 3-1 depicts one possible organizational structure of Directory Services.



**Figure 3-1 Example Directory Services (X.500) Structure**



## Using Directory Services

FTAM uses Directory Services whenever you call `ft_connect()` or high level calls to specify your directory or the directory with which you want to communicate.

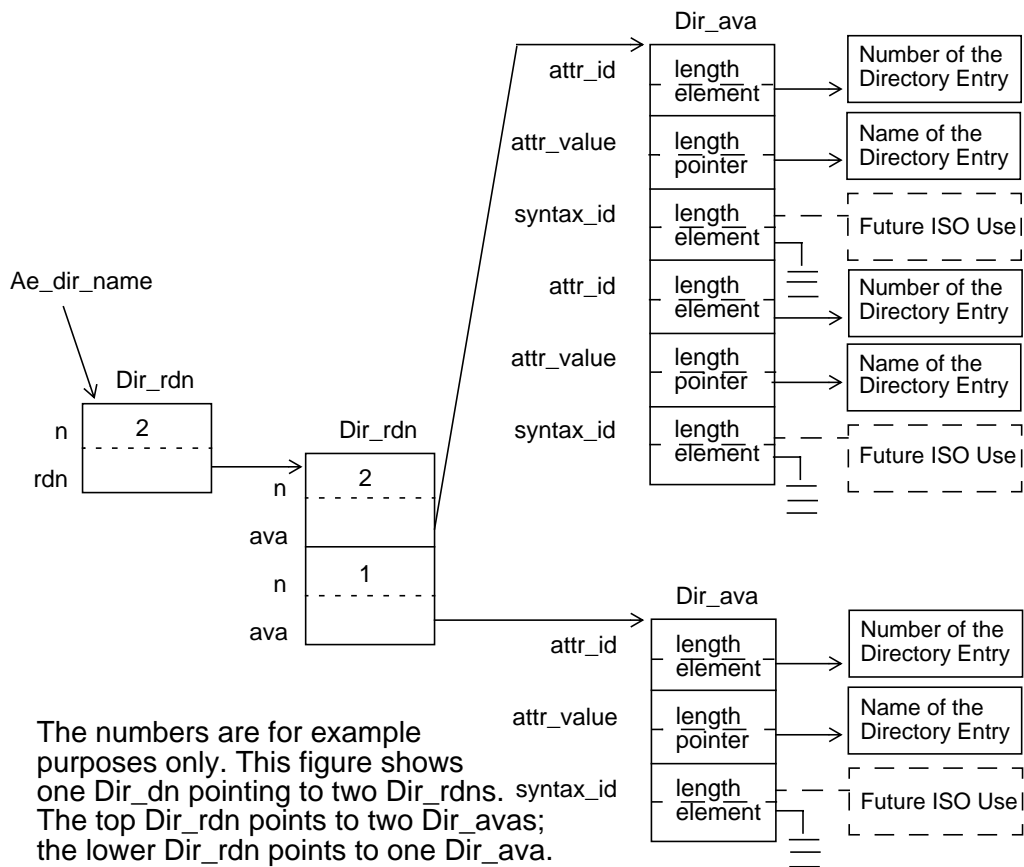
<b>Function</b>	<b>Parameter</b>	<b>Identifies</b>
<code>ft_connect()</code>	<code>called_dir_name</code>	Directory distinguished name of the responder process to which you want to connect
<code>ft_fcopy()</code> and <code>ft_fmove()</code>	<code>source_dirname</code>	Directory distinguished name specifying the source filestore
	<code>destination_dirname</code>	Directory distinguished name specifying the destination filestore
<code>ft_fattributes()</code> <code>ft_frattributes()</code> <code>ft_fdelete()</code>	<code>dirname</code>	Directory distinguished name of the responder process from which you want to change or read attributes, or delete a file

## Ae\_dir\_name

```
typedef Dir_dn      *Ae_dir_name;
```

Ae\_dir\_name is the primary structure for setting the directory distinguished name when calling ft\_connect(), ft\_aeactivation(), and all high level, context free (HLCF) functions. The embedded structures are Dir\_dn, Dir\_rdn, and Dir\_ava ( Figure 3-2). To have the system automatically set the Ae\_dir\_name structure, call convert\_ddn; the source is in /opt/ftam/demos/cnvrtd\_addr.c.

**Figure 3-2 Ae\_dir\_name Structure**



## Dir\_dn

```
typedef struct Dir_dn /* Directory Distinguished name */
{
  Sint8      n;          /*No. of relative dist. names */
  struct Dir_rdn *rdn;  /*Ptr to a sequence of Dir_rdn */
} Dir_dn;
```

Dir\_dn identifies the directory distinguished name.

- The first field (n) is the number of relative distinguished names (rdn).
- The rdn field points to an array of Dir\_rdn structures containing attribute value assertions (avas).
- The embedded structures are Dir\_rdn and Dir\_ava.
- Each Dir\_dn that specifies an application entity (AE) has a corresponding P\_address stored in the ICS, which provides the mapping between the two.
- The Dir\_dn must be exactly the same as the one set during configuration.

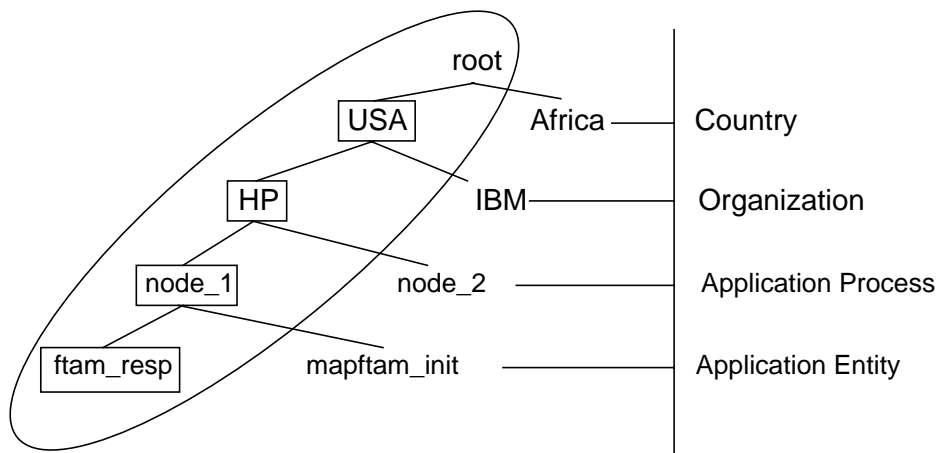
### EXAMPLE

In this example, the directory distinguished name for the ftam\_resp is as follows. Each portion of the name is an rdn (e.g., O=HP).

/C=US/O=HP/AP=node\_1/AE=ftam\_resp

Figure 3-3

Example Directory Distinguished Name Structure



## Dir\_rdn

```
typedef struct Dir_rdn /* Relative distinguished names */
{
    Sint8      n;          /*Number of avas */
    struct Dir_ava *avas;
} Dir_rdn;
```

Dir\_rdn identifies the relative distinguished name (Figure 3-4).

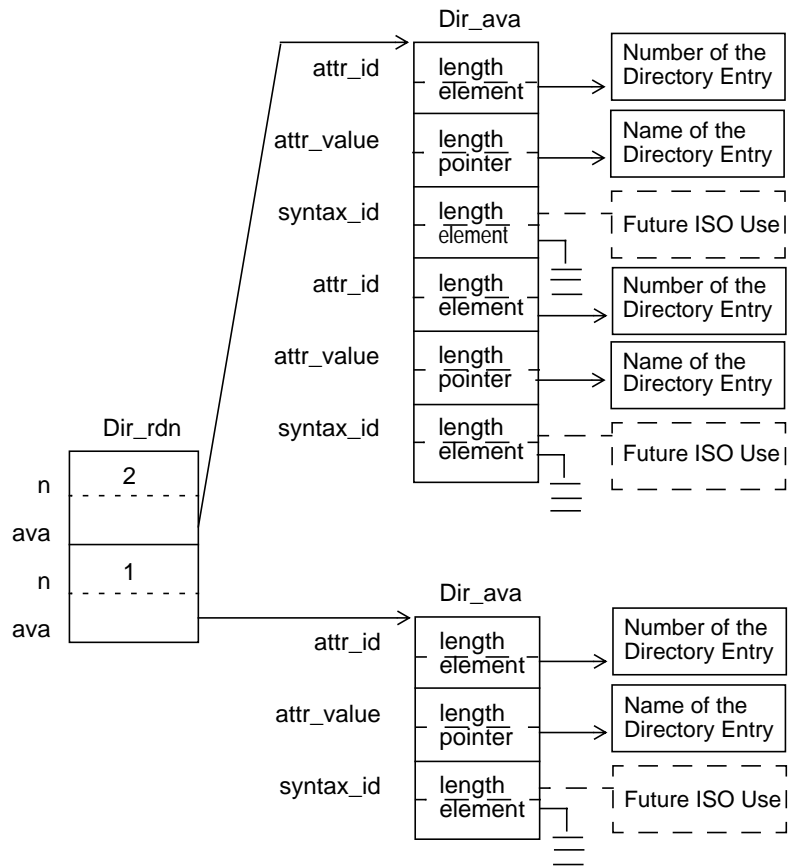
- Each rdn is part of the directory distinguished name. Refer to the following example.
- The first field (n) is the number of attribute value assertions (avas).
- Each avas points to a Dir\_ava structure containing attr\_id, attr\_value, and syntax\_id.

### EXAMPLE

This example depicts five rdns: three rdns contain one avas each, and one rdn (OU) contains two avas.

```
/C=us/O=hp/L=ca/OU=falc/OU=hawk/AP=nodal/AE=ftam_resp
```

**Figure 3-4 Dir\_rdn Structure**



The numbers are for example purposes only. The top `Dir_rdn` points to two `Dir_ava`; the lower `Dir_rdn` points to one `Dir_ava`.

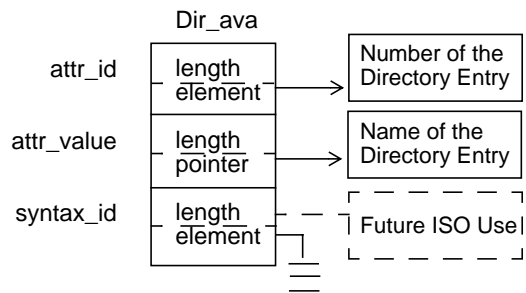
## Dir\_ava

```
typedef struct Dir_ava /* Attribute Value assertions */  
{  
  struct Object_id attr_id;  
  struct Octet_string attr_value;  
  struct Object_id syntax_id; /* for future use */  
} Dir_ava;
```

Dir\_ava is the final embedded structure in the Ae\_dir\_name structure that defines the directory distinguished name (Figure 3-5).

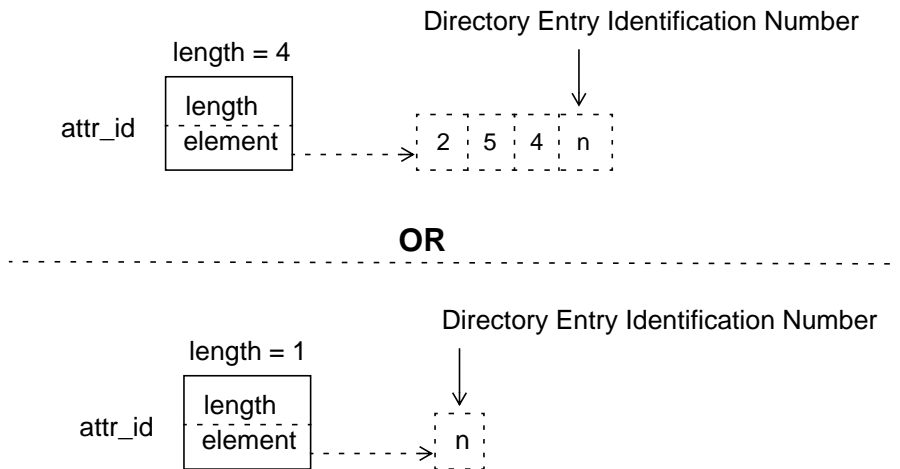
Each Dir\_ava contains the following fields: attr\_id (e.g., country), attr\_value (e.g., USA), and syntax\_id (reserved for future ISO extensions).

**Figure 3-5 Dir\_ava Structure**



- attr\_id** The attr\_id is an attribute type identifier that is of type struct Object\_id. You must set attr\_id in one of these two ways.
- Set the first three fields to 2, 5, and 4. Set the fourth field to the directory entry identification number. You must use these numbers since they define a standard attribute type for the protocol.
    - 2 = joint\_iso\_ccitt
    - 5 = attribute syntax
    - 4 = protocol Object\_id
    - n = attr\_id or directory entry identification number
  - Set only the directory entry identification number
- attr\_value** The attr\_value is of type struct Octet\_string. Set attr\_value to the name of the country in the Directory Services entry. Set the length equivalent to the number of characters contained in the attr\_value pointer.

**Figure 3-6 attr\_id Structure**





## Setting Ae\_dir\_name Example

This example sets the Ae\_dir\_name structure.

```
#include "map.h"
#include "mapftam.h"
#include <stdio.h>
#include <malloc.h>
/*
**
**          setup_ddn
**
** DESCRIPTION:
** This routine assigns valid values to the Ae_dir_name
** structure.
**
** PARAMETERS:
** Outputs:
**   dirname :      pointer to the Ae_dir_name structure
**                  Ae_dir_name is a pointer to struct
**                  Dir_dn.
**
**
*/
void
setup_ddn(dirname)
Ae_dir_name      *dirname;
{
    /*
     * Initialize all fields of the Ae_dir_name structure.
     * Set up:
     */
    "/C=us/O=hp/OU=org_unit/CN=machine_name/CN=ftam_resp"
    /*
     *dirname = (struct Dir_dn *)malloc(sizeof(struct Dir_dn));
     (*dirname)->rdn=(struct Dir_rdn *)malloc(5*sizeof(struct
     Dir_rdn));
     (*dirname)->n = 5;
     */
    /*
     * C = us
     * The Country attribute id is defined by ISO to be "6".
     */
    (void)addrdn( & (*dirname)->rdn[0], 6, "us");
    /*
     * O = hp
     * The Organization attribute id is defined by ISO to be
     "10".
     */
    (void)addrdn( & (*dirname)->rdn[1], 10, "hp");
    /*
     * OU = org_unit
     * The Organization Unit attribute id is defined by ISO to be
     "11".
     */
    (void)addrdn( & (*dirname)->rdn[2], 11, "org_unit");
    /*
     * AP = machine_name
     * The Application Process attribute id is defined by ISO to

```

HP FTAM/9000 Data Structures  
**Directory Services Data Structures**

```

be "3".
    /*
    (void)addrdn( & (*dirname)->rdn[3], 3, "machine_name");
    /*
        AE = ftam_resp
        The Application Entity attribute id is defined by ISO to
be "3".
    /*
    (void)addrdn( & (*dirname)->rdn[4], 3, "ftam_resp");
}
/*
***          addrdn
**
** DESCRIPTION:
** This routine sets up the Dir_rdn structure using the input
** parameters for values.
**
** PARAMETERS:
** Input:
**   att_id :          integer value for the attribute id
**   att_value :      character string for the attribute value
**
** Output:
**   rdn :            pointer to struct Dir_rdn being set up
**
*/
Sint32
addrdn(rdn, att_id, att_value)
struct Dir_rdn      *rdn;
int                 att_id;
char                 *att_value;
{
    /* Allocate the memory for struct Dir_ava
    */
    rdn->n = 1;
    rdn->avas = (struct Dir_ava *)malloc(sizeof(struct Dir_ava));
    rdn->avas->attr_value.pointer = (Octet *)malloc
        (strlen(att_value)*sizeof(Octet));
    rdn->avas->attr_id.element = (Sint32 *)malloc(sizeof(Sint32));
    /*
        Set up the attribute id
    */
    *(rdn->avas->attr_id.element) = att_id;
    rdn->avas->attr_id.length = 1;
    /*
        Set up the attribute value
    */
    memcpy(rdn->avas->attr_value.pointer, att_value,
strlen(att_value));
    rdn->avas->attr_value.length = strlen(att_value);
    /*
        The syntax id is for future use
    */
    rdn->avas->syntax_id.element = NULL;
    rdn->avas->syntax_id.length = 0;
}

```

---

## Ae\_label

```
typedef Uint32 Ae_label
```

### Ae\_label Is Input To These Functions

ft\_aedeactivation()  
ft\_aereset()  
ft\_connect()  
ft\_fcattributes()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fmove()  
ft\_frattributes()

### Ae\_label Is Output From These Functions

ft\_aeactivation()  
ft\_fcattributes()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fmove()  
ft\_frattributes()

Ae\_label identifies the ftam\_init activated on a successful ft\_aeactivation() request. On subsequent ft\_connect(), ft\_aereset(), and ft\_aedeactivation() requests, you must specify the ae\_label of the ftam\_init servicing the user program.

Use Ae\_label on HLCF functions to uniquely identify the desired ftam\_init servicing the HLCF call.

- If the value pointed to by the ae\_label is NULL, the interface activates ftam\_init for you, returns its ae\_label identifier to the user program, and then deactivates ftam\_init.
- If the value is not NULL, the interfaces services the call based on the ftam\_init identified by ae\_label.

**Ae\_title**

---

**Ae\_title**

```
typedef union Ae_title
{
  struct Object_id      ae_object_id;
  struct Dir_dn        ae_dir_dn;
} Ae_title;
```

**Ae\_title Is Input To These Functions**

ft\_aeactivation()  
ft\_connect()

**Ae\_title Is Output From These Functions**

ft\_connect()

The my\_ae\_title input to ft\_aeactivation() specifies the ae\_title of the ftam\_init to activate. The called\_ae\_title input to ft\_connect() enables you to specify an optional ae\_title for the responder.

Both my\_ae\_title and called\_ae\_title may be represented as either an ae\_object\_id or ae\_dir\_dn. However, HP-UX FTAM supports only ae\_object\_id.

The Ae\_title\_option value determines whether the Ae\_title union takes the ae\_object\_id, ae\_dir\_dn, or No\_value\_option form.

Ae\_title carries no semantic value. If not used, set length to zero and ae\_dir\_dn to NULL.

If you use Ae\_title, the object\_id.element value is set for you to the following value: 1 3 9999 1 ftam\_nil\_ap\_title (7).

ae_object_id	Contains local and remote application entity (AE) information.
ae_dir_dn	Contains the directory distinguished name form of the Ae_title.

---

## Ae\_title\_option

```
enum Ae_title_option
{
    No_value_option,
    Dir_object_id_option, /* Object_id value obtained from the
                          * network or local system directory */
    User_object_id_option, /* Object_id value supplied by the
                          * user */
    Dir_dist_name_option, /* Use the distinguished name value
                          * provided by the my_dir_name
                          * or called_dir_name parameter. */
    User_dist_name_option /* Distinguished name value supplied
                          * by the user. */
};
```

### Ae\_title\_option Is Input To These Functions

ft\_aeactivation()  
ft\_connect()

### Ae\_title\_option Is Output From These Functions

None

Ae\_title\_option specifies whether to use the Object\_id or directory distinguished name on ft\_connect() and ft\_aeactivation().

The following tables list the supported values for the appropriate functions. HP-UX responders return an error if you use an unsupported value.

Ae_title_option	ft_aeactivation()	ft_connect()
No_value_option	Supported	Supported
Dir_object_id_option	Supported	Not supported

**Ae\_title\_option**

<b>Ae_title_option</b>	<b>ft_aeactivation()</b>	<b>ft_connect()</b>
User_object_id_option	Supported	Supported
Dir_dist_name_option	Not supported	Not supported
User_dist_name_option	Not supported	Not supported
No_value_option	Ignore the Ae_title on ft_aeactivation() and ft_connect(); no Ae_title is sent to the remote system.	
	<ul style="list-style-type: none"> <li>• For ft_aeactivation(), use my_dir_name.</li> <li>• For ft_connect(), use the P_address if it exists; otherwise, use called_dir_name.</li> </ul>	
Dir_object_id_option	The network or local system directory supplies the Object_id. For ft_aeactivation(), FTAM uses my_dir_name to query the Initial Configuration Store (ICS) for the Object_id.	
User_object_id_option	You supply the Object_id value. If you specify User_object_id_option, you must also specify ae_object_id in the Ae_title.	
Dir_dist_name_option	Ignore the Ae_title.	
	<ul style="list-style-type: none"> <li>• For ft_aeactivation(), use my_dir_name as the Ae_title.</li> <li>• For ft_connect(), use called_dir_name as the Ae_title.</li> </ul>	
User_dist_name_option	You supply the directory distinguished name. If you specify User_dist_name_option, you must also specify ae_dir_dn in Ae_title union.	

---

## Api\_rc

```
typedef struct Api_rc
{
Return_code   return_code; /* MAP errors */
Return_code   vendor_code; /* Optional Vendor defined errors */
} Api_rc;
```

**Api\_rc** (Application Interface return\_code) provides you with error information when a function fails. Every function has a parameter of this type, usually in its inout\_dcb and usually defined as the result parameter.

Api\_rc

return_code	<p>Specifies the MAP 3.0 error that occurred. These errors are listed as defined FTExxx constants. Refer to the <i>HP FTAM/9000 Reference Manual</i> for a list of possible errors, their causes, and corrective actions.</p> <ul style="list-style-type: none"><li>• A return_code value of SUCCESS (zero) indicates the request went through the application interface, ftam_init, and responder, and returned back to the user program. If the return_code value indicates an error (non-zero), the error could be detected in any one of these locations.</li><li>• For synchronous calls, the function return value and return_code are always identical.</li><li>• For asynchronous calls with a function return value of SUCCESS, the return_code indicates the success or failure of the call.</li><li>• For asynchronous calls, the function return value and return_code are identical if the function return value of the original call is not SUCCESS.</li></ul>
vendor_code	<p>Contains HP-UX vendor-specific error information that describes the error. Note, not all errors return vendor_codes.</p>

The upper 16 bits contain the log instance.

The lower 16 bits define errors that are listed as defined FTVxxx constants. Refer to the *HP FTAM/9000 Reference Manual* for a list of possible vendor\_codes, their causes, and corrective actions.



---

## Connection\_id

```
typedef Uint32      Connection_id;
```

### Connection\_id Is Input To These Functions

ft\_abort()  
ft\_bgroup()  
ft\_cancel()  
ft\_cattributes()  
ft\_close()  
ft\_create()  
ft\_delete()  
ft\_deselect()  
ft\_edata()  
ft\_egroup()  
ft\_erase()  
ft\_etransfer()  
ft\_fclose()  
ft\_fopen()  
ft\_ireceive()  
ft\_locate()  
ft\_nwcleared()  
ft\_open()  
ft\_rattributes()  
ft\_rcancel()  
ft\_rdata()  
ft\_read()  
ft\_rrequest()  
ft\_sdata()  
ft\_select()  
ft\_write()

### Connection\_id Is Output From These Functions

ft\_connect()

If `ft_connect()` is successful, a unique identifier for the connection returns in `connection_id`. Thereafter, use `connection_id` for any of the functions listed in the above table to identify the connection to the desired filestore. Each `connection_id` is unique to an application process.

## **Ft\_access\_context**

```
enum    Ft_access_context {
        FT_AC_ENUM_DEFAULT                = -1,
        FT_HIERARCHICAL_ALL_DATA_UNITS    = 0,
        FT_HIERARCHICAL_NO_DATA_UNITS     = 1,
        FT_FLAT_ALL_DATA_UNITS            = 2,
        FT_FLAT_ONE_LEVEL_DATA_UNITS      = 3,
        FT_FLAT_SINGLE_DATA_UNIT          = 4,
        FT_UNSTRUCTURED_ALL_DATA_UNITS     = 5,
        FT_UNSTRUCTURED_SINGLE_DATA_UNIT  = 6
    };
```

### **Ft\_access\_context Is Input To These Functions**

`ft_read()`

### **Ft\_access\_context Is Output From These Functions**

None

`Ft_access_context` specifies how you want to read the file depending on the document type.

- For FTAM-1, FTAM-3, and INTAP-1 files, you must specify `FT_UNSTRUCTURED_ALL_DATA_UNITS`.
- For FTAM-2 files, you must specify either `FT_FLAT_ALL_DATA_UNITS` or `FT_UNSTRUCTURED_ALL_DATA_UNITS`.

FT\_FLAT\_ALL\_DATA\_UNITS

For FTAM-2 files, specifies that you will see both file structure and data.

FT\_UNSTRUCTURED\_ALL\_DATA\_UNITS

For FTAM-2 files, specifies that you will see only data. For FTAM-1 and FTAM-3 files, reads the whole file.

FT\_AC\_ENUM\_DEFAULT

FT\_HIERARCHICAL\_ALL\_DATA\_UNITS

FT\_HIERARCHICAL\_NO\_DATA\_UNITS

FT\_FLAT\_ONE\_LEVEL\_DATA\_UNITS

FT\_FLAT\_SINGLE\_DATA\_UNIT

FT\_UNSTRUCTURED\_SINGLE\_DATA\_UNIT

HP-UX responders return an error if you use one of these values.

---

## Ft\_access\_control

```
struct Ft_access_control {
    struct Ft_access_control_element  *insert_ace;
    struct Ft_access_control_element  *delete_ace;
};
```

### **Ft\_access\_control Is Input To These Functions**

ft\_cattributes()  
ft\_create()  
ft\_fattributes()  
ft\_fopen()

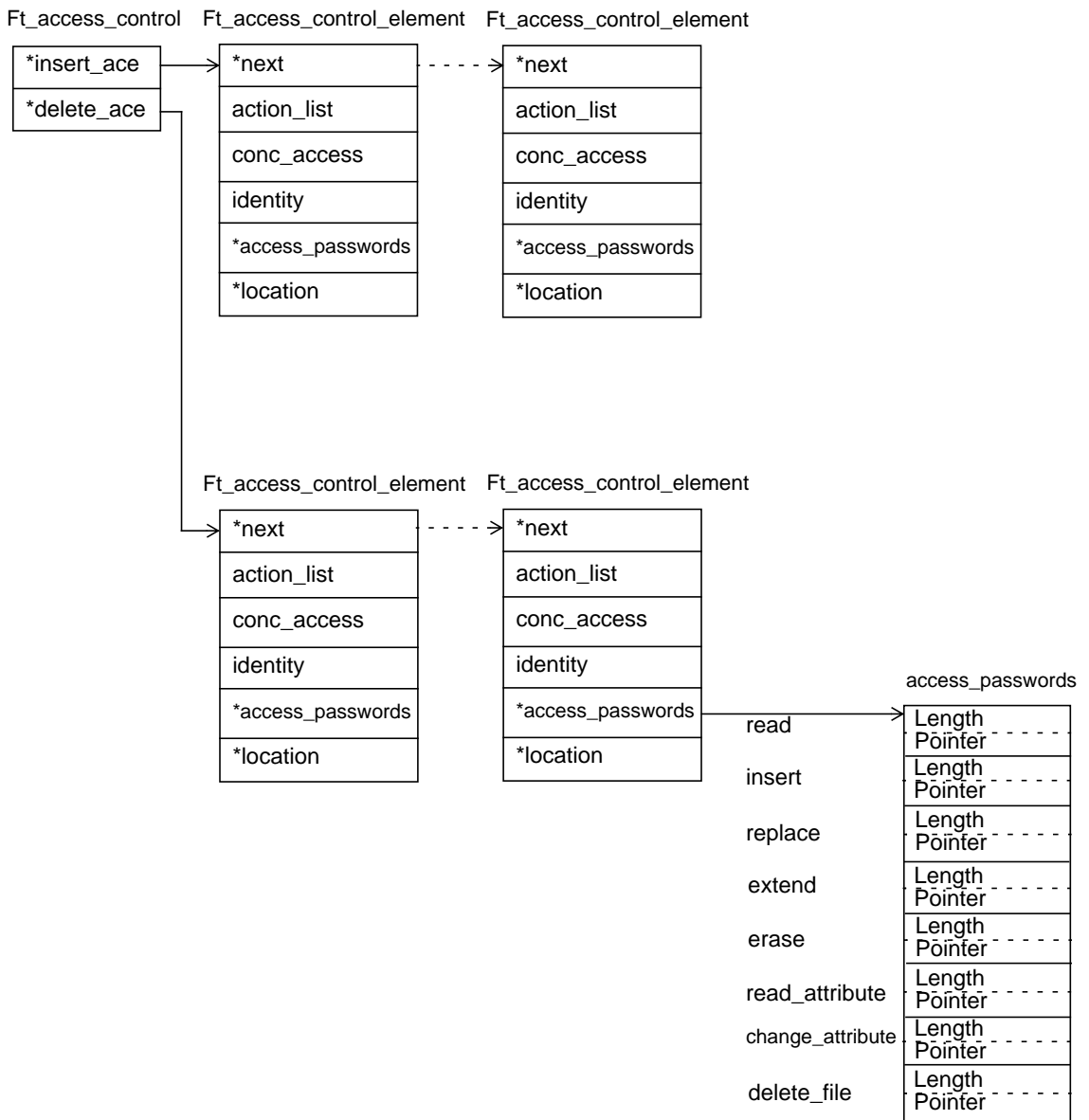
### **Ft\_access\_control Is Output From These Functions**

None

Use Ft\_access\_control to add users to and delete users from the access list (Ft\_access\_control\_elements) for a particular file. Both insert\_ace and delete\_ace are linked lists of Ft\_access\_control\_element structures ( ). This structure specifies user access privileges for the file.

- If insert\_ace is NULL when creating a file (ft\_create() and ft\_fopen()), anyone can access the file.
- If you use both insert\_ace and delete\_ace, HP-UX responders process delete\_ace first.

**Figure 3-7 Ft\_access\_control Structure**



**Ft\_access\_control****Ft\_access\_control\_element**

```

struct Ft_access_control_element {
    struct Ft_access_control_element *next;
    Ft_file_actions action_list;
    Uint32 conc_access;
    Ft_initiator_identity identity;
    struct Ft_file_passwords *access_passwords;
    Ae *location;
};

```

**Ft\_access\_control\_element** defines not only who can access a file, but also how they can access it. HP-UX initiators and responders use only the **identity** and **action\_list** fields. You can set values in the other fields if another FTAM implementation uses them.

When interacting with an HP-UX FTAM responder, you can use at most three (3) access control elements, one each for user, group, and other. Other responders may have different requirements.

<b>*next</b>	Points to the next <b>Ft_access_control_element</b> structure in the linked list. For HP-UX FTAM responders, the maximum number in the linked list is three per file.
<b>action_list</b>	Specifies the allowable actions the designated users can perform on a file. An action is present if the corresponding bit is set (ON). <ul style="list-style-type: none"> <li>Change Attribute</li> <li>Delete File</li> <li>Erase</li> <li>Extend</li> <li>Insert</li> <li>Read</li> <li>Read Attribute</li> <li>Replace</li> </ul>

For **ft\_select()** and **ft\_fopen()**, the **requested\_access** parameter must be a subset of the values stored in **action\_list**.

For `ft_open()`, the `processing_mode` parameter must be a subset of the values stored in `action_list`.

The `action_list` is of type `Ft_file_actions`. Refer to the “`Ft_file_actions`” section for detailed information.

`conc_access`

The HP-UX implementation of FTAM ignores this value. You can set the bits in `conc_access`, but HP-UX initiators and responders will not use it. Other FTAM implementations might. If you are unsure of your situation, see the documentation for the other FTAM implementation.

Bits in `conc_access` correspond to the allowable actions listed in the `action_list`. Here are the allowed locks, in order, from least to most restrictive:

File Action Lock	File Owner Can	Others Can
Set by <code>conc_access</code>	Perform the Action	Perform the Action
Not Required	No	Yes
Shared	Yes	Yes
Exclusive	Yes	No
No Access	No	No

Use the following defined constants to set any combination of `conc_access` bits for file actions. Again, note that these locks are ignored by HP-UX initiators and responders.

```

FT_AC_READ_NOTREQ      FT_AC_REPLACE_NOACC      FT_AC_READ_ATTRIB_EXCL
FT_AC_READ_SHARED      FT_AC_EXTEND_NOTREQ      FT_AC_READ_ATTRIB_NOACC
FT_AC_READ_EXCL        FT_AC_EXTEND_SHARED      FT_AC_CHANGE_ATTRIB_NOTREQ
FT_AC_READ_NOACC       FT_AC_EXTEND_EXCL        FT_AC_CHANGE_ATTRIB_SHARED
FT_AC_INSERT_NOTREQ    FT_AC_EXTEND_NOACC       FT_AC_CHANGE_ATTRIB_EXCL
FT_AC_INSERT_SHARED    FT_AC_ERASE_NOTREQ       FT_AC_CHANGE_ATTRIB_NOACC
FT_AC_INSERT_EXCL      FT_AC_ERASE_SHARED       FT_AC_DELETE_FILE_NOTREQ
FT_AC_INSERT_NOACC     FT_AC_ERASE_EXCL        FT_AC_DELETE_FILE_SHARED
FT_AC_REPLACE_NOTREQ   FT_AC_ERASE_NOACC       FT_AC_DELETE_FILE_EXCL
FT_AC_REPLACE_SHARED   FT_AC_READ_ATTRIB_NOTREQ FT_AC_DELETE_FILE_NOACC
FT_AC_REPLACE_EXCL     FT_AC_READ_ATTRIB_SHARED

```

`identity`

Specifies the user to whom the access privileges pertain (the “owner” in the previous table). For HP-UX initiators and responders, `identity` must have one of the following values: `user`, `group`, or `other`.

The `identity` is of type `struct Ft_initiator_identity`. Refer to the “`Ft_initiator_identity`” section for detailed information.

**Ft\_access\_control**

---

*access_passwords	Specifies a user password for each element listed in the action_list. HP-UX initiators and responders do not use this information; access_passwords is only used with other FTAM implementations.  Points to the Ft_file_passwords structure. Refer to the “Ft_file_passwords” section for detailed information.
*location	A pointer of type Ae. HP-UX initiators and responders do not use this information; location is only used with other FTAM implementations.



---

## Ft\_account

```
typedef char      *Ft_account;
```

### **Ft\_account Is Input To These Functions**

ft\_connect()  
ft\_create()  
ft\_fattributes()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fmove()  
ft\_fopen()  
ft\_frattributes()  
ft\_select()

### **Ft\_account Is Output From These Functions**

None

HP-UX responders accept, but ignore, Ft\_account. Setting the value to NULL is a recommended practice.

---

## Ft\_attribute\_groups

```
typedef Uint16    Ft_attribute_groups;
```

### **Ft\_attribute\_groups Is Input To These Functions**

ft\_connect()

### **Ft\_attribute\_groups Is Output From These Functions**

ft\_connect()

Ft\_attribute\_groups specifies the level of support for attributes.

Attributes are grouped as either kernel, storage, security, or private.

- The kernel attributes are always present (i.e., you do not negotiate them).
- Attributes within the storage, security, and private groups are valid only if you negotiate that group during ft\_connect().
- Response to an attribute value request (e.g., ft\_cattributes()) always includes one of the following items:
  - If you request an unnegotiated attribute, the response includes no value available and an error (result\_code); may optionally include a diagnostic.
  - If the HP-UX responder partially supports the attribute, the value (zero (0) for integers; “no value available” for strings) indicates that there is no value available.
  - If the HP-UX responder fully supports the attribute, actual values are returned.
- For ft\_rattributes() and ft\_frattributes(), the attribute\_names field defaults to indicate all attributes in the attribute\_groups negotiated on the ft\_connect() request.

The available attribute groups and their associated attributes are as follows:

**Kernel Group** Provides basic attributes that are always available.

filename  
permitted\_actions  
contents\_type

**Storage Group** Provides additional information on how the file is stored.

storage\_account \*<sup>1</sup>  
date\_time\_of\_creation  
date\_time\_of\_modification  
date\_time\_of\_read  
date\_time\_of\_attribute\_mod<sup>1</sup>  
identity\_of\_creator  
identity\_of\_modifier<sup>1</sup>  
identity\_of\_reader<sup>1</sup>  
identity\_of\_attribute\_mod<sup>1</sup>  
file\_availability \*  
filesize  
future\_filesize \*<sup>1</sup>

<sup>1</sup> When these values are output from HP-UX FTAM responders, the value (zero (0) for integers; “no value available” for strings) signifies there is no value for the attribute.

**Ft\_attribute\_groups**

---

Security Group	Allows FTAM to enforce what users may operate on the file and what access each user has.
----------------	--

access\_control

legal\_qualification \*

Private Group	ISO does not define the private attributes group.
---------------	---

private\_use \*

\* HP-UX responders accept, but ignore, these values.

To set attribute groups, use the FT\_AG\_xxx defined constants in the mapftam.h file.

FT\_AG\_STORAGE

FT\_AG\_SECURITY

FT\_AG\_PRIVATE

---

## Ft\_attribute\_names

```
typedef Uint32 Ft_attribute_names;
```

### Ft\_attribute\_names Is Input To These Functions

ft\_cattributes()  
ft\_create()  
ft\_fcattributes()  
ft\_fopen()  
ft\_frattributes()  
ft\_rattributes()

### Ft\_attribute\_names Is Output From These Functions

None

This section discusses Ft\_attribute\_names as a primary structure. For information on Ft\_attribute\_names as it relates to the mask field in struct Ft\_attributes, refer to the “Ft\_attributes” section.

Ft\_attribute\_names specifies the attributes you want available for each file. Ft\_attribute\_names is the type for the mask field in Ft\_attributes.

- In ft\_create() and ft\_fopen(), use Ft\_attribute\_names to specify which attributes you want to set for the newly created file.
- In ft\_rattributes() and ft\_frattributes(), use Ft\_attribute\_names to specify which attributes you want to read.
- In ft\_cattributes() and ft\_fcattributes(), use Ft\_attribute\_names to specify which attributes you want to change.

---

#### NOTE

When using Ft\_cattributes or Ft\_fcattributes, the attribute\_names field has precedence over attributes.mask.

---

## HP FTAM/9000 Data Structures

### **Ft\_attribute\_names**

Use **Ft\_attribute\_names** to set the attribute values you want available for each file. Use the **FT\_AN\_xxx** defined constants in the **mapftam.h** file.

```
FT_AN_FILENAME
FT_AN_PERMITTED_ACTIONS
FT_AN_CONTENT_TYPE
FT_AN_STORAGE_ACCT
FT_AN_CREATE_DATE_TIME
FT_AN_MOD_DATE_TIME
FT_AN_READ_DATE_TIME
FT_AN_ATT_MOD_DATE_TIME
FT_AN_ID_OF_CREATOR
FT_AN_ID_OF_MODIFIER
FT_AN_ID_OF_READER
FT_AN_ID_OF_ATT_MOD
FT_AN_FILE_AVAILABILITY
FT_AN_FILESIZE
FT_AN_FUTURE_FILESIZE
FT_AN_ACCESS_CONTROL
FT_AN_LEGAL_QUAL
FT_AN_PRIVATE_USE
```

If you invoke **ft\_rattributes()** or **ft\_fattributes()**, HP-UX responders return the indication no value available (0) for all partially supported attributes.

If you invoke functions that contain attributes in their **input\_dcb**, HP-UX responders accept, but ignore, all partially supported attributes.

---

## Ft\_attributes

```
struct Ft_attributes {  
    Ft_attribute_names    mask;  
    struct Ft_attribute_values values;  
};
```

### **Ft\_attributes Is Input To These Functions**

ft\_cattributes()  
ft\_create()  
ft\_fcattributes()  
ft\_fopen()  
ft\_select()

### **Ft\_attributes Is Output From These Functions**

ft\_cattributes()  
ft\_create()  
ft\_fcattributes()  
ft\_fcopy()  
ft\_fmove()  
ft\_fopen()  
ft\_frattributes()  
ft\_rattributes()  
ft\_select()

Every file contains attributes that uniquely identify the file and are inherent in its existence (e.g., filename). Ft\_attributes is the primary structure for setting these attributes. You can change only certain file attributes after creating the file, so be careful to set these attributes as you want them during ft\_fopen() and ft\_create().

<b>Attributes You Can Change</b>	<b>Attributes You Cannot Change</b>
filename	permitted_actions
storage_account*	contents_type
file_availability*	date_time_of_creation
future_filesize*	date_time_of_modification
access_control	date_time_of_read
legal_qualification*	date_time_of_attribute_mod
private_use*	identity_of_creator
	identity_of_modifier
	identity_of_reader
	identity_of_attribute_mod
	filesize

\* With HP-UX FTAM responders, changing these values has no effect.

Ft\_attributes may exist in the input\_dcb and inout\_dcb.

mask	The mask identifies which attributes are available for each file.
values	The values identifies the attribute values of the attributes specified in the mask.



## Ft\_attribute\_names

```
typedef Uint32 Ft_attribute_names;
```

### Ft\_attribute\_names Is Input To These Functions

ft\_cattributes()  
ft\_create()  
ft\_fcattributes()  
ft\_fopen()  
ft\_frattributes()  
ft\_rattributes()

### Ft\_attribute\_names Is Output From These Functions

ft\_cattributes()  
ft\_create()  
ft\_fcattributes()  
ft\_fopen()  
ft\_frattributes()  
ft\_rattributes()

Ft\_attribute\_names is the type for the mask field in Ft\_attributes.

Use Ft\_attribute\_names to identify which attributes you want available for each file. Use only those values in the FT\_AN\_XXX defined constants. Refer to the previous “Ft\_attribute\_names” section for more information.

---

**NOTE**

When using Ft\_cattributes or Ft\_fcattributes, the attribute\_names field has precedence over attributes.mask.

---

**Ft\_attributes****Ft\_attribute\_values**

```

struct Ft_attribute_values {
    Ft_filename                filename;
    Ft_permitted_actions       permitted_actions;
    struct Ft_contents_type    contents_type;
    Ft_account                 storage_account;
    Time                      date_time_of_creation;
    Time                      date_time_of_modification;
    Time                      date_time_of_read;
    Time                      date_time_of_attribute_mod;
    Ft_initiator_identity      identity_of_creator;
    Ft_initiator_identity      identity_of_modifier;
    Ft_initiator_identity      identity_of_reader;
    Ft_initiator_identity      identity_of_attribute_mod;
    enum Ft_file_availability   file_availability;
    Sint32                    filesize;
    Sint32                    future_filesize;
    struct Ft_access_control    access_control;
    Ft_legal_qualification      legal_qualification;
    struct Octet_string         private_use;
};

```

Use `Ft_attribute_values` to specify the attribute values you want each attribute to have.

filename	Character string that identifies the name of the file.															
permitted_actions	<p>Specifies all allowable actions on the file (without specifying who is allowed to perform the actions). The permitted_actions is type Ft_permitted_actions.</p> <ul style="list-style-type: none"><li>For each file, set bits in permitted_actions for the actions you are allowing to be performed. Use these defined constants:</li></ul> <p>FT_PA_READ FT_PA_INSERT FT_PA_REPLACE FT_PA_EXTEND FT_PA_ERASE FT_PA_READ_ATTRIBUTE FT_PA_CHANGE_ATTRIBUTE FT_PA_DELETE_FILE FT_PA_TRAVERSAL FT_PA_REV_TRAVERSAL FT_PA_RANDOM_ORDER</p> <ul style="list-style-type: none"><li>Each document type defines certain permitted_actions values, as follows:</li></ul> <table><thead><tr><th><b>FTAM-1, -2, -3, INTAP-1 and NBS-9</b></th><th><b>FTAM-1, FTAM-3 and INTAP-1</b></th><th><b>FTAM-2</b></th></tr></thead><tbody><tr><td>CHANGE_ATTRIBUTE</td><td>ERASE</td><td>ERASE</td></tr><tr><td>DELETE_FILE</td><td>EXTEND</td><td>INSERT</td></tr><tr><td>READ</td><td>REPLACE</td><td>TRAVERSAL</td></tr><tr><td>READ_ATTRIBUTE</td><td></td><td></td></tr></tbody></table>	<b>FTAM-1, -2, -3, INTAP-1 and NBS-9</b>	<b>FTAM-1, FTAM-3 and INTAP-1</b>	<b>FTAM-2</b>	CHANGE_ATTRIBUTE	ERASE	ERASE	DELETE_FILE	EXTEND	INSERT	READ	REPLACE	TRAVERSAL	READ_ATTRIBUTE		
<b>FTAM-1, -2, -3, INTAP-1 and NBS-9</b>	<b>FTAM-1, FTAM-3 and INTAP-1</b>	<b>FTAM-2</b>														
CHANGE_ATTRIBUTE	ERASE	ERASE														
DELETE_FILE	EXTEND	INSERT														
READ	REPLACE	TRAVERSAL														
READ_ATTRIBUTE																
contents_type	Specifies the document type of the file; the contents_type is of type struct Ft_contents_type. Refer to the "Ft_contents_type" section for a detailed explanation.															

**Ft\_attributes**

---

<b>storage_account</b>	Identifies the accountable authority responsible for accumulated file storage charges. HP-UX FTAM responders set <code>storage_account</code> so that the indication no value available (0) returns if you invoke <code>ft_rattributes()</code> or <code>ft_fattributes()</code> .  If you invoke functions that contain <code>storage_account</code> in the <code>input_dcb</code> , HP-UX responders accept, but ignore, the value.
	Specifies, respectively, the following dates.
<b>date_time_of_creation</b>	Date file was created
<b>date_time_of_modification</b>	Date file was last modified
<b>date_time_of_read</b>	Date file was last read
<b>date_time_of_attribute_mod*</b>	Date file attributes were last modified
	These values are of type Time.
	Specifies, respectively, the following user identities:
<b>identity_of_creator</b>	User who created the file
<b>identity_of_modifier*</b>	User who last modified the file
<b>identity_of_reader*</b>	user who last read the file
<b>identity_of_attribute_mod*</b>	User who last modified the file attributes

\* Returns no value available

<b>file_availability</b>	<p>The <code>file_availability</code> is of the following type:</p> <pre>enum    Ft_file_availability {                                 FT_IMMEDIATE_AVAIL,                                 FT_DEFERRED_AVAIL                                 };</pre> <p>For HP-UX responders, you can access the file immediately regardless of which value you select. For some implementations, <code>file_availability</code> indicates whether a delay occurs before you can access the file.</p>
<b>filesize</b>	Specifies the maximum size (in Octets) of the file.
<b>future_filesize</b>	Specifies the maximum size (in Octets) to which the file may grow if you modify or extend it. HP-UX responders return a value of zero (no value) on calls to <code>ft_rattributes()</code> or <code>ft_frattributes()</code> .
<b>access_control</b>	Defines the conditions under which you and other users can access the file. The <code>access_control</code> is of type <code>struct Ft_access_control</code> and specifies who can access the file, their access privileges, types of file locks, and user passwords. Refer to the “ <code>Ft_access_control</code> ” section for a detailed explanation.
<b>legal_qualifications</b>	Conveys information about the legal status of the file and its use. HP-UX responders return a value of zero (no value) on calls to <code>ft_rattributes()</code> or <code>ft_frattributes()</code> .
<b>private_use</b>	HP-UX responders return a value of zero (no value) on calls to <code>ft_rattributes()</code> or <code>ft_frattributes()</code> .

---

## Ft\_concurrency\_control

```
struct Ft_concurrency_control {
    enum    Ft_file_lock    read;
    enum    Ft_file_lock    insert;
    enum    Ft_file_lock    replace;
    enum    Ft_file_lock    extend;
    enum    Ft_file_lock    erase;
    enum    Ft_file_lock    read_attribute;
    enum    Ft_file_lock    change_attribute;
    enum    Ft_file_lock    delete_file;
};
```

**Ft\_concurrency\_control****Ft\_concurrency\_control Is  
Input To These Functions**

ft\_create()  
 ft\_copy()  
 ft\_fattributes()  
 ft\_fdelete()  
 ft\_fmove()  
 ft\_fopen()  
 ft\_frattributes()  
 ft\_open()  
 ft\_select()

**Ft\_concurrency\_control Is  
Output From These Functions**

ft\_fopen()  
 ft\_open()

Concurrency locks define whether users can simultaneously access a file. Ft\_concurrency\_control specifies the type of available concurrency locks for each action on a file.

Change Attribute	Insert
Delete File	Read
Erase	Read Attribute
Extend	Replace

Each value is of type enum Ft\_file\_lock which specifies the type of concurrency locks on a file.

Setting allowable file actions and locks requires that you follow a few rules. Familiarize yourself with struct Ft\_concurrency\_control and enum Ft\_file\_lock, and then review the rules in the “Rules for Ft\_concurrency\_control” section.

## **Ft\_file\_lock**

```
enum    Ft_file_lock {
        FT_NOT_REQUIRED    = 0,
        FT_SHARED          = 1,
        FT_EXCLUSIVE       = 2,
        FT_NO_ACCESS       = 3
    };
```

**Ft\_file\_lock** enumerates the type of locks on a file. It is the enumerated type for values in struct **Ft\_concurrency\_control**. For example, if you want to be able to read a file, but you do not want other users to be able to, set the **concurrency\_control.read** lock to **FT\_EXCLUSIVE**.

<b>File Action Lock Set by Ft_file_lock</b>	<b>Owner May Perform The Action</b>	<b>Others May Perform The Action</b>
Not Required	No	Yes
Shared	Yes	Yes
Exclusive	Yes	No
No Access	No	No

## **Rules for Ft\_concurrency\_control**

- You may wish to set concurrency control to NULL wherever it exists as a parameter. This way, the responder uses its own default values. HP-UX responders use the defaults specified in the NBS Phase III agreements. The values used by a responder during the open regime are returned in the **inout\_dcb** of the **ft\_open()** and **ft\_fopen()** calls.
- If you use concurrency control (concurrency control is not NULL), you must specify a lock for each file action.
- For those actions that are available (because you requested them in the **requested\_access** parameter), the applicable locks are **FT\_SHARED** and **FT\_EXCLUSIVE**.
- For those actions not available (because you did not request them in the **requested\_access** parameter), the applicable locks are **FT\_NO\_ACCESS** and **FT\_NOT\_REQUIRED**.

**Ft\_concurrency\_control**

- The ft\_open() concurrency control must be identical to—or more restrictive than—the ft\_select() concurrency control; it may not be less restrictive

**CAUTION**

When a system has an NFS-mounted file system, two different files can have the same device id and inode number. If both are used by FTAM concurrently, the locks applied to one file could interfere with a select or open operation on the other file.

Furthermore, files on NFS-mounted file systems are visible on more than one host. Therefore, it is possible for FTAM to have locks on the same file from two different hosts. While each host may believe it has exclusive access to the file, in fact both hosts have concurrent access.

**Table 3-1 Concurrency Control: Default Settings**

In the SELECT regime:	For actions not set in the requested_access parameter, corresponding locks default to:	For each action set in the requested_access parameter, if the action is read or read_attribute, the corresponding lock defaults to:	For each action set in the requested_access parameter, otherwise, the corresponding lock defaults to:
	Not Required	Shared	Exclusive

In the OPEN regime:	By default, all locks in the OPEN regime are set to the lock value for the current SELECT regime, either the defaults (above) or user-selected.
---------------------	---



---

## Ft\_contents\_type\_element

```
struct Ft_contents_type_element {
    struct Ft_contents_type_element *next_element;
    struct Ft_contents_type         contents_type;
};
```

### **Ft\_contents\_type\_element Is Input To These Functions**

ft\_connect()

### **Ft\_contents\_type\_element Is Output From These Functions**

ft\_connect()

On input, Ft\_contents\_type\_element is the requested document types per connection; on output, it is a linked list of allowable (negotiated) document types per connection. Use Ft\_contents\_type\_element to specify up to three document types: FTAM-1, FTAM-2, and FTAM-3.

*next_element	Points to the next Ft_contents_type_element structure in the linked list. HP-UX responders accept up to three Ft_contents_type_element structures; after three, HP-UX responders ignore the value.  HP-UX responders accept only FTAM-1, FTAM-2, and FTAM-3 document types; if you request any other document type, HP-UX responders remove the element from the response list.
contents_type	Per connection, specify one contents_type for each document type you want to manipulate (transfer, access, or manage).

---

## Ft\_contents\_type

```
struct Ft_contents_type {
    enum    Ft_contents_form    contents_form;
    union   Ft_contents_info    contents_info;
};
```

Ft\_contents\_type specifies the allowable document type for a particular file.

Ft\_contents\_type is the type for the contents\_type field in struct Ft\_contents\_type\_element (a linked list of allowable document types per connection). This structure is also nested within the values field, which is of type struct Ft\_attributes.

**contents\_form** Specifies the contents\_type form as a known or unknown document type.

**contents\_info** Specifies the exact document type.

## Ft\_contents\_form

```
enum    Ft_contents_form {
    FT_DOCUMENT_TYPE           = 0,
    FT_ABS_CON_SET_PAIR_FORM   = 1,
    FT_CONTENTS_UNKNOWN        = 2
};
```

Ft\_contents\_form specifies whether contents\_type is a known or unknown document type.

- For all functions that use the Ft\_contents\_type, (except ft\_fopen() and ft\_open()), you must specify FT\_DOCUMENT\_TYPE.
- HP-UX FTAM does not permit you to use the value FT\_ABS\_CON\_SET\_PAIR\_FORM.
- Specify FT\_CONTENTS\_UNKNOWN on ft\_open() and ft\_fopen() if you do not know the document type; the document type is returned in the inout\_dcb.

## Ft\_contents\_info

```
union Ft_contents_info {
    struct Ft_document_type    document;
    struct Ft_abs_con_set_pair abs_con_set_pair;
};
```

Ft\_contents\_info specifies the document type.

- If you specified FT\_DOCUMENT\_TYPE in Ft\_contents\_form, specify the document.
- If you specified FT\_CONTENTS\_UNKNOWN in Ft\_contents\_form, Ft\_contents\_info is ignored.
- As noted under Ft\_contents\_form, HP-UX responders do not permit use of the abs\_con\_set\_pair structure.

## Ft\_document\_type

```
struct Ft_document_type {
    struct Object_id    name;
    char                *parameters;
};
```

Ft\_document\_type defines the document type structure of the file. For detailed information (e.g., restrictions imposed) on document types, refer to the “Document Types and Constraint Sets” chapter.

**Ft\_contents\_type**

**name** Specifies the name of the document type and is of the following type struct **Object\_id** (from **map.h**).

```
struct Object_id
{
    Uint16      length;
    Sint32      *element; /* List of integers */
};
```

- Set the **length** field to 5 for all document types except NBS-9; for document type NBS-9 set length to 6.

```
struct Ft_contents_type cont_type;
cont_type.contents_info.document.name.length = 5;
```

- Specify the **document\_type** name by setting the element array as follows:

```
1 0 8571 5 1 for FTAM-1 document type
1 0 8571 5 2 for FTAM-2 document type
1 0 8571 5 3 for FTAM-3 document type
1 2 392 10 2 1 for INTAP-1 document type
1 3 14 5 5 9 for NBS-9 document type
```

**EXAMPLE:** This example sets the **Ft\_contents\_type** structure for an FTAM-3 file.

The **document\_type** name for FTAM-3 is 1 0 8571 5 3

```
struct Ft_contents_type          cont_type;
struct Ft_dt_ftam_3             *ftm_3;

cont_type.contents_form = FT_DOCUMENT_TYPE;
ftm_3 = (struct Ft_dt_ftam_3 *)malloc(sizeof(struct Ft_dt_ftam_3));
cont_type.contents_info.document.name.element =
(Sint32 *)malloc(5*sizeof(Sint32));

ftm_3->string_length = 512;
ftm_3->significance = FT_SS_NO_SIGNIFICANCE;
cont_type.contents_info.document.parameters = (char *)ftm_3;

cont_type.contents_info.document.name.length = 5;
cont_type.contents_info.document.name.element[0]=1;
cont_type.contents_info.document.name.element[1]=0;
cont_type.contents_info.document.name.element[2]=8571;
cont_type.contents_info.document.name.element[3]=5;
cont_type.contents_info.document.name.element[4]=3;
```

**\*parameters** Generic pointer to one of the following structures: **Ft\_dt\_ftam\_1**, **Ft\_dt\_ftam\_2**, **Ft\_dt\_ftam\_3**, **Ft\_dt\_intap\_1**, or **Ft\_dt\_nbs\_9**.

### **Ft\_dt\_ftam\_1.**

```
struct Ft_dt_ftam_1
{
    enum Ft_class          class;
    Uint16                string_length;
    enum Ft_string_significance  significance;
};
```

- The class must be FT\_CL\_IA5\_STRING or FT\_CL\_GENERAL\_STRING.
- The string\_length supported by responders must be at least 0 to 134; for HP-UX responders, string\_length is unlimited.
- The significance must be FT\_SS\_NO\_SIGNIFICANCE.

### **Ft\_dt\_ftam\_2.**

```
struct Ft_dt_ftam_2
{
    enum Ft_class          class;
    Uint16                string_length;
    enum Ft_string_significance  significance;
};
```

- The class must be FT\_CL\_GRAPHIC\_STRING.
- The string\_length supported by responders must be at least 0 to 134; for HP-UX responders, string\_length is unlimited.
- The significance must be FT\_SS\_NO\_SIGNIFICANCE.

### **Ft\_dt\_ftam\_3.**

```
struct Ft_dt_ftam_3
{
    Uint16                string_length;
    enum Ft_string_significance  significance;
};
```

- Ft\_dt\_ftam\_3 defaults to FT\_CL\_OCTET\_STRING.
- The string\_length supported by responders must be at least 0 to 512; for HP-UX responders, string\_length is unlimited.
- The significance must be FT\_SS\_NO\_SIGNIFICANCE.

## **Ft\_contents\_type**

### **Ft\_dt\_intap\_1.**

```
struct Ft_dt_intap_1
{
    Uint16          record_length;
    enum Ft_string_significance  significance;
};
```

- Ft\_dt\_intap\_1 defaults to FT\_CL\_OCTET\_STRING.
- The record\_length is unlimited.
- The significance must be FT\_SS\_VARIABLE or FT\_SS\_FIXED.

### **Ft\_dt\_nbs\_9.**

```
struct Ft_dt_nbs_9
{
    Ft_attribute_names  mask;
};
```

- mask is a 32-bit unsigned integer. Use any of the FT\_AN\_xxx defined constants to set bits in the mask. See Ft\_attribute\_names for details.
- This parameter is used only on a call to ft\_open(). An NBS-9 document (a directory) can only be opened for reading. Therefore, any attribute names you supply on the ft\_open() call specify the attributes you will receive for the files in the directory.
- If you omit this parameter, the responder will supply default attributes. HP-UX FTAM responders will return all attributes by default.

---

## Ft\_data\_unit

```

struct Ft_data_unit {
    struct Ft_data_unit    *next;
    enum Ft_structure_id  structure_id;
    union {
        struct Ft_data_element *data_element;
        struct Ft_node_descriptor *node;
    } data;
};

```

### Ft\_data\_unit Is Input To These Functions

ft\_sdata()

### Ft\_data\_unit Is Output From These Functions

ft\_rdata()

Ft\_data\_unit is a linked list of data units (maximum 14 data units per call). Each data unit contains the data that FTAM manipulates (Figure 3-8).

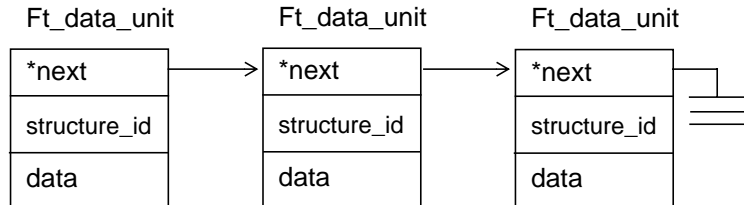
The responder uses Ft\_data\_unit to return data when you call ft\_rdata().

For variable and fixed data, data element boundaries are preserved. You receive the exact sequence of data elements that existed in the original file.

If Ft\_string\_significance is FT\_SS\_NO\_SIGNIFICANCE, data element boundaries are ignored. Up until the final data element, the responder returns the maximum string\_length allowed per data element. The final data element contains the remaining data.

**Figure 3-8** **Ft\_data\_unit Structure**

Up to 14 struct Ft\_data\_unit per call




---

**NOTE** To understand Ft\_data\_unit, you should be familiar with the concepts FADU, data unit, data element, and node descriptor. Refer to the “HP-UX FTAM Overview” chapter for descriptions.

---

- |              |  |
|--------------|--|
| *next        | Points to the next Ft_data_unit structures in the linked list. The maximum is 14.  |
| structure_id | Specifies which structure in the data union is applicable.   |
| data         | <p>The data union contains *data_element and *node. The value you choose in Ft_structure_id dictates which structure you choose in the data union.</p> <ul style="list-style-type: none"> <li>• The *data_element is of type struct Ft_data_element and contains the actual data.</li> <li>• The *node is of type struct Ft_node_descriptor and indicates a new FADU.</li> </ul> |

### **Ft\_structure\_id**

```

enum    Ft_structure_id {
        FT_DATA_UNIT           = 0,
        FT_NODE_DESC           = 1,
        FT_ENTER_SUBTREE       = 2,
        FT_EXIT_SUBTREE        = 3,
        FT_DATA_END_IND        = 4,
        FT_CANCEL_IND          = 5
};
  
```



**Ft\_structure\_id** specifies which structure in the data union is applicable (**Ft\_data\_element** or **Ft\_node\_descriptor**) or it specifies the end or cancellation of data transfer (on output only).

<b>FT_DATA_UNIT</b>	Use <b>FT_DATA_UNIT</b> to send actual data. <b>FT_DATA_UNIT</b> dictates that you choose <b>data_element</b> in the data union.
<b>FT_NODE_DESC</b>	Use <b>FT_NODE_DESC</b> only for FTAM-2 document types to begin a new FADU. <b>FT_NODE_DESC</b> dictates that you choose <b>node</b> in the data union.
<b>FT_ENTER_SUBTREE</b> <b>FT_EXIT_SUBTREE</b>	HP-UX responders return an error if you use either of these values.
<b>FT_DATA_END_IND</b>	When using <b>ft_rdata()</b> , FTAM returns <b>FT_DATA_END_IND</b> in <b>inout_dcb.data_unit-&gt;structure_id</b> to indicate data transmission is complete.
<b>FT_CANCEL_IND</b>	If an error occurs when transferring data, FTAM returns <b>FT_CANCEL_IND</b> in <b>inout_dcb.data_unit-&gt;structure_id</b> to indicate data transmission is cancelled. To acknowledge the data transfer was cancelled, you must issue <b>ft_rcancel()</b> to exit the Data Transfer regime.

## **Ft\_data\_element**

```

struct Ft_data_element {
    enum Ft_prim_type {
        FT_DE_BOOLEAN           = 1,
        FT_DE_INTEGER           = 2,
        FT_DE_BIT_STRING        = 3,
        FT_DE_OCTET_STRING      = 4,
        FT_DE_NULL               = 5,
        FT_DE_IA5_STRING         = 22,
        FT_DE_UTC                = 23,
        FT_DE_TIME               = 24,
        FT_DE_GRAPHIC_STRING     = 25,
        FT_DE_VISIBLE_STRING     = 26,
        FT_DE_GENERAL_STRING     = 27,
        FT_DE_FLOATING_POINT     = 100,
        FT_DE_UL_FLOAT           = 101,
        FT_DE_UL_INTEGER         = 102,
        FT_DE_ATTRIBUTES         = 200,
        FT_DE_RECORD_END        = 201,
        FT_DE_RECORD_CONT       = 202
    } prim_type;
}

```

**Ft\_data\_unit**

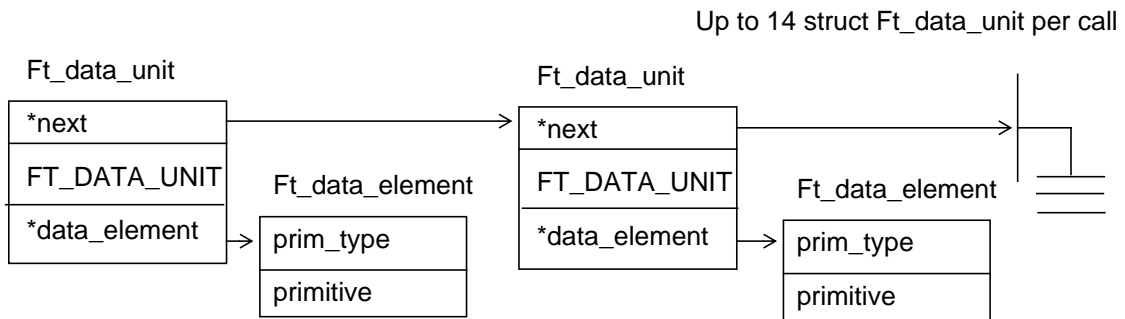
```

union Ft_primitive {
    Bool                boolean;
    Sint32              integer;
    struct Bit_string   bit_string;
    struct Octet_string octet_string;
    struct Octet_string ia5string;
    Time                utc;
    Time                time;
    char                *graphic_string;
    char                *visible_string;
    struct Octet_string general_string;
    double              floating_point;
    struct Octet_string ul_integer;
    struct Ft_ul_floating_point {
        Sint16          sign;
        struct Bit_string mantissa;
        struct Octet_string exponent;
    } ul_floating_point;
    struct Ft_attributes attributes;
    struct Octet_string record_end;
    struct Octet_string record_cont;
} primitive;
};
    
```

Set `Ft_data_element` if you specified `FT_DATA_UNIT` in `Ft_structure_id`.

`Ft_data_element` contains an enum `Ft_prim_type` that indicates which primitive is relevant in union `Ft_primitive` (Figure 3-9).

**Figure 3-9** **Ft\_data\_element Structure**



**prim\_type** The `prim_type` specifies which field in the primitive union to select. Only a subset is valid for each document type:

FTAM-1	FTAM-2	FTAM-3 and INTAP-1	NBS-9
IA5_STRING	GRAPHIC_STRING	OCTET_STRING	ATTRIBUTE
GENERAL_STRING	GENERAL_STRING		
GRAPHIC_STRING			
GENERAL_STRING			

**primitive** The primitive is the specific unit that contains actual data. Set primitive to the value specified by prim\_type. See the previous table. Note that the maximum string length is 7168 (7K) octets.

**EXAMPLE:** This example sets a data\_element for FTAM-2 document type.

```

struct Ft_data_unit          data_unit;

data_unit.next = NULL;
data_unit.structure_id = FT_DATA_UNIT;
data_unit.data.data_element = (struct Ft_data_element *)
malloc(sizeof (struct Ft_data_element));

data_unit.data.data_element->prim_type = FT_DE_GRAPHIC_STRING;
data_unit.data.data_element->primitive.graphic_string =
"This example shows data in a data element;"

```

## Ft\_node\_descriptor

```

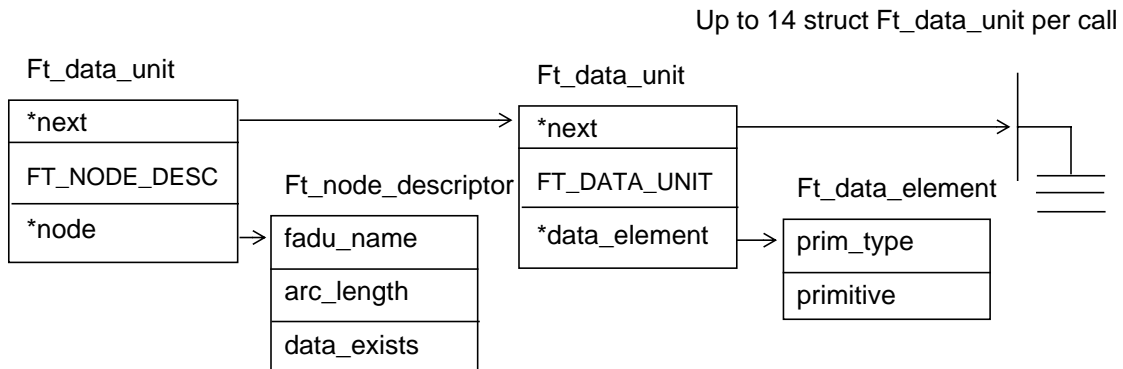
struct Ft_node_descriptor {
    Ft_fadu_name      fadu_name;
    Uint16           arc_length;
    Bool             data_exists;
};

```

Set Ft\_node\_descriptor if you specified FT\_NODE\_DESC in Ft\_structure\_id (Figure 3-10).

Use Ft\_node\_descriptor only with FTAM-2 document types to begin a new FADU.

**Figure 3-10 Ft\_node\_descriptor Structure**



- fadu\_name** HP-UX responders accept, but ignore, the fadu\_name field. HP recommends you set the fadu\_name.length to zero and fadu\_name.pointer to NULL.
- arc\_length** You must set arc\_length to 1 since FTAM-2 is confined by the sequential flat constraint set (which has only one hierarchical level).
- data\_exists** You must set data\_exists to TRUE since all FADUs must contain data.
- EXAMPLE:** This example sets the node\_descriptor for an FTAM-2 document type.

```

struct Ft_data_unit          data_unit;
data_unit.next = NULL;
data_unit.structure_id = FT_NODE_DESC;
data_unit.data_node = (struct Ft_node_descriptor *)
    malloc(sizeof (struct Ft_node_descriptor));
data_unit.data.node->fadu_name.length = 0;
data_unit.data.node->fadu_name.pointer = NULL;
data_unit.data.node->arc_length = 1;
data_unit.data.node->data_exists = TRUE;
  
```

---

**NOTE** Creating the Ft\_data\_unit linked lists for FTAM-2 document types requires that you follow a few rules. First, familiarize yourself with the Ft\_node\_descriptor structure. Then read the next section, which describes the necessary rules.

---

## Rules for Ft\_data\_unit Linked Lists (FTAM-2 Only)

The following rules apply to Ft\_data\_unit linked lists for FTAM-2 document types only.

- If you have an empty file, you must begin the linked list with a node descriptor (not a data element).
- If a file already contains data, you may or may not have a node descriptor, depending on whether you are starting a new FADU.
- Do not immediately follow a node descriptor with another node descriptor.

You may, however, follow a data element with another data element (i.e., node descriptors may have multiple data elements).

- You may submit an ft\_sdata() request containing only a node descriptor, though you must follow it with an ft\_sdata() request that starts with a data element.
- Do not use more than one node descriptor per ft\_sdata() request.
- You must set node.arc\_length to 1.
- You must set data\_exists to TRUE since FADUs must contain data.

---

## Ft\_dcb\_type

```
enum    Ft_dcb_type {
        FTiAactivation          = 0,
        FToAactivation          = 1,
        FTiAdeactivation        = 2,
        FToAdeactivation        = 3,
        FTiConnect              = 4,
        FToConnect              = 5,
        FTiRelease              = 6,
        FToRelease              = 7,
        FTiAbort                = 8,
        FToAbort                = 9,
        FTiAereset              = 10,
        FToAereset              = 11,
        FTiIreceive             = 12,
        FToIreceive             = 13,
        FTiNwcleared            = 99,
        FTiFCopy                = 100,
        FToFCopy                = 101,
        FTiFMove                = 102,
        FToFMove                = 103,
        FTiFDelete              = 104,
        FToFDelete              = 105,
        FTiFRattributes         = 106,
        FToFRattributes         = 107,
        FTiFCattributes         = 108,
        FToFCattributes         = 109,
        FTiFOpen                = 110,
        FToFOpen                = 111,
        FTiFClose               = 112,
        FToFClose               = 113,
        FTiSelect               = 200,
        FToSelect               = 201,
        FTiDeselect             = 202,
        FToDeselect             = 203,
        FTiOpen                 = 204,
        FToOpen                 = 205,
        FTiClose                = 206,
        FToClose                = 207,
        FTiCreate                = 208,
        FToCreate                = 209,
        FTiDelete               = 210,
        FToDelete               = 211,
        FTiReadattributes       = 212,
        FToReadattributes       = 213,
        FTiChangeattributes     = 214,
        FToChangeattributes     = 215,
        FTiRead                 = 216,
        FToRead                 = 217,
        FTiWrite                 = 218,
```

The Ft\_dcb\_type data type is continued on the next page.

This Ft\_dcb\_type data type is continued from the previous page.

```

        FTiWrite           = 219,
        FTiErase           = 220,
        FTiErase           = 221,
        FTiLocate          = 222,
        FTiLocate          = 223,
        FTiSData           = 224,
        FTiSData           = 225,
        FTiRData           = 226,
        FTiRData           = 227,
        FTiDataEnd         = 228,
        FTiDataEnd         = 229,
        FTiTransEnd        = 230,
        FTiTransEnd        = 231,
        FTiBGroup          = 232,
        FTiBGroup          = 233,
        FTiEGroup          = 234,
        FTiEGroup          = 235,
        FTiCancel          = 236,
        FTiCancel          = 237,
        FTiRCancel         = 238,
        FTiRCancel         = 239
};

```

**Ft\_dcb\_type Is Input To These  
Functions**

ft\_didcb()

**Ft\_dcb\_type Is Output From  
These Functions**

None

Ft\_dcb\_type identifies the type of data control block (DCB). The ft\_didcb() function then initializes and allocates memory for the fixed parameters within the specified DCB.

- A lowercase i after FT indicates an input\_dcb.
- A lowercase o after FT indicates an inout\_dcb.

## Ft\_delete\_action

```
enum    Ft_delete_action {
                                FT_DELETE_FILE      = 0,
                                FT_DONT_DELETE_FILE  = 1
};
```

### **Ft\_delete\_action Is Input To These Functions**

ft\_fclose()

### **Ft\_delete\_action Is Output From These Functions**

None

Ft\_delete\_action specifies whether to delete a file or only deselect it on an ft\_fclose() function.

FT\_DELETE\_FILE

Closes, deselects, and then deletes the file.

FT\_DONT\_DELETE\_FILE

Closes and deselects, but does not delete the file.



---

## Ft\_delete\_overwrite

```
enum    Ft_delete_overwrite {
                                FT_RECREATE_FILE      = 0,
                                FT_DONT_RECREATE_FILE  = 1
};
```

<b>Ft_delete_overwrite Is Input To These Functions</b>	<b>Ft_delete_overwrite Is Output From These Functions</b>
--	---

ft_fcopy()	None
ft_fmove()	

Ft\_delete\_overwrite specifies whether or not to copy over (ft\_fcopy()) or move over (ft\_fmove()) an existing file.

FT_RECREATE_FILE	Copy or move the source file over the existing destination file.
FT_DONT_RECREATE_FILE	Do not copy or move the source file over the existing destination file; the function fails and you receive an error.

**Ft\_diagnostic****Ft\_diagnostic**

```

struct Ft_diagnostic {
    struct Ft_diagnostic *next;
    enum Ft_diag_type    diag_type;
    Uint16               error_id;
    enum Ft_entity_ref   error_observer;
    enum Ft_entity_ref   error_source;
    Uint16               suggested_delay;
    char                 *further_details;
};

```

**Ft\_diagnostic Is Input To  
These Functions**

ft\_abort()  
ft\_cancel()  
ft\_edata()  
ft\_rcancel()

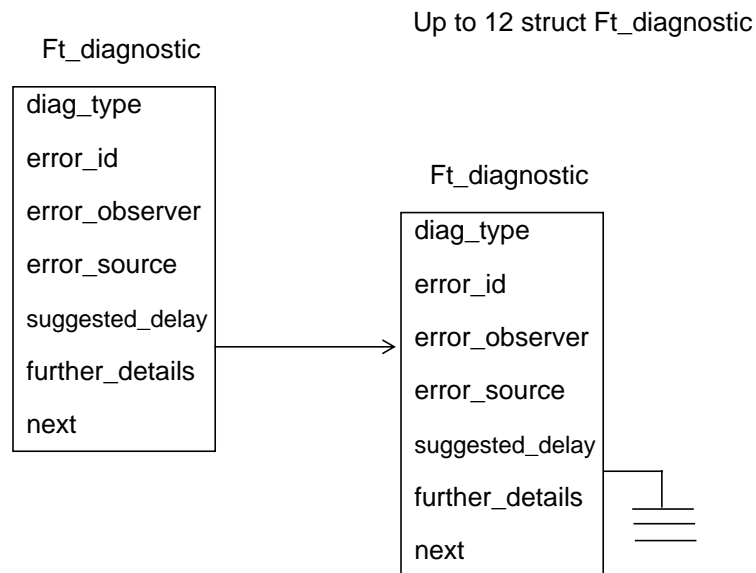
**Ft\_diagnostic Is Output  
From These Functions**

ft\_cattributes()  
ft\_close()  
ft\_connect()  
ft\_create()  
ft\_delete()  
ft\_deselect()  
ft\_edata()  
ft\_erase()  
ft\_etransfer()  
ft\_fattributes()  
ft\_fclose()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fmove()  
ft\_fopen()  
ft\_frattributes()  
ft\_ireceive()  
ft\_locate()  
ft\_open()  
ft\_rattributes()  
ft\_sdata()  
ft\_select()  
ft\_rdata()

Ft\_diagnostic provides additional information for ISO-specific errors.

The `further_details` field may vary across different vendor responders. Refer to the *HP FTAM/9000 Reference Manual* for a list of HP-UX diagnostic messages, their probable causes, and possible corrective actions.

**Figure 3-11 Ft\_diagnostic Structure**



- `*next`                      Linked list of `Ft_diagnostic` structures; maximum number is 12.
- `diag_type`                Specifies the kind of ISO error that occurred.
- `error_id`                   Specifies the integer representing the ISO error that occurred; defined in `f_error.h`.
- `error_observer`          Specifies the entity that detected the error.

**Ft\_diagnostic**


---

<b>error_source</b>	Specifies the entity in which the error occurred.
<b>suggested_delay</b>	Specifies the amount of time to wait before attempting recovery if the <b>Ft_diag_type</b> is <b>FT_TRANSIENT</b> . HP-UX responders do not use this field; they never return a <b>diag_type</b> of <b>FT_TRANSIENT</b> .
<b>*further_details</b>	Provides additional information (a text message) about the cause of the error. This information may vary among different vendor responders.

**Ft\_diag\_type**

```
enum    Ft_diag_type {
        FT_INFORMATIVE      = 0,
        FT_TRANSIENT        = 1,
        FT_PERMANENT        = 2
    };
```

**Ft\_diag\_type** specifies the kind of ISO error that occurred and is the enumeration for the **diag\_type** field, which is of type struct **Ft\_diagnostic**.

<b>FT_INFORMATIVE</b>	The error does not affect the current file service and therefore, does not require recovery. An example is <b>F_CREATE_SLCTD_EXIST_FILE</b> , which means override selected an existing file.
<b>FT_TRANSIENT</b>	The current file service is affected. It requires recovery.
<b>FT_PERMANENT</b>	The FTAM request failed.

**Ft\_entity\_ref**

```
enum    Ft_entity_ref {
        FT_NOT_CATEGORIZED          = 0,
        FT_INITIATING_FILE_SERVICE_USER      = 1,
        FT_INITIATING_FILE_PROTOCOL_MACHINE  = 2,
        FT_SERVICE_SUPPORTING_FILE_PROTOCOL_MACHINE  = 3,
        FT_RESPONDING_FILE_PROTOCOL_MACHINE  = 4,
        FT_RESPONDING_FILE_SERVICE_USER      = 5
    };
```

**Ft\_entity\_ref** identifies the entity that detected the error (**error\_observer**) and where the error occurred (**error\_source**). You may find this information particularly helpful if the error occurs with other vendors.

FT_NOT_CATEGORIZED	The error may have occurred or been detected anywhere.
FT_INITIATING_FILE_SERVICE_USER	The error is in the user program.
FT_INITIATING_FILE_PROTOCOL_- MACHINE	The error is in or detected by the initiator.
FT_SERVICE_-SUPPORTING_FILE_- PROTOCOL_MACHINE	The error is in one of the lower layers, probably the Presentation Service.
FT_RESPONDING_FILE_PROTOCOL_- MACHINE	The error is in or detected by the responder.
FT_RESPONDING_FILE_SERVICE_- USER	The error is in or detected by the VFS.

**Ft\_fadu\_identity**

---

**Ft\_fadu\_identity**

```

struct Ft_fadu_identity {
    enum Ft_fadu_form      fadu_form;
    union Ft_fadu_info     fadu_info;
};

```

**Ft\_fadu\_identity Is Input To  
These Functions**

ft\_erase()  
ft\_locate()  
ft\_read()  
ft\_write()

**Ft\_fadu\_identity Is Output  
From These Functions**

ft\_locate()

**Ft\_fadu\_identity** identifies the whole file (the single FADU) for FTAM-1, FTAM-3, and INTAP-1 files. This structure also identifies either the whole file or individual FADUs for FTAM-2 files. Use the identified FADUs to erase, read, and write (FTAM-1, FTAM-2, and FTAM-3) and to locate (FTAM-2 only).

**fadu\_form** The **fadu\_form** indicates you will use either a FADU location or number. *For HP-UX responders, you must use location.*

**fadu\_info** The **fadu\_info** specifies the FADU location or number.

**Ft\_fadu\_form**

```

enum Ft_fadu_form {
    FT_FADU_LOCATION      = 0,
    FT_FADU_NAME          = 1,
    FT_FADU_NAME_LIST     = 2,
    FT_FADU_NUMBER        = 3
};

```

**Ft\_fadu\_form** is the type for the **fadu\_form** field in **Ft\_fadu\_identity**.

- The value specified in `fadu_form` dictates the value used in `fadu_info`.
- For HP-UX responders, you must specify `FT_FADU_LOCATION`. HP-UX responders return an error if you use `FT_FADU_NAME`, `FT_NAME_LIST`, or `FT_NUMBER`.

## **Ft\_fadu\_info**

```
union Ft_fadu_info {
    enum      Ft_fadu_location   fadu_location;
    Ft_fadu_name fadu_name;
    struct    Ft_fadu_name_list  fadu_name_list;
    Sint32    fadu_number;
};
```

`Ft_fadu_info` is the type for `fadu_info` in struct `Ft_fadu_identity` and is also the union for the enum `Ft_fadu_form`.

`Ft_fadu_info` specifies the actual `fadu_identity` location.

**fadu\_location** Specifies the FADU location you want to access within a file. If you selected `FT_FADU_LOCATION` in `Ft_fadu_form`, you must specify `fadu_location`.

- The `fadu_location` is of the type `Ft_fadu_location`.

```
enum      Ft_fadu_location {
    FT_FIRST      = 0,
    FT_LAST       = 1,
    FT_PREVIOUS   = 2,
    FT_CURRENT    = 3,
    FT_NEXT       = 4,
    FT_BEGIN      = 5,
    FT_END        = 6
};
```

**FT\_FIRST** First FADU in the file.

**FT\_LAST \*** Last FADU in the file.

**FT\_PREVIOUS\*** The FADU immediately before the FADU where the file pointer is currently located.

**FT\_CURRENT \*** The FADU where the file pointer is currently located.

**Ft\_fadu\_identity**

---

FT_NEXT	The FADU immediately after the FADU where the file pointer is currently located.
FT_BEGIN	For ft_read(), ft_write(), and ft_erase(), initial location in the file, before all FADUs. For ft_locate(), earliest FADU in the file.
FT_END	The final location in the file, after all FADUs.

\*Invalid for use with HP-UX responders.

- Use only FT\_FIRST for FTAM-1, FTAM-3, INTAP-1, and NBS-9 document types (constrained by the Unstructured All constraint set).
- Use any of the fields for FTAM-2 document type (constrained by the Sequential Flat constraint set).
- You can locate, read, write, and erase the whole file or leaf FADUs as defined in the following table. Remember, only FTAM-2 files have leaf FADUs.
- For example, you can locate any of the individual FADUs, but you can only erase the whole file.

	<b>ft_locate()</b>	<b>ft_read()</b>	<b>ft_write()</b>	<b>ft_erase()</b>
Begin	Leaf FADU	Whole File		Whole File
End	End of File		End of File	
First	Leaf FADU	Leaf FADU		
Last	Leaf FADU	Leaf FADU		
Current	Leaf FADU	Leaf FADU		
Next	Leaf FADU	Leaf FADU		
Previous	Leaf FADU	Leaf FADU		

fadu\_name  
fadu\_name\_list  
fadu\_number

HP-UX responders return an error if you use these fields.



---

## Ft\_fadu\_operation

```
enum    Ft_fadu_operation {
        FT_INSERT      = 0,
        FT_REPLACE     = 1,
        FT_EXTEND      = 2
    };
```

### Ft\_fadu\_operation Is Input To These Functions

ft\_write()

### Ft\_fadu\_operation Is Output From These Functions

None

Ft\_fadu\_operation specifies the action you are taking on a FADU.

FT_INSERT	Inserts data at the end of a file ( <i>FTAM-2 only</i> ).
FT_REPLACE	Replaces the existing FADU contents ( <i>FTAM-1, FTAM-3, and INTAP-1 only</i> ).
FT_EXTEND	Appends data to the end of the existing FADU ( <i>FTAM-1, FTAM-3, and INTAP-1 only</i> ).

---

## Ft\_file\_actions

```
typedef Uint16 Ft_file_actions;
```

<b>Ft_file_actions Is Input To These Functions</b>	<b>Ft_file_actions Is Output From These Functions</b>
ft_create()	None
ft_fopen()	
ft_select()	

If you have access control, Ft\_file\_actions specifies the allowable actions the designated users can perform on a file. If you do not have access control, Ft\_file\_actions specifies the requested actions on the file.

Ft\_file\_actions is the type for the action\_list field in struct Ft\_access\_control\_elements (defines who can access a file and how they can access it). Ft\_file\_actions is also the type of the requested\_access parameter in the File Selection regime.

- Use the FT\_FA\_XXX defined constants in the mapftam.h header file.

```
FT_FA_READ  
FT_FA_INSERT  
FT_FA_REPLACE  
FT_FA_EXTEND  
FT_FA_ERASE  
FT_FA_READ_ATTRIBUTE  
FT_FA_CHANGE_ATTRIBUTE  
FT_FA_DELETE_FILE
```

FT_FA_READ	Reads contents of the file (FTAM-1, FTAM-3, and INTAP-1) or of the FADU (FTAM-2).
FT_FA_INSERT	Inserts data at the end of a file (FTAM-2 only).
FT_FA_REPLACE	Replaces the existing FADU contents (FTAM-1, FTAM-3, and INTAP-1 only).
FT_FA_EXTEND	Appends data to the end of the existing FADU (FTAM-1, FTAM-3, and INTAP-1 only).
FT_FA_ERASE	Erases the entire file.
FT_FA_READ_ATTRIBUTE	Reads the file attributes.
FT_FA_CHANGE_ATTRIBUTE	Changes the file attributes.D
FT_FA_DELETE_FILE	Deletes the entire file.

---

## Ft\_file\_passwords

```
struct Ft_file_passwords {
    Ft_single_file_pw  read;
    Ft_single_file_pw  insert;
    Ft_single_file_pw  replace;
    Ft_single_file_pw  extend;
    Ft_single_file_pw  erase;
    Ft_single_file_pw  read_attribute;
    Ft_single_file_pw  change_attribute;
    Ft_single_file_pw  delete_file;
};
```

**Ft\_file\_passwords**

<b>Ft_file_passwords Is Input To These Functions</b>	<b>Ft_file_passwords Is Output From These Functions</b>
ft_create()	None
ft_fcopy()	
ft_fmove()	
ft_fcattributes()	
ft_fopen()	
ft_frattributes()	
ft_select()	

---

**NOTE**

---

Since the HP-UX file system does not define file passwords, HP-UX FTAM responders do not use file password information.

Ft\_file\_passwords specifies a user password for each action. To use file passwords, first negotiate security in Ft\_attribute\_groups during the connection establishment. You must use file\_passwords for each action you request if the following conditions are true:

- The file has an access\_control\_element with an identity field that specifies you.
- The action listed in actions\_list has an associated password.

---

## Ft\_file\_status

```
enum    Ft_file_status {
        FT_FS_ENUM_DEFAULT      = -1,
        FT_NEW                   = 0,
        FT_OLD                   = 1,
        FT_REPLACE_CONTENTS     = 2,
        FT_RECREATE              = 3,
        FT_UNKNOWN               = 4
};
```

### Ft\_file\_status Is Input To These Functions

ft\_create()

ft\_fopen()

### Ft\_file\_status Is Output From These Functions

None

Ft\_file\_status determines what action to take in the file specified on ft\_create() or ft\_fopen() already exists. Ft\_file\_status specifies whether to create or select a file and the effects these actions have on the already existing file.

If the file does not exist, a new file is created.

### Ft\_file\_status Field

FT\_FS\_ENUM\_DEFAULT

FT\_NEW

FT\_OLD

### If The File Already Exists

For ft\_create(), request fails

For ft\_fopen(), existing file selected

Request fails

Existing file selected

**Ft\_file\_status**

---

<b>FT_REPLACE_CONTENTS</b>	Existing file contents deleted Existing file attributes remain the same Existing file selected
<b>FT_RECREATE</b>	Existing file contents and attributes deleted New file created with attributes specified in struct Ft_attributes
<b>FT_UNKNOWN</b>	Existing file selected

---

## Ft\_filename

```
typedef char      *Ft_filename;
```

<b>Ft_filename s Input To These Functions</b>	<b>Ft_filename Is Output From These Functions</b>
ft_create()	None
ft_fattributes()	
ft_fcopy()	
ft_fdelete()	
ft_fmove()	
ft_fopen()	
ft_frattributes()	

Ft\_filename identifies a specific file with a unique path name given in the syntax of the original file in the real filestore.

---

## Ft\_functional\_units

```
typedef Uint16    Ft_functional_units;
```

### **Ft\_functional\_units Input To These Functions**

ft\_connect()

### **Ft\_functional\_units Is Output From These Functions**

ft\_connect()

Ft\_functional\_units specifies which functions are available per connection. You must use the FT\_FU\_XXX defined constants to specify Ft\_functional\_units.

```
FT_FU_READ
FT_FU_WRITE
FT_FU_FILE_ACCESS
FT_FU_LTD_MGMT(Limited File Management)
FT_FU_ENH_MGMT(Enhanced File Management)
FT_FU_GROUPING
FT_FU_FADU_LOCKING
FT_FU_RECOVERY
FT_FU_RESTART_XFR
```

- HP-UX responders accept, but ignore, FT\_FU\_FADU\_LOCKING, FT\_FU\_RECOVERY, and FT\_FU\_RESTART\_XFR.
- By default, you always have the following functions.

ft_aeactivation()	ft_deselect()
ft_deactivation()	ft_select()
ft_ireceive()	ft_aereset()
ft_didcb()	ft_abort()
ft_dfdbc()	ft_release()
ft_gperror()	ft_connect()



- You must negotiate Ft\_functional\_units on the ft\_connect() request. For example, if you want to be able to read a file, set FT\_FU\_READ when making the ft\_connect() request.
- Ft\_service\_class dictates the mandatory and optional functional\_units (as noted in the following table). Ft\_service\_class must support all selected Ft\_functional\_unit values; otherwise, you receive an error. Refer to the “Ft\_service\_class” section for detailed information.

EXAMPLE:        On ft\_connect(), if you specify the  
FT\_SC\_TRANSFER service\_class,

- You must specify FT\_FU\_GROUPING functional\_units,
- You must specify either or both FT\_FU\_READ and FT\_FU\_WRITE,
- You may optionally specify either or both FT\_FU\_LTD\_MGMT and FT\_FU\_ENH\_MGMT, and
- You cannot specify FT\_FU\_FILE\_ACCESS.
- The kernel functional\_unit is always available.

The following table shows the service\_classes you should set to use the functional\_units. Use this key to read the table.

TSC=File Transfer Service Class

ASC=File Access Service Class

MSC=File Management Service Class

TMSC=File Transfer and Management Service Class

USC=Unconstrained Service Class

M = Mandatory

O = Optional

• = One or Both

X = Not Permitted

HP FTAM/9000 Data Structures  
**Ft\_functional\_units**

See the previous page for the reference key to this table.

<b>Functional Unit</b>	<b>FTAM Functions</b>	<b>TSC</b>	<b>ASC</b>	<b>MSC</b>	<b>TMSC</b>	<b>USC</b>
Kernel	ft_abort	M	M	M	M	M
	ft_aeactivation					
	ft_aedeactivation					
	ft_aereset					
	ft_connect					
	ft_deselect					
	ft_ireceive					
	ft_rrequest					
	ft_select					
	Read	ft_cancel	•	M	X	•
ft_close						
ft_ettransfer						
ft_open						
ft_rcancel						
ft_rdata						
ft_rdataqos						
ft_read						
Write	ft_cancel	•	M	X	•	O
	ft_close					
	ft_edata					
	ft_ettransfer					
	ft_nwcleared					
	ft_rcancel					
	ft_sdata					
	ft_open					
	ft_sdata					
	ft_write					
File Access	ft_erase	X	M	X	X	O
	ft_locate					
Limited File Management	ft_create	O	O	M	M	O
	ft_delete					
	ft_rattributes					
Enhanced File Management Grouping	ft_cattributes	O	O	O	O	O
	ft_bgroup	M	O	M	M	O
	ft_egroup					

---

## Ft\_initiator\_identity

```
typedef char      *Ft_initiator_identity;
```

### Ft\_initiator\_identity Is Input To These Functions

ft\_connect()  
ft\_fdelete()  
ft\_fcattributes()  
ft\_fcopy()  
ft\_fmove()  
ft\_frattributes()

### Ft\_initiator\_identity Is Output From These Functions

None

Ft\_initiator\_identity identifies the HP-UX login account if using HP-UX responders. This structure is the type for the following fields and specifies the following user identities.

identity_of_creator	User who created the file
identity_of_modifier*	User who last modified the file
identity_of_reader*	User who last read the file
identity_of_attribute_mod*	User who last modified file attributes
initiator_identity*	User's login account
	In Ft_access_control_element, user who has the access privileges

\* HP-UX responders accept, but ignore, these values.

---

## Ft\_processing\_mode

```
typedef Uint16 Ft_processing_mode;
```

<b>Ft_processing_mode Input To These Functions</b>	<b>Ft_processing_mode Is Output From These Functions</b>
ft_open()	None

Ft\_processing\_mode specifies the current file action for ft\_open () (i.e., specifies an action you want to perform now).

- These actions must be a subset of those actions selected in requested\_access (FT\_FA\_XXX defined constants).
- Only after closing the file can you specify other actions through Ft\_processing\_mode (if you first re-open the file).
- You must use one of the following FT\_PM\_XXX defined constants to specify Ft\_processing\_mode:

FT_PM_READ	Reads contents of the file ( <i>FTAM-1, FTAM-3, INTAP-1, and NBS-9</i> ) or of the FADU ( <i>FTAM-2</i> ).
FT_PM_INSERT	Inserts data at the end of a file ( <i>FTAM-2 only</i> ).
FT_PM_REPLACE	Replaces the existing FADU contents ( <i>FTAM-1, FTAM-3, and INTAP-1 only</i> ).
FT_PM_EXTEND	Appends data to the end of the existing FADU ( <i>FTAM-1, FTAM-3, and INTAP-1 only</i> ).
FT_PM_ERASE	Erases the entire file. ( <i>FTAM-1, FTAM-2, FTAM-3, and INTAP-1 only</i> ).

---

## Ft\_qos

```
enum    Ft_qos  {
                FT_NO_RECOVERY      = 0,
                FT_CLASS_1_RECOVERY = 1,
                FT_CLASS_2_RECOVERY = 2,
                FT_CLASS_3_RECOVERY = 3
};
```

### Ft\_qos Is Input To These Functions

ft\_connect()

### Ft\_qos Is Output From These Functions

ft\_connect()

Ft\_qos determines the level of recovery by FTAM by specifying which recovery classes are active. The qos stands for quality of service.

FT_NO_RECOVERY	FTAM does not automatically attempt recovery. You must select FT_NO_RECOVERY and perform the recovery.
FT_CLASS_1_RECOVERY	Transient errors occurring during data transfer are recovered by restarting the data transfer.
FT_CLASS_2_RECOVERY	Transient errors occurring during data transfer are recovered by restarting the select/open regime.
FT_CLASS_3_RECOVERY	Not supported. If this class is selected, you will receive the error FTE 173_INV_FTQOS.

## Ft\_service\_class

```
typedef Uint16    Ft_service_class;
```

### **Ft\_service\_class Is Input To These Functions**

ft\_connect()

### **Ft\_service\_class Is Output From These Functions**

ft\_connect()

Ft\_service\_class specifies whether you transfer, access, or manage a file per connection. You must use one of the following FT\_SC\_XXX defined constants to specify Ft\_service\_class:

<b>Service Class</b>	<b>Description</b>
Unconstrained FT_SC_UNCONSTRAINED	Allows you to request any combination of functional_units with the mandatory kernel group
File Management FT_SC_MANAGEMENT	Allows you to manipulate whole files Change attributes Create files Delete files Read attributes

---

File Transfer FT_SC_TRANSFER	Allows you to transfer data
File Transfer and Management FT_SC_TRANSFER_AND_MGMT	Allows you to move or copy bulk data Allows you to manipulate whole files: Change attributes Create files Delete files Read attributes Combines all capabilities of File Transfer and File Management
File Access FT_SC_ACCESS	Allows you to transfer data Provides you with access to individual FADUs

You can use any of the following combinations of service\_classes. Note, you cannot use FT\_SC\_UNCONSTRAINED by itself; you must use it in conjunction with other service\_classes.

- FT\_SC\_MANAGEMENT
- FT\_SC\_TRANSFER
- FT\_SC\_ACCESS
- FT\_SC\_TRANSFER and FT\_SC\_ACCESS
- FT\_SC\_MANAGEMENT, FT\_SC\_TRANSFER, and FT\_SC\_TRANSFER\_AND\_MGMT
- FT\_SC\_MANAGEMENT, FT\_SC\_TRANSFER, FT\_SC\_ACCESS, and FT\_SC\_TRANSFER\_AND\_MGT
- FT\_SC\_UNCONSTRAINED and FT\_SC\_MANAGEMENT
- FT\_SC\_UNCONSTRAINED and FT\_SC\_TRANSFER
- FT\_SC\_UNCONSTRAINED and FT\_SC\_ACCESS

**Ft\_service\_class**

- FT\_SC\_UNCONSTRAINED, FT\_SC\_TRANSFER and FT\_SC\_ACCESS
- FT\_SC\_UNCONSTRAINED, FT\_SC\_MANAGEMENT, FT\_SC\_TRANSFER, and FT\_SC\_TRANSFER\_AND\_MGMT
- FT\_SC\_UNCONSTRAINED, FT\_SC\_MANAGEMENT, FT\_SC\_TRANSFER, FT\_SC\_ACCESS, and FT\_SC\_TRANSFER\_AND\_MGT
- You must negotiate Ft\_service\_class when calling ft\_connect().
- Ft\_service\_class defines the mandatory and optional Ft\_functional\_units. Set these restrictions using the defined constants. Each FT\_SC\_XXX dictates which FT\_FU\_XXX (Ft\_functional\_unit) defined constants are mandatory and optional (as noted in the following table).

**EXAMPLE:**

On ft\_connect(), if you specify the FT\_SC\_TRANSFER\_AND\_MGT service\_class,

- You must specify FT\_FU\_GROUPING and FT\_FU\_LTD\_MGMT functional\_units,
- You must specify either or both FT\_FU\_READ and FT\_FU\_WRITE,
- You can optionally specify FT\_FU\_ENH\_MGMT, and
- You cannot specify FT\_FU\_FILE ACCESS.



<b>Service Class</b>	<b>Mandatory Functional Units</b>	<b>One or Both Functional Units</b>	<b>Optional Functional Units</b>	<b>Not Allowed</b>
Unconstrained	• Kernel		• Enhanced File Mgmt • File Access • Group • Limited File Mgmt • Read • Write	
Management	• Kernel • Grouping • Limited File Management		• Enhanced File Mgmt	• File Access • Read • Write
Transfer	• Kernel • Grouping	• Read • Write	• Enhanced File Mgmt • Limited File Mgmt	• File Access
Transfer and Management	• Kernel • Grouping • Limited File Mgmt	• Read • Write	• Enhanced File Mgmt	• File Access
Access	• Kernel • File Access • Read • Write		• Enhanced File Mgmt • Grouping • Limited File Mgmt	

Ft\_service\_class restricts the way in which you can group functions (with ft\_egroup()). Refer to the following table and list to determine these restrictions.

- If using FT\_SC\_TRANSFER, open the file with group #1, transfer data as needed, and close the file with group #2.
- If using FT\_SC\_MANAGEMENT, you must use group #3 and you cannot call low level functions outside of this group.

**Ft\_service\_class**

- If using FT\_SC\_TRANSFER\_AND\_MGMT, you must use group #1 and #2, or use group #3, as follows.
  - Open the file with group #1, transfer data as needed, and close the file with group #2 OR
  - Use group #3 to select or create a file, read or change attributes, and deselect or delete the file.
- If you select FT\_SC\_ACCESS, you can use any of the groupings.

<b>Mandatory Beginning Functions</b>	<b>Must Have One of these Functions</b>	<b>Optional Functions</b>	<b>Must Have One of these Functions</b>	<b>Mandatory Ending Functions</b>
#1 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()	ft_open()	ft_egroup()
#2 ft_bgroup()	ft_close()	ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()
#3 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()
#4 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()		ft_egroup()
#5 ft_bgroup()		ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()

---

## **Ft\_single\_file\_pw**

```
typedef struct Octet_string      Ft_single_file_pw;
```

### **Ft\_single\_file\_pw s Input To These Functions**

```
ft_connect()
ft_create()
ft_fattributes()
ft_fcopy()
ft_fdelete()
ft_fmove()
ft_frattributes()
```

### **Ft\_single\_file\_pw Is Output From These Functions**

None

Ft\_single\_file\_pw is of type struct Octet\_string which contains a length field (Uint16) and a pointer field (Octet\_pointer). The meaning of Ft\_single\_file\_pw varies as noted in the following table.

<b>Function</b>	<b>Ft_single_file_pw</b>	<b>Description</b>
ft_connect() ft_fattributes() ft_fdelete() ft_frattributes()	file_store_pw	Password for initiator_identity.
ft_create() fcopy() ft_fmove()	create_file_pw	Password to establish permission to create files in the filestore; HP-UX responders accept, but ignore, this value.
ft_fcopy() ft_fmove()	source_filestore_p wdest_filestore_pw	Password for source_init_id. Password for dest_init_id.
ft_fdelete()	delete_file_pw	Password to establish permission to delete files in the filestore. HP-UX responders accept, but ignore, this value.

---

## inout\_dcb

You receive large amounts of data from the FTAM interface. To reduce the number of exposed parameters, `Ft_output` and `Ft_xxx_out_dcb` contain a group of parameters. The `xxx` varies, depending on the function (e.g., `Ft_rdata_out_dcb`). Except for `size`, all `inout_dcb` parameters are output parameters.

### Ft\_output

```
struct Ft_output {
    Uint32          size;
    struct Api_rc   result;
};
```

#### **Ft\_output Is Input To These Functions**

*Only the Ft\_output size parameter and only if you use Ft\_output in those calls listed in the output column*

#### **Ft\_output Is Output From These Functions**

`ft_abort()`  
`ft_aeactivation()`  
`ft_aedeactivation()`  
`ft_aereset()`

You must either pass the address of an `Ft_output` or you must pass the address of a NULL value. If you pass the address of an `Ft_output`, you must have enough memory to return `size` and `result`.

<code>size</code>	The size of the <code>Ft_output</code> structure and associated data in Octets. The <code>size</code> parameter is mandatory input if using <code>Ft_output</code> .
<code>result</code>	The result is of type <code>Api_rc</code> and provides error information.

## **Ft\_xxx\_out\_dcb**

```

struct Ft_xxx_out_dcb {
    Uint32
    struct Api_rc
    Function-specific
};

```

size;  
result;

info;

### **Ft\_xxx\_out\_dcb Is Input To These Functions**

*Only the Ft\_xxx\_out\_dcb size parameter and only if you use Ft\_xxx\_out\_dcb in those calls listed in the output column*

### **Ft\_xxx\_out\_dcb Is Output From These Functions**

ft\_bgroup()  
ft\_cancel()  
ft\_cattributes()  
ft\_close()  
ft\_connect()  
ft\_create()  
ft\_delete()  
ft\_deselect()  
ft\_edata()  
ft\_egroup()  
ft\_erase()  
ft\_etransfer()  
ft\_fattributes()  
ft\_fclose()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fattributes()  
ft\_fmove()  
ft\_fopen()  
ft\_ireceive()  
ft\_locate()  
ft\_nwcleared()  
ft\_open()  
ft\_rattributes()  
ft\_rcancel()  
ft\_rdata()  
ft\_read()  
ft\_rrequest()  
ft\_sdata()  
ft\_select()  
ft\_write()

**inout\_dcb**

You must either pass the address of an `inout_dcb` or you must pass the address of a `NULL` value. If you pass the address of an `inout_dcb`, you must have enough memory to return size and result. You may or may not supply additional values, depending on the function.

**size**      The size of the `Ft_xxx_out_dcb` structure and associated data in Octets. The size parameter is mandatory input if using `Ft_xxx_out_dcb`.

**result**    The result is of type `Api_rc` and provides error information.

## Memory Allocation for `inout_dcbs`

For asynchronous calls, the memory occupied by `Ft_output` and `Ft_xxx_out_dcb` is logically inconsistent until the request completes. Therefore, you should not reference the `inout_dcb` from the time the asynchronous call returns `SUCCESS` until `em_wait()` verifies completion of the request.

You can allocate memory for `Ft_output` and `Ft_xxx_out_dcb` in three ways.

- The recommended method is to have the output `inout_dcb` parameter reference the address of a `NULL` value when making the call. By doing so, the FTAM interface allocates the memory necessary; thus, you avoid having to anticipate how much memory is required.

After the FTAM function returns or for asynchronous calls, after `em_wait()` returns `SUCCESS`, you must call `ft_dfdcb()` to free the memory (even though the FTAM interface allocated the memory).

- Allocate the memory yourself (e.g., using `malloc()` or using variables). Ensure the size field is large enough to hold all possible return information; otherwise, you may receive an `FTE004_OUTPUT_BUFFER_OVERFLOW` error. For example, you want to have a size large enough to store the `inout_dcb` structure and all possible returning diagnostics (a linked list of 14).
- Before the request, invoke `ft_didcb()` to allocate memory for `Ft_xxx_out_dcb`. Free the memory used by the `input_dcb` parameter by invoking `ft_dfdcb()` after the function returns (synchronous calls) or after `em_wait()` returns (asynchronous calls). Ensure the

`additional_size` parameter is large enough to hold all possible return information (e.g., diagnostics and returned data); otherwise, you receive an `FTE004_OUTPUT_BUFFER_OVERFLOW` error.

---

**NOTE**

Your application must not perform any operations on or with an `inout DCB` parameter until `em_wait()` indicates the call is complete. Failure to follow this rule can cause serious internal errors to occur, resulting in unpredictable behavior or software failures.

---

**input\_dcb****input\_dcb**

```

struct Ft_xxx_in_dcb
{
    .
    .
    .
};

```

**Ft\_xxx\_in\_dcb Is Input To  
These Functions**

ft\_abort()  
ft\_aeactivation()  
ft\_bgroup()  
ft\_cancel()  
ft\_cattributes()  
ft\_close()  
ft\_connect()  
ft\_create()  
ft\_delete()  
ft\_deselect()  
ft\_edata()  
ft\_egroup()  
ft\_erase()  
ft\_etransfer()  
ft\_fattributes()  
ft\_fclose()  
ft\_fcopy()  
ft\_fdelete()  
ft\_fattributes()  
ft\_fmove()  
ft\_fopen()  
ft\_ireceive()  
ft\_locate()  
ft\_open()  
ft\_rattributes()  
ft\_rcancel()  
ft\_read()  
ft\_rrequest()  
ft\_select()  
ft\_write()

**Ft\_xxx\_in\_dcb Is Output  
From These Functions**

None



You pass large amounts of data to the FTAM interface. To reduce the number of exposed parameters, the `Ft_xxx_in_dcb` contains optional and function-specific parameters.

The `input_dcb` parameters may be mandatory or optional; they must occur after the exposed input parameters and before the `inout_dcb` parameters. If any of the `input_dcb` parameters are mandatory, you must pass the address of an `input_dcb`; otherwise, you may pass a NULL value.

You can allocate memory for `Ft_xxx_in_dcb` two ways. If you are using optional parameters, remember to allocate memory for them.

- Allocate the memory yourself (e.g., using `malloc()` and `free()`, or C variables).
- Before the request, a recommended practice is to invoke `ft_didcb()` to allocate memory for `Ft_xxx_in_dcb` and then set the appropriate `Ft_xxx_in_dcb` fields. After the request completes, invoke `ft_dfdbc()` to free the memory used by `Ft_xxx_in_dcb`.

---

## Local\_event\_name

```
typedef Sint32      Local_event_name;
```

### Local\_event\_name Is Input To These Functions

```
ft_abort()
ft_aeactivation()
ft_aedeactivation()
ft_aereset()
ft_bgroup()
ft_cancel()
ft_cattributes()
ft_close()
ft_connect()
ft_create()
ft_delete()
ft_deselect()
ft_edata()
ft_egroup()
ft_erase()
ft_etransfer()
ft_fcattributes()
ft_fclose()
ft_fcopy()
ft_fdelete()
ft_frattributes()
ft_fmove()
ft_fopen()
ft_ireceive()
ft_locate()
ft_nwcleared()
ft_open()
ft_rattributes()
ft_rcancel()
ft_rdata()
ft_read()
ft_rrequest()
ft_sdata()
ft_select()
ft_write()
```

### Local\_event\_name Is Output From These Functions

```
em_wait()
```

**Local\_event\_name** uniquely identifies a specific asynchronous function call and also determines whether a call is synchronous or asynchronous.

If the call is synchronous, the value must be zero (0).

The `em_wait()` function uses **Local\_event\_name** to identify a completed asynchronous operation. This identification is the same value that you passed to the operation when you initiated it. Though `em_wait()` can wait on multiple asynchronous operations, only one **Local\_event\_name** returns per call. You must know the **Local\_event\_name** of the event for which you are waiting.

Each outstanding asynchronous function must have a unique **Local\_event\_name** with a value greater than zero (0). A recommended practice is to start with the value one (1) and then increase by increments. You will find this method especially helpful when tracking incoming events.

---

## P\_address

```

struct P_address
{
struct Octet_string *p_selector; /* NULL if it is not present */
struct Octet_string *s_selector; /* NULL if it is not present */
struct Octet_string *t_selector; /* NULL if it is not present */
Sint16 n_nsaps; /* Number of NSAPs */
struct Octet_string *nsaps; /* ptr to a sequence of NSAPs */};
/* Maximum length of psap_selector is 16 Octets.
 * Maximum length of ssap_selector is 16 Octets.
 * Maximum length of tsap_selector is 32 Octets.
 * Maximum length of NSAP is 20 Octets.
 * Maximum number of NSAPS is 8.
 */

```

### **P\_address Is Input To These Functions**

ft\_connect()

### **P\_address Is Output From These Functions**

ft\_connect()

As an alternative to specifying the directory distinguished name, you can specify the P\_address when invoking ft\_connect() to communicate with other systems and processes.

The P\_address must be unique within the network and must be exactly the same as the one set during configuration. To have the system automatically set the P\_address structure, call convert\_paddr; the source is in /opt/ftam/demos/cnvrtd\_addr.c.

- Each application on a node must have a unique triple of p\_selector, s\_selector, and t\_selector. For example, one application could have (0000,0000,0000), another (0000,0000,0001), and another (2375,232323,75757575).
- Each node must have a unique nsap.
- The maximum length of the P\_address is 224 Octets. These Octets can be printable or unprintable characters.
- Each called\_dir\_name has a corresponding P\_address. Directory Services provides the mapping between the two.

p_selector	Presentation Service Access Point Selector: unique value that allows network layers to identify the connection to its peer.
s_selector	Session Service Access Point Selector: unique value that allows network layers to identify the connection to its peer.
t_selector	Transport Service Access Point Selector: unique value that allows network layers to identify the connection to its peer.
n_nsaps	Number of nsaps (Network Service Access Points)

The maximum number of nsaps is eight.

You might want to have multiple nsaps to uniquely label cards if using two network cards on one machine.

nsaps	Network Service Access Point: unique value that allows network layers to identify the connection to its peer.
-------	---

HP FTAM/9000 Data Structures  
**P\_address**



---

**NOTE**

---

References to Event Management functions mean the combination of `em_fdmemory()`, `em_wait()`, and `em_gperror()`.

Support functions are not bound to a specific regime. The only exception is `ft_nwcleared()`, which is used only when `ft_sdata()` returns the error `FTE008_NO_CON_RESOURCES`.



---

## Chapter Overview

This chapter describes the following functions in the order listed.

<b>Task to Perform</b>	<b>Function Used to Perform the Task</b>	<b>Description</b>
Managing Memory	ft_dfdbc()	De-allocate memory for DCBs (dynamic DCB de- allocation)
	ft_didcb()	Allocate memory for DCBs (dynamic DCB allocation)
	em_fdmemory()	Free dynamic memory allocated by em_gpperror() for the return_code and vendor_code strings
	ft_fdmemory()	Free dynamic memory allocated by ft_gpperror() for the return_code and vendor_code strings
Responding to Asynchronous Calls	em_wait()	Suspends application processing until a MAP event occurs
	em_hp_select()	Suspends application processing until a MAP or non- MAP event occurs
	em_hp_sigio()	Notifies that an asynchronous call completed by way of a SIGIO
Translating Error Messages	em_gpperror()	Translate returned Event Management return_codes and vendor_codes to printable strings
	ft_gpperror()	Translate returned FTAM return_codes and vendor_codes to printable strings
Determining Available Resources	ft_nwcleared()	Determine that a resource constraint cleared and that resource is now available

## Managing Memory

Use `ft_didcb()` and `ft_dfidcb()` to manage your DCB memory by allocating and de-allocating memory for DCB structures. Refer to the “Using FTAM” chapter for recommendations on allocating memory.

### `ft_didcb()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_didcb (dcb_type, additional_size, dcb_pointer, result)
enum Ft_dcb_type      dcb_type;
Uint32                additional_size;
Octet                 **dcb_pointer;
Api_rc                *result;
```

Use `ft_didcb()` to allocate memory for `Ft_xxx_in_dcb` or `Ft_xxx_out_dcb` structures. This function allocates the desired memory at the location pointed to by `dcb_pointer`.

- When allocating memory for an `inout_dcb`, you may optionally use the `additional_size` field to allocate more memory than the size of the DCB structure. All fields that are pointers may need additional memory. You must provide a large enough `additional_size` to contain all returning output.
- The outcome of `ft_didcb()` does not return in `result` if you pass an invalid result structure address on input.

## ft\_didcb() Parameters

<b>ft_didcb() Parameter</b>	<b>Type</b>	<b>Description</b>
dcb_type	Mandatory Input	Identifies the DCB to initialize
additional_size	Optional Input inout_dcb only	Specifies the amount of memory allocated beyond that allocated for the DCB
dcb_pointer	Input  Output	Address of the DCB to initialize  Address of the DCB pointer
result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)

## ft\_dfdbc()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_dfdbc (dcb_pointer, result)
Octet      *dcb_pointer;
Api_rc     *result;
```

Use `ft_dfdbc()` to de-allocate memory that was allocated by `ft_didcb()`. Also use `ft_dfdbc()` to de-allocate memory if you specified NULL for the `inout_dcb` when calling an FTAM function.

- If you allocated memory (e.g., using `malloc()`), you cannot call `ft_dfdbc()` to de-allocate memory.
- The outcome of `ft_dfdbc()` does not return in `result` if you pass an invalid result structure address on input.

## ft\_dfdcb() Parameters

<b>ft_dfdcb() Parameter</b>	<b>Type</b>	<b>Description</b>
dcb_pointer	Mandatory Input	Address of the DCB to free
result	Output	Pointer to the struct <code>Api_rc</code> containing the outcome of the operation: <code>result_code</code> (MAP 3.0 error) and <code>vendor_code</code> (HP-UX—specific error)

## em\_fdmemory()

```
#include </opt/ftam/include/map.h>
Return_code
em_fdmemory(memory_pointer, result)
Octet      *memory_pointer;
Api_rc     *result;
```

Use `em_fdmemory()` to free dynamic memory allocated by `em_gperror()` for the `return_code` and `vendor_code` strings. Note, if you supplied the memory for `em_gperror()`, do not use `em_fdmemory()` to free it.

- You may receive unpredictable results if using `free()` for memory allocated by `em_gperror()`.
- The outcome of `em_fdmemory()` does not return in `result` if you pass an invalid result structure address on input.
- If you allocated memory yourself (e.g., using `malloc()`), you must also free the memory.

## em\_fdmemory() Parameters

<b>em_fdmemory() Parameter</b>	<b>Type</b>	<b>Description</b>
memory_pointer	Mandatory Input	Points to the beginning of the string; address of the dynamic memory to free (allocated by em_gperror())
result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)

## ft\_fdmemory()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fdmemory (memory_pointer, result)
Octet      *memory_pointer;
Api_rc     *result;
```

Use ft\_fdmemory() to free dynamic memory allocated for ft\_gperror() return\_code and vendor\_code strings. Note, if you supplied the memory for ft\_gperror(), do not use ft\_fdmemory() to free it.

- You may receive unpredictable results if using free() for memory allocated by the FTAM library.
- The outcome of ft\_fdmemory() does not return in result if you pass an invalid result structure address on input.
- If you allocated memory yourself (e.g., using malloc()), you must also free the memory.

### **ft\_fdmemory() Parameters**

<b>ft_fdmemory() Parameter</b>	<b>Type</b>	<b>Description</b>
memory_pointer	Mandatory Input	Points to the beginning of the string; address of the dynamic memory to free (allocated by ft_gperror())
result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)

## Responding to Asynchronous Calls

The MAP 3.0 Event Management interface plays a central role in your use of asynchronous calls to FTAM functions.

It is common to have multiple asynchronous function calls outstanding. As responses to these calls come in, the FTAM Service Provider Process (SPP) queues them. The Event Management interface takes the first response off this queue and returns it to the caller as a MAP 3.0 event. At the same time, the Event Management interface updates the called function's inout DCB to show the outcome of the asynchronous call.

The FTAM/9000 implementation of MAP 3.0 Event Management consists of three functions, each used under slightly different circumstances. The functions are:

- `em_wait()`
- `em_hp_select()`
- `em_hp_sigio()`

Whenever possible, use only `em_wait()` to wait for and obtain MAP events. Since `em_wait()` is the only one of the three defined by the MAP 3.0 specification, using it eases porting to non-HP platforms. However, `em_wait()` is also very limited. Sometimes, you may have no alternative but to use one or both proprietary functions instead of or in addition to `em_wait()`.

Following are some guidelines for each of the three event management functions.

### When to Use `em_wait()`

The `em_wait()` function enables you to suspend application processing until a MAP event occurs. However, `em_wait()` does not provide for waiting on any non-MAP events.

Use `em_wait()` when your application needs to wait exclusively on MAP 3.0 events. For example, if your application is a daemon process which interacts only with the OSI network via FTAM protocols.

## When to Use `em_hp_select()`

The `em_hp_select()` function is a Hewlett-Packard proprietary extension to the MAP 3.0 specification. It can be regarded as a hybrid of the MAP 3.0 `em_wait()` function and the HP-UX `select()` system call. It is located in the common MAP library (`libmap.a`).

Use `em_hp_select()` instead of `em_wait()` when your application needs to wait on MAP events, plus non-MAP events that can be represented by a file descriptor and waited on via the HP-UX `select()` system call. This may be true if your application is interactive (for example, using asynchronous input from Terminal I/O or the X11 Window system). It also may be true if your application needs to communicate with other processes via named pipes or domain sockets as well as perform FTAM operations over the OSI network.

In addition, use `em_hp_select()` instead of `em_wait()` if your application would otherwise need `em_wait()` to return when interrupted by a signal. The `em_hp_select()` function does not return by itself when interrupted by a signal, but techniques exist to simulate this behavior. See the section on handling signal interrupts with `em_hp_select()`.

## When to Use `em_hp_sigio()`

The `em_hp_sigio()` function is another Hewlett-Packard proprietary extension in the common MAP library (`libmap.a`). It enables the Event Management interface to notify your application via the SIGIO signal when there is a MAP 3.0 event pending. However, `em_hp_sigio()` does not return any MAP events itself. Therefore, it should complement your use of `em_wait()` or `em_hp_select()`.

Use `em_hp_sigio()` along with `em_wait()` or `em_hp_select()` when your application needs to control the dispatch of MAP and non-MAP components by using signals and/or semaphores to indicate when a particular component has work to do. This is often necessary when porting applications from other operating systems such as AT& T's System V Unix or DEC's VMS.



## Using em\_wait()

```
#include %</opt/ftam/include/map.h>
Return_code
em_wait (timeout, return_event_name, result)
Em_t      timeout;
Local_event_name *return_event_name;
Api_rc    *result;
```

This section describes general use of asynchronous calls with `em_wait()`. For additional details on `em_wait()`, refer to the *HP FTAM/9000 Reference Manual*.

Call functions asynchronously when making multiple requests and when the order in which results are processed does not matter. If you make an asynchronous call, you can perform other operations while your request is being processed. Following are examples of when to call functions asynchronously or synchronously.

- If you cannot call function B before function A completes, you should call function A synchronously.
- If copying a file to four different nodes and the processing order does not matter, you would call `ft_copy()` asynchronously and then wait on the `return_event_names` to know when each `ft_copy()` completed successfully.

Using Support Functions  
Responding to Asynchronous Calls

The following text describes the sequence of making asynchronous requests and checking for their successful return from the responder.

1. Submit one or more asynchronous requests, each with a unique `return_event_name`.
2. Call `em_wait()` to determine if the requests returned successfully from the responder; you cannot specify the `return_event_name` for which you want to wait.

Though `em_wait()` function waits on multiple asynchronous calls, it returns `return_event_names` one at a time and in the order that each call completes.

3. Check the `inout_dcb` of the FTAM function (specified by the `return_event_name`) to determine the result of a completed request. Note, the `inout_dcb` pointer and structure are not valid until the `em_wait()` request returns successfully.

Since events can return in any order, the `return_event_name` returned may or may not be the first operation you requested or the one on which you are waiting.

A `return_event_name` returns only once. Therefore, you should process the `inout_dcb` for all `return_event_names` in the order in which they return or track all the `return_event_names` that return (for future use).

4. Continue calling `em_wait()` until you receive the `return_event_name` you want. If applicable, call `em_wait()` until all requests are processed.

---

**NOTE**

Do not perform any operations on or with an `inout_dcb` pointer or structure until `em_wait()` indicates the call using the `inout_dcb` is complete.

---

Following is an example of the use of `em_wait()`. This example creates and opens a file by asynchronously calling `ft_bgroup()`, `ft_create()`, and `ft_open()`, followed by synchronously calling `ft_egroup()`. These functions were called with `return_event_names` of 1, 2, 3, and 0, respectively. The `em_wait()` function waits on these calls.

```
struct Ft_bgroup_out_dcb      *bgr_inout_dcb;
struct Ft_create_out_dcb     *cre_inout_dcb;
struct Ft_open_out_dcb      *ope_inout_dcb;
Em_t                          time;
Local_event_name             return_event_name;
Api_rc                       result;
int                           i;
Return_code                  res;
struct Ft_diagnostic         *diag = NULL;
/* This loop waits on the three previously submitted
asynchronous calls. */
for (i = 1; i %<= 3; ++i) {
    time = -1;
    /* Call em_wait() to receive the latest event. */
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    /* Check the result of each function. */
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb->result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb->result, diag);
            break;
        case 2:
            if (cre_inout_dcb->result.return_code != SUCCESS)
                error_handler(cre_inout_dcb->result,
                             cre_inout_dcb->diagnostic);
            break;
        case 3:
            if (ope_inout_dcb->result.return_code != SUCCESS)
                error_handler(ope_inout_dcb->result,
                             ope_inout_dcb->diagnostic);
            break;
        default:
            break;
    }
}
```

## Using `em_hp_select()`

```
#include %</opt/ftam/include/map.h>
Return_code
em_hp_select (timeout, return_event_name, nfds, readfds,
              writefds, exceptfds, result)

Em_t
Local_event_name      *return_event_name
int                   *nfds
unsigned               *readfds
unsigned               *writefds
unsigned               *exceptfds
Api_rc                 *result;
```

This section describes how to use `em_hp_select()` and provides two program examples. For additional details, refer to the *HP FTAM/9000 Reference Manual*.

---

**NOTE**

FTAM events are asynchronous FTAM operations that complete. Non-FTAM events consist of any type of descriptor supported by the `select(2)` system call that becomes readable or writable, or has exceptions pending.

Multiple asynchronous FTAM functions calls are commonly outstanding at the same time. An application may also need to interact with other processes using an interprocess communication mechanism (for instance, HP-UX Domain Sockets or pipes). It may also require response to asynchronous events from a terminal keyboard or window system (for instance, X- Windows).

The MAP 3.0 Event Management interface does not provide an independent method to wait on both FTAM and non-FTAM events simultaneously. Therefore, HP added a proprietary function—`em_hp_select()`—to provide this capability.

Neither `em_wait()` nor `em_hp_select()` automatically return when interrupted by a signal. However, the following section describes a way to make a signal generate a non-FTAM event, causing `em_hp_select()` to return on receipt of a signal.

## Waiting On FTAM and Non-FTAM Events

Use the `em_hp_select()` function when an application must simultaneously wait for both an FTAM and a non-FTAM event to occur.

Without `em_hp_select()`, the application would need to implement a “busy-wait” loop to poll for the FTAM and non-FTAM events. However, this technique is very expensive in terms of CPU utilization, and degrades system performance. This is unacceptable for most real-time or highly interactive applications. Therefore, HP provides `em_hp_select()` as an alternative method for waiting on both FTAM and non-FTAM events.

The `em_hp_select()` function suspends processing of the application until an event occurs. While the application is suspended (i.e, blocked), it does not contribute to the demand for CPU time. Therefore, the busy-wait logic can be replaced with the following:

```
em_hp_select(...with indefinite timeout to block until event  
occurs...)  
handle_event())
```

If you need to wait only on FTAM events, use the `em_wait()` function. If you need to wait only on non-FTAM events, use the HP-UX `select()` system call. However, if you need to wait on both types of events simultaneously, use the `em_hp_select()` function. Any combination of `em_wait()`, `select()`, and `em_hp_select()` within the same program is valid; all three functions are compatible with one another.

The following program example illustrates how to use `em_hp_select()` to wait on FTAM and non-FTAM events simultaneously. The program reads commands from `stdin` and starts the requested FTAM operation. The operation is performed asynchronously so that new commands can be read in before the previous operation completes. The operation results are processed as they complete.

This program calls two functions to process the FTAM and non-FTAM events. The logic for these functions is not shown in the example; however, a brief description of each function follows.

**read\_and\_process\_command()**. This function processes the non-FTAM events by reading and parsing a command line string from `stdin`. If the command is a “Quit” command, the function returns `TRUE` to indicate termination. Otherwise, the appropriate FTAM operation is initiated asynchronously. In this case, the function returns `FALSE` to indicate more commands can be read from `stdin`.

For example, this function could be a batch entry console for file management. Several file transfers could be started without having to wait for each transfer to complete before starting the next.

**map\_operation\_completed (map\_event, map\_rc).** This function processes the FTAM events by performing some action on the results of the FTAM operation that completed. The operation is identified by the map\_event passed as input. If map\_rc equals SUCCESS, the inout\_dcb for the operation contains the outcome of the operation. Otherwise, the map\_rc equals EME032\_IPC\_ERROR; the operation fails and the inout\_dcb does not contain valid information. The specific actions involved with processing a completed FTAM event are specific to the particular event, and could be as simple as logging the results to an output file.

```
#include <stdio.h>
#include </opt/ftam/include/map.h>
#define BITS_PER_INT 32
#define SET_MASK(f, fds) (fds)[((f)/BITS_PER_INT)] |=
(1<<((f)%BITS_PER_INT))
#define TST_MASK(f, fds) ((fds)[((f)/BITS_PER_INT)] &
(1<<((f)%BITS_PER_INT)))
main()
{
    Bool done;
    Local_event_name map_event;
    Return_code map_rc;
    Api_rc map_result;
    char *ret_str;
    char *ven_str;
    int nfd;
    unsigned readfds[2]; /* handles upto 64 fds */
    /* Read and process the first command. No FTAM operations are
    ** outstanding at this time, so do not wait for any to complete.
    */
    done = read_and_process_command();
    /* Wait indefinitely for MAP and non-MAP (i.e., stdin readable) events
    ** to occur. If an EME032_IPC_ERROR occurs, the MAP operation
    ** identified by map_event is treated as having completed with the IPC
    ** error. Non-MAP events can also return in this case, so they
    ** are processed as well.
    **
    ** If any other error occurs, there is a programming error when calling
    ** em_hp_select(); in this case, use em_gperror() and em_fdmemory() to
    ** translate the error into printable character strings, then
    ** abort processing so you can fix the coding error.
    **
    ** Continue processing both FTAM and non-FTAM events until a "Quit"
    command
    ** is entered as the non-FTAM event.
    */
}
```

## Using Support Functions

### Responding to Asynchronous Calls

---

```
while ( ! done )
{
    readfds[0] = 0;
    readfds[1] = 0;
    SET_MASK(stdin, readfds);
    nfds = stdin + 1;
    map_rc = em_hp_select(FOREVER, & map_event, & nfds, readfds, NULL,
NULL,
                        & map_result);
    if ((map_rc == SUCCESS) || (map_rc == EME032_IPC_ERROR))
    {
        if ((map_event != 0))
            map_operation_completed(map_event, map_rc);
        if (TST_MASK(stdin, readfds))
            done = read_and_process_command();
    }
    else
    {
        ret_str = NULL;
        ven_str = NULL;
        map_rc = em_gperror(& map_result, & ret_str, & ven_str, &
map_result);
        if (map_rc == SUCCESS)
            fprintf(stderr, "%s%s", ret_str, ven_str);
        em_fdmemory(ret_str, & map_result);
        em_fdmemory(ven_str, & map_result);
        exit(1);
    }
} /* while */
/* You finished entering commands, but all of the MAP operations may not
** be complete. Since only MAP events are left, em_wait() is used instead
** of em_hp_select(). An EME002_EXP_EMPTY result indicates all MAP
** operations have completed.
*/
done = FALSE;
do {
    map_rc = em_wait(FOREVER, & map_event, & map_result);
    if ((map_rc == SUCCESS) || (map_rc == EME032_IPC_ERROR))
        map_operation_completed(map_event, map_rc);
    else if (map_rc == EME002_EXP_EMPTY)
        done = TRUE;
    else
    {
        done = TRUE;
        ret_str = NULL;
        ven_str = NULL;
        map_rc = em_gperror(& map_result, & ret_str, & ven_str, & map_result);
        if (map_rc == SUCCESS)
            fprintf(stderr, "%s%s", ret_str, ven_str);
        em_fdmemory(ret_str, & map_result);
        em_fdmemory(ven_str, & map_result);
        exit(2);
    }
} while ( ! done );
exit(0);
} /* main */
```



## Handling Signal Interrupts

Suppose an application needs the `em_wait()` function to return prematurely if a signal interrupt occurs. One case where this is applicable is when receipt of a signal should cause a new FTAM operation to be initiated or an outstanding operation to be aborted (e.g., the application is being terminated abruptly). Since FTAM functions are not reentrant, the application needs to force `em_wait()` to return prematurely so the appropriate action can be taken upon receipt of the signal.

The `em_wait()` and `em_hp_select()` functions cannot detect signal interrupts and automatically return. However, you can cause `em_hp_select()` to return when interrupted by a signal. This involves translating the signal interrupt into an `em_hp_select()` non-FTAM event while executing the signal handler. When `em_hp_select()` resumes processing after the signal handler completes, the non-FTAM event will be detected and `em_hp_select()` will return.

Several techniques exist to translate the signal interrupt into an `em_hp_select()` non-FTAM event. One way is to use an HP-UX pipe as follows:

1. Open the pipe for both reading and writing within the application program.
2. For the expected signals, install a signal handler that writes the trapped signal value to the pipe. This will cause the pipe to become readable.
3. Replace calls to `em_wait()` (which waits for only FTAM events) with calls to `em_hp_select()`. Initialize the `readfds` mask to include the read descriptor for the pipe. If a signal interrupt occurs while executing or blocking within `em_hp_select()`, the pipe will become readable and cause `em_hp_select()` to return.
4. After `em_hp_select()` returns, read the signal value from the pipe so that the notification will be cleared; then take the appropriate action based on the signal value.

## Using Support Functions

### Responding to Asynchronous Calls

The following program example illustrates how to use `em_hp_select()` and an HP-UX pipe to implement the signal handling technique described previously.

```
#include %<signal.h>
#include %</opt/ftam/include/map.h>
#define BITS_PER_INT 32
#define SET_MASK(f, fds) ((fds)[((f)/BITS_PER_INT)] |=
(1%<%<((f)%BITS_PER_INT))
#define TST_MASK(f, fds) ((fds)[((f)/BITS_PER_INT)] &
(1%<%<((f)%BITS_PER_INT)))
#define READ 0
#define WRITE 1
int sig_pipe[2]; /* Pipe r/w descriptors used to queue signals */
void
signal_handler(sigval)
int sigval;
{
    /* Put signal on the pipe to make the pipe readable. This will cause
    ** em_hp_select() to return.
    */
    write(sig_pipe[WRITE], & sigval, sizeof(int));
    /* Perform other signal handling logic if any. */
}
main()
{
    int sigval; /* The signal value read from pipe */
    struct sigvec vec; /* sigvector for installing handler */
    Local_event_name map_event; /* MAP Event Name */
    Return_code map_rc; /* MAP return code */
    Api_rc map_result; /* MAP result structure */
    int nfds; /* File descriptor bound */
    unsigned readfds[2]; /* File descriptor mask */
    /*
    ** Create the HP-UX pipe which is used to translate the signal interrupt
    ** into an em_hp_select() non-MAP event.
    */
    pipe (sig_pipe);
```

---

```
/* Install the signal handler for the signal you want to trap. */
vec.sv_handler = signal_handler;
vec.sv_mask    = 0;
vec.sv_flags   = 0;
sigvector(signal_you_want_to_trap, &vec, NULL);
/* Initiate the asynchronous FTAM operations. The code for
** initiating these operations is not shown in this example.
*/
/* Wait for a MAP operation to complete or a signal to be caught.
** This is done by blocking on both MAP events and the sig_pipe
** to become readable.
*/
readfds[0] = 0;
readfds[1] = 0;
SET_MASK(sig_pipe[READ], readfds);
nfds = sig_pipe[READ] + 1;
map_rc = em_hp_select(FOREVER, &map_event, &nfds, readfds, NULL, NULL,
                    &map_result);
if ((map_rc == SUCCESS) || (map_rc == EME032_IPC_ERROR))
{
    if (map_event != 0)
    {
        /* Process the results of the completed MAP operation. The
        ** code for this function is not shown in this example.
        */
        map_operation_completed(map_event, map_rc);
    }
    if (TST_MASK(sig_pipe[READ], readfds))
    {
        /* em_hp_select() was interrupted by a signal. Remove the signal
        ** value from the pipe to clear the interrupt notification.
        */
        read(sig_pipe[READ], &sigval, sizeof(int));
        /* Perform the application specific logic to be done when
        ** em_hp_select() unblocks because of a signal.
        */
    }
}
else
{
    /* Handle the em_hp_select() error. See Example 1. */
}
```

## Using `em_hp_sigio()`

```
#include %</opt/ftam/include/map.h>
Return_code
em_hp_sigio (action, pid, result)
Unit32      action;
Sint32      pid;
Api_rc      *result;
```

This section describes how to use `em_hp_select()` and provides a program example. For additional details, refer to the *HP FTAM/9000 Reference Manual*.

Use `em_hp_sigio()` to enable the MAP Event Management interface to generate SIGIO signals whenever an asynchronous MAP event is pending. Unlike the other MAP event management functions, `em_hp_sigio()` does not return a MAP event to the caller. Instead, it merely puts the MAP event management interface into an extended capability mode.

By using this capability, you can suspend the processing of your application until any of a variety of signals or semaphore operations cause your application to resume execution. Then, only after the SIGIO signal has been received do you need to call `em_wait()` to obtain the MAP event that is pending. Furthermore, since you know the MAP event is already pending, you can call `em_wait()` with a zero timeout to obtain the event and return immediately.

The following program example illustrates one possible use of `em_hp_sigio()`. The program initiates and processes two types of events: MAP events in the form of FTAM and/or FTAM asynchronous operations, and Non-MAP events that take the form of SIGUSR1 signals. The `em_hp_sigio()` function is used to enable SIGIO signal notification in the MAP Event Management interface, and `em_wait()` is used to process the MAP event after SIGIO is caught.

```
/*
**
**          Sample program for em_hp_sigio()
**
** This program is designed with three primary components:
**
**      * a central control/dispatch component
**      * a MAP event processing component
**      * a Non-MAP event processing component
**
** The central control/dispatch component makes use of an array of semaphores
** to manage program suspension and resumption, and to keep track of which
** components (MAP or Non-MAP) have work to do. Additionally, a signal
** handler is installed for the MAP and Non-MAP generated signals which
** manipulates the semaphore array. The SIGIO signal is used to notify the
** control/dispatch component that a MAP event needs to be processed, and the
** SIGUSR1 signal is used for Non-MAP events.
**
** The MAP and Non-MAP processing components are referenced as external
** functions. Details of these components are specific to each application
** and are, therefore, not included here. Note, calling em_wait() to obtain
** the MAP event would normally be included in the MAP specific processing
** component, but em_wait() is pulled out into the main component here to
** demonstrate the relationship between em_hp_sigio() and em_wait().
*/
#include %<stdio.h>
#include %<errno.h>
#include %<signal.h>
#include %<sys/types.h>
#include %<sys/ipc.h>
#include %<sys/sem.h>
#include %</opt/ftam/include/map.h>
/*
** Functions and variables not provided by the demonstration program.
*/
extern Bool operations_to_wait_on_and_process;
extern void initiate_MAP_operation();
extern void process_MAP_operation();
extern void initiate_NON_MAP_operation();
extern void process_NON_MAP_operation();
/*
** Semaphores used to manage suspend/resume activity
**
** Three semaphores are used to manage the MAP and Non-MAP asynchronous
** activity. One semaphore is used to control the suspension and resumption
** of process. Another semaphore is used to keep track of the number of
** pending events for each event type, MAP and Non-MAP in this case.
**
** The semaphores are operated on by a set of functions included following
** the main program. The functions provide logical P and V operations on a
** set of semaphores, based on HP-UX semaphore operations (system calls).
*/

#define NUM_SEMAPHORES 3
#define SEM_CONTROL 1 /* Used for suspend & resume control */
#define SEM_MAP 2 /* Used to track pending MAP events */
#define SEM_NON_MAP 4 /* Used to track pending Non-MAP events */
```

## Using Support Functions

### Responding to Asynchronous Calls

---

```
#define WAIT          0          /* Tells sem_p() to block if necessary */
#define NO_WAIT      1          /* Tells sem_p() to return immediately */
int  sem_id;                /* The semaphore array for async mgmt */
int  sem_create();         /* Create semaphores on the system */
void sem_remove();        /* Remove semaphores from the system */
int  sem_p();              /* Perform a P (get) operation */
void sem_v();             /* Perform a V (give) operation */
/*
** Signals and signal handler used to manage asynchronous event notification.
** The signal_handler logic is included following the main program.
*/
#define SIG_MAP      SIGIO      /* MAP notification signal */
#define SIG_NON_MAP  SIGUSR1    /* Non-MAP signal (for example) */
void signal_handler();        /* Signal handler for async event mgmt */
/*
**
**                               MAIN PROGRAM
*/
main()
{
    Local_event_name    map_event_name; /* MAP 3.0 event name (identifier) */
    Return_code         map_rc;         /* MAP 3.0 function return code */
    Api_rc              map_result;     /* MAP 3.0 result structure */
    struct sigvec       vec;            /* Signal Vector for event handler */
    printf("Demonstration program for em_hp_sigio()\n");
    /*
    ** Create and initialize the semaphore array for process suspension and
    ** resumption.
    */
    printf("Creating semaphore for process suspend & resume.\n");
    sem_id = sem_create(NUM_SEMAPHORES);
    /*
    ** Install the signal handler to catch interrupts for each signal used
    ** as an event notification.
    */
    printf("Installing signal handlers for MAP and Non-MAP events\n");
    vec.sv_handler = signal_handler;
    vec.sv_mask    = 0;
    vec.sv_flags   = 0;
    sigvector(SIGIO, & vec, NULL);    /* MAP events */
    sigvector(SIGUSR1, & vec, NULL); /* Non-MAP events */
    /*
    ** Put the MAP 3.0 Interface into SIGIO notification mode. This causes
    ** MAP Event Management to generate a SIGIO signal whenever there is
    ** an asynchronous event to process.
    **
    ** Note, the process to signal is the local (this) process. The process
    ** id is negated to distinguish it from a process group id.
    */
    printf("Calling em_hp_sigio() to enable SIGIO notification for MAP.\n");
    map_rc = em_hp_sigio(EM_SIGIO_ENABLE, getpid(), & map_result);
    if (map_rc != SUCCESS)
    {
        fprintf(stderr, "ERROR: em_hp_sigio(ENABLE) failed, %d\n", map_rc);
        exit(1);
    }
}
```

---

```
/*
** Activate MAP 3.0 Application Entity for FTAM or FTAM, then initiate
** asynchronous MAP 3.0 operations.
*/
initiate_MAP_operation();
/*
** Initialize and initiate Non-MAP asynchronous operations.
*/
initiate_NON_MAP_operation();
while (operations_to_wait_on_and_process)
{
    /*
    ** Suspend processing until an asynchronous event occurs.
    */
    printf("\nSuspending processing until an event occurs.\n");
    sem_p(sem_id, SEM_CONTROL, WAIT);
    /*
    ** Processing has resumed. Determine the event type that occurred.
    */
    printf("Resuming processing after semaphore popped.\n");
    /*
    ** Process MAP 3.0 (FTAM or FTAM) event if one is pending.
    **
    ** A non-blocking sem_p() operation is used to atomically test and
    ** decrement the pending MAP event count. If a MAP event is pending,
    ** sem_p() returns zero; otherwise, it returns non-zero.
    **
    ** Note, calling em_hp_sigio() earlier merely put the MAP interface
    ** into SIGIO notification mode. You must still call em_wait() to
    ** obtain the event name for the MAP 3.0 operation that triggered
    ** the signal notification. Furthermore, the operation may only be
    ** partially complete. Calling em_wait() processes the MAP event as
    ** much as possible, but returns EME005_TIMEOUT if the event is not
    ** completely processed.
    */
    if (sem_p(sem_id, SEM_MAP, NO_WAIT) == 0)
    {
        printf("Calling em_wait() to obtain MAP event.\n");
        map_rc = em_wait(0, & map_event_name, & map_result);
        if (map_rc == SUCCESS)
            process_MAP_operation(map_event_name);
        else if (map_rc == EME005_TIMEOUT)
            printf("em_wait timeout: operation partially complete\n");
        else
            fprintf(stderr, "ERROR: em_wait() failed, %d\n", map_rc);
    }
}
```

## Using Support Functions

### Responding to Asynchronous Calls

---

```
/*
** Process Non-MAP event if one is pending.
**
** A non-blocking sem_p() operation is used to atomically test and
** decrement the pending Non-MAP event count. If a Non-MAP event is
** pending, sem_p() returns zero; otherwise, it returns non-zero.
**
** Note, event processing is specific to your own application;
** details are not included in this example.
*/
if (sem_p(sem_id, SEM_NON_MAP, NO_WAIT) == 0)
{
    process_NON_MAP_operation();
}
} /* while - operations to process */
/*
** Disable SIGIO notification for the MAP 3.0 Interface.
** Note, the process id parameter is ignored so any value is fine.
*/
printf("\nCalling em_hp_sigio() to disable SIGIO notification.\n");
map_rc = em_hp_sigio(EM_SIGIO_DISABLE, 0, & map_result);
if (map_rc != SUCCESS)
    fprintf(stderr, "ERROR: em_hp_sigio(DISABLE) failed, %d\n", map_rc);

/*
** Remove semaphore resources.
*/
printf("Cleaning up semaphore resources.\n");
sem_remove(sem_id);
exit(0);
} /* end main() */
/*
**
**                               Signal Handler for Event Notification
**
** The event notification signal handler performs a semaphore V operation
** on the suspend/resume semaphore, plus the specific event type semaphore.
*/
void
signal_handler(sig)
int sig;
{
    unsigned long sem_mask;
    printf("Signal handler taken for signal %d\n", sig);
    /*
    ** Always included the Control semaphore in the semaphore mask. This is
    ** what causes the main program to resume execution.
    */
    sem_mask = SEM_CONTROL;
```



```
/*
** Determine the event type based on the signal received.
** Update the semaphore mask to include the event type specific
** semaphore.
*/
if (sig == SIG_MAP)
    sem_mask |= SEM_MAP;
else if (sig == SIG_NON_MAP)
    sem_mask |= SEM_NON_MAP;
else {
    fprintf(stderr, "ERROR: unsupported signal, %d\n", sig);
    exit(1);
}
/*
** Perform a V (give) operation on the specified semaphores so that the
** process can resume (if suspended), and the event count for the given
** type is incremented atomically.
*/
sem_v(sem_id, sem_mask);
} /* signal_handler */
/*
**
**
**
** This function creates a semaphore array for the specified number of
** semaphores (up to 32) and initializes them to an "unblocked" state.
*/
int
sem_create(sem_cnt)
int sem_cnt;
{
    int          sem_id;
    ushort      sem_val[sizeof(unsigned long)];
    sem_id = semget(IPC_PRIVATE, sem_cnt, IPC_CREAT | 0600);
    if (sem_id == -1)
    {
        fprintf(stderr, "ERROR: creating semaphore, errno %d\n", errno);
        exit(1);
    }
    memset(sem_val, 0, sizeof(sem_val));
    if (semctl(sem_id, 0, SETALL, sem_val) == -1)
    {
        fprintf(stderr, "ERROR: initializing semaphore, errno %d\n", errno);
        exit(1);
    }
    return(sem_id);
}
```

## Using Support Functions

### Responding to Asynchronous Calls

---

```
/*
**
**          sem_remove()
**
** This function removes the semaphore array from the system.
*/
void
sem_remove(sem_id)
int sem_id;
{
    if (semctl(sem_id, 0, IPC_RMID, 0) == -1)
    {
        fprintf(stderr, "ERROR: removing semaphore, errno %d\n", errno);
        exit(1);
    }
}
/*
**
**          sem_p()
**
** This function performs a semaphore P (get) operation on the set of
** semaphores specified by 'sem_mask'. The sem_mask is defined such that
** semaphore number N is represented by bit (1 << (N-1)) in the mask.
**
** If the 'sem_nowait' flag is FALSE, processing is suspended until blocking
** conditions for all of the specified semaphores have cleared.
** If the 'sem_nowait' flag is TRUE, processing continues immediately and the
** return value indicates whether a blocking condition exists or not.
**
** If all blocking conditions are cleared, the function returns zero.
** Otherwise, non-zero returns.
*/
int
sem_p(sem_id, sem_mask, sem_nowait)
int sem_id;
unsigned long sem_mask;
int sem_nowait;
{
    struct sembuf sem_ops[sizeof(unsigned long)];
    int sem_num = 0;
    int sem_cnt = 0;
    int rc = 0;
    while ((sem_num << sizeof(unsigned long)) && sem_mask)
    {
        if (sem_mask & 1)
        {
            sem_ops[sem_cnt].sem_num = sem_num;
            sem_ops[sem_cnt].sem_op = -1;
            sem_ops[sem_cnt].sem_flg = (sem_nowait ? IPC_NOWAIT : 0);
            sem_cnt++;
        }
        sem_num++;
        sem_mask >= 1;
    }
}
```

---

```
    if (sem_cnt > 0)
    {
        do {
            rc = semop(sem_id, sem_ops, sem_cnt);
        } while ((rc == -1) && (errno == EINTR));
        if ((rc == -1) && !((sem_nowait) && (errno == EAGAIN)))
        {
            fprintf(stderr, "ERROR: semaphore P (get), errno %d\n", errno);
            exit(1);
        }
    }
    return(rc);
} /* sem_p */
/*
**
**                               sem_v()
**
** This function performs a semaphore V (give) operation on the set of
** semaphores specified by 'sem_mask'. The sem_mask is defined such that
** semaphore number N is represented by bit (1 < (N-1)) in the mask.
*/
void
sem_v(sem_id, sem_mask)
int      sem_id;
unsigned long sem_mask;
{
    struct sembuf      sem_ops[sizeof(unsigned long)];
    int                sem_num = 0;
    int                sem_cnt = 0;
    int                rc;
    while ((sem_num < sizeof(unsigned long)) && sem_mask)
    {
        if (sem_mask & 1)
        {
            sem_ops[sem_cnt].sem_num = sem_num;
            sem_ops[sem_cnt].sem_op  = 1;
            sem_ops[sem_cnt].sem_flg = 0;
            sem_cnt++;
        }
        sem_num++;
        sem_mask >= 1;
    }
    if (sem_cnt > 0)
    {
        do {
            rc = semop(sem_id, sem_ops, sem_cnt);
        } while ((rc == -1) && (errno == EINTR));

        if (rc == -1)
        {
            fprintf(stderr, "ERROR: semaphore V (give), errno %d\n", errno);
            exit(1);
        }
    }
} /* sem_v */
```

## Translating Error Messages

FTAM and Event Management functions return `Return_codes`.

- To translate errors returned by Event Management to character strings, invoke `em_gperror()`.
- To translate errors return by FTAM functions to character strings, invoke `ft_gperror()`.

### `em_gperror()`

```
#include %</opt/ftam/include/map.h>
Return_code
em_gperror(input_result, return_string, vendor_string, result)
Api_rc      *input_result;
Octet      **return_string;
Octet      **vendor_string;
Api_rc      *result;
```

The `em_gperror()` function translates the `input_result` structure from Event Management functions to character strings, as follows:

- The `input_result->return_code` is translated to a character string called `return_string`.
- The `input_result->vendor_code` is translated to a character string called `vendor_string`.
- If the address that the `return_string` points to is `NULL`, the interface automatically allocates the required memory. After `em_gperror()` returns, you must use `em_fdmemory()` to free the memory. Do not use `free()` to deallocate memory allocated by FTAM.
- If the address that the `return_string` points to is not `NULL`, the value must be the address of the memory that was previously allocated. The specified data area must be large enough to hold a 256-byte string (including the `NULL`-terminator).
- You may receive unpredictable results if using `free()` for memory allocated by `em_gperror()`.

## **em\_gperror() Parameters**

<b>em_gperror() Parameter</b>	<b>Type</b>	<b>Description</b>
input_result	Mandatory Input	Pointer to the Api_rc structure containing information to translate: return_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
return_string	Mandatory Input	Address of where to place the character string for input_result->return_code
	Output	Address of the return_code character string; this string is NULL terminated and already contains the NEW LINE character
vendor_string	Optional Input	Address of where to place the character string for input_result->vendor_string
	Output	Address of the vendor_code character string; this string is NULL terminated and already contains the NEW LINE character
result	Output	Pointer to the Api_rc structure containing outcome of the operation: return_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)

Using Support Functions  
Translating Error Messages

## ft\_gperror()

```
#include %</opt/ftam/include/map.h>
Return_code
ft_gperror (input_result, return_string, vendor_string, result)
Api_rc      *input_result;
Octet       **return_string;
Octet       **vendor_string;
Api_rc      *result;
```

The `ft_gperror()` function translates the `input_result` structure from FTAM functions to character strings, as follows:

- The `input_result->return_code` is translated to a character string called `return_string`.
- The `input_result->vendor_code` is translated to a character string called `vendor_string`.
- If the address that the `return_string` points to is `NULL`, the interface automatically allocates the required memory. After `ft_gperror()` returns, you must use `ft_fdmemory()` to free the memory.

---

**NOTE**

---

Do not use `free()` to deallocate memory allocated by FTAM.

- If the address that the `return_string` points to is not `NULL`, you must allocate the memory. The specified data area must be large enough to hold a 256-byte string (including the `NULL`-terminator).

### **ft\_gperror() Parameters**

<b>ft_gperror() Parameter</b>	<b>Type</b>	<b>Description</b>
input_result	Mandatory Input	Pointer to the Api_rc structure containing information to translate: return_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
return_string	Mandatory Input	Address of where to place the character string for input_result->return_code
	Output	Address of the return_code character string; this string is NULL terminated and already contains the NEW LINE character
vendor_string	Optional Input	Address of where to place the character string for input_result->vendor_string
	Output	Address of the vendor_code character string; this string is NULL terminated and already contains the NEW LINE character
result	Output	Pointer to the Api_rc structure containing outcome of the operation: return_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)

## Determining Available Resources

While sending data, limited flow control resources may prevent the operation from completing. This circumstance is indicated by the error FTE008\_NO\_CON\_RESOURCES returning on `ft_sdata()` function calls. To determine when the resource clears, call `ft_nwcleared()`.

### `ft_nwcleared()`

```
#include </opt/ftam/include/map.h>
#include </opt/ftam/include/mapftam.h>
Return_code
ft_nwcleared (connection_id, local_event_name, inout_dcb)
Connection_id          connection_id;
Local_event_name       local_event_name;
struct Ft_nwcleared_out_dcb  **inout_dcb;
```

Use `ft_nwcleared()` to determine when the FTE008\_NO\_CON\_RESOURCES error condition (on `ft_sdata()`) clears and there are enough resources to continue.

- The `ft_nwcleared()` `connection_id` must be the same as the connection that returned the FTE008\_NO\_CON\_RESOURCES error.
- If you invoke `ft_nwcleared()` synchronously, the function does not return until a resource is acquired. Though resources clear, there is no guarantee they will be available for a subsequent `ft_sdata()` request.
- If you invoke `ft_nwcleared()` asynchronously, you cannot call other functions on that `connection_id` until the `ft_nwcleared()` is noted by `em_wait()`. If you call another function before `ft_nwcleared()` is noted, you are not informed when the resource is available (i.e., the `ft_nwcleared()` is revoked).
- When calling `ft_sdata()` a second time, all parameters must be the same as those of the initial `ft_sdata()` request.



**EXAMPLE:** This example shows the use of `ft_nwcleared()`. The `connection_id` and `data_unit` parameters were previously set.

```

Connection_id      connection_id;
struct Ft_data_unit data_unit;
Local_event_name   sda_event_name, nwc_event_name;
struct Ft_sdata_out_dcb *sda_inout_dcb;
struct Ft_nwcleared_out_dcb *nwc_inout_dcb;
Result_code        res;

sda_event_name = 1;
sda_inout_dcb = NULL;
res = ft_sdata(connection_id, & data_unit, sda_event_name, & sda_inout_dcb);
if (res == FTE008_NO_CON_RESOURCES) {
    nwc_event_name = 0;
    nwc_inout_dcb = NULL;
    res = ft_nwcleared(connection_id, nwc_event_name, & nwc_inout_dcb);
    if (res == SUCCESS) {
        res = ft_sdata(connection_id, & data_unit, sda_event_name,
                       & sda_inout_dcb);
    }
}

```

### **ft\_nwcleared() Parameters**

<b>ft_nwcleared() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>connection_id</code>	Mandatory Input	Uniquely identifies a specific connection to the filestore
<code>local_event_name</code>	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
<code>inout_dcb.size</code>	Mandatory input if using <code>inout_dcb</code>	Size (in Octets) of the <code>inout_dcb</code> structure and data
<code>inout_dcb.result</code>	Output	Pointer to the struct <code>Api_rc</code> containing the outcome of the operation: <code>result_code</code> (MAP 3.0 error) and <code>vendor_code</code> (HP-UX—specific error)

Using Support Functions  
**Determining Available Resources**



## Using High Level, Context Free Functions

High level functions simplify FTAM programming by performing a series of low level functions for you. Since the high level functions described in this chapter are context free, you can use them at any time. Applications written with high level, context free (HLCF) functions are smaller, easier to write, and less prone to error.

Refer to the “Example Programs” chapter for model programs using the HLCF functions.

---

## Chapter Overview

This chapter describes the following HLCF functions in the order listed.

<b>Task to Perform</b>	<b>Function Used to Perform the Task</b>	<b>Description</b>
Copying and Moving FTAM Files	ft_fcopy() ft_fcopy_aet()	Copy files
	ft_fmove_aet() ft_fmove()	Move files
Reading and Changing Attributes	ft_frattributes() ft_frattributes_aet()	Read attributes
	ft_fcattributes() ft_fcattributes_aet()	Change attributes
Deleting Files	ft_fdelete() ft_fdelete_aet()	Delete files

---

**NOTE**

This chapter does not explain the parameter structures. For detailed structure information (parameter settings), refer to the “FTAM Data Structures” chapter.

---

## Copying and Moving FTAM Files (HLCF)

Use `ft_fcopy()` and `ft_fcopy_aet()` to copy files, and `ft_fmove()` and `ft_fmove_aet()` to move files with only one call. Since copying or moving files using low level calls requires most FTAM functions, you can save significant coding time and effort using `ft_fcopy()`, `ft_fcopy_aet()`, `ft_fmove()`, and `ft_fmove_aet()`.

### `ft_fcopy()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fcopy(source_dirname, source_filename, destination_dirname,
         destination_filename, ae_label, return_event_name,
         input_dcb, inout_dcb)

Ae_dir_name      source_dirname;
Ft_filename      source_filename;
Ae_dir_name      destination_dirname;
Ft_filename      destination_filename;
Ae_label         *ae_label;
Local_event_name return_event_name;
struct Ft_fcopy_in_dcb *input_dcb;
struct Ft_fcopy_out_dcb **inout_dcb;
```

The `ft_fcopy()` function copies one FTAM file (contents and attributes) to another FTAM file. The original file remains in tact. The new file contains the source file attributes and those attributes that are automatically set (e.g., filesize).

- If the file you are copying to already exists, the outcome depends on how you set `input_dcb->overwrite`. If `input_dcb->overwrite` is set to `FT_RECREATE_FILE`, FTAM copies the source file over the existing destination file. If the value is `FT_DONT_RECREATE_FILE`, FTAM does not copy the source file over the existing file, the function fails, and you receive an error message.

---

#### NOTE

The following points do not apply to HP-UX FTAM responders. HP-UX FTAM responders do not use these fields. The information is provided here because other implementations may use these fields.

---

- If the file has access control elements, and an identity field matches the source\_init\_id, the input\_dcb->src\_concur\_cntl must be a subset of the values stored in the conc\_access field of struct Ft\_access\_control\_element.
- If the file has access control elements, and an identity field matches the dest\_init\_id, the input\_dcb->dest\_concur\_cntl must be a subset of the values stored in the conc\_access field of struct Ft\_access\_control\_element.
- You must supply matching passwords in input\_dcb->source\_file\_passwords for FT\_FA\_READ and FT\_FA\_READ\_ATTRIBUTE file actions if the following conditions exist.
  - If the file you are copying from contains an identity (in Ft\_access\_control\_element) that matches the source\_init\_id AND
  - If passwords exist for these file actions
- You must supply matching passwords in input\_dcb->dest\_file\_passwords if the file you copying to exists and contains an identity (in Ft\_access\_control\_element) that matches the dest\_init\_id and if passwords exist for these file actions.
  - If the file is an FTAM-1, FTAM-3, or INTAP-1 document type, you must supply matching passwords for FT\_FA\_DELETE\_FILE and FT\_FA\_REPLACE.
  - If the file is an FTAM-2 document type, you must supply matching passwords for FT\_FA\_DELETE\_FILE and FT\_FA\_INSERT.

### **Ft\_fcopy\_in\_dcb**

```
struct Ft_fcopy_in_dcb {
    Ft_initiator_identity      source_init_id;
    Ft_single_file_pw          source_filestore_pw;
    Ft_account                  source_account;
    struct Ft_file_passwords   source_file_passwords;
    Ft_initiator_identity      dest_init_id;
    Ft_single_file_pw          dest_filestore_pw;
    Ft_account                  dest_account;
    struct Ft_file_passwords   dest_file_passwords;
    Ft_single_file_pw          create_file_pw;
    enum Ft_delete_overwrite   overwrite;
    struct Ft_concurrency_control *src_concur_cntl;
    struct Ft_concurrency_control *dest_concur_cntl;
};
```

Using High Level, Context Free Functions  
**Copying and Moving FTAM Files (HLCF)**

**Ft\_fcopy\_out\_dcb**

```
struct Ft_fcopy_out_dcb {
    Uint32 size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

**ft\_fcopy() Parameters**

<b>ft_fcopy() Parameter</b>	<b>Type</b>	<b>Description</b>
source_dirname	Optional Input	Directory distinguished name of the source filestore; NULL implies local filestore
source_filename	Mandatory Input	Path name of the file you are copying; must be in the syntax of the real filestore
destination_dirname	Optional Input	Directory distinguished name of the destination filestore; NULL implies local filestore
destination_filename	Optional Input	Path name of the file to which you are copying; NULL implies source_filename; must be in the syntax of the real filestore
ae_label	Mandatory Input	Unique identifier of ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Unique identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->source_init_id	Optional Input	Source login account



<b>ft_copy() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->source_filestore_pw	Optional Input	Login password for the source_init_id account
input_dcb->source_account	Optional Input	Identifies who is responsible for accumulated storage charges on source file; HP-UX responders accept, but ignore, this value.
input_dcb->source_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for FT_FA_READ and FT_FA_READ_ATTRIBUTE. Set source_file_passwords only if the file you are copying from has an identity (in Ft_access_control_element) that matches source_init_id and if passwords exist for these two actions
input_dcb->dest_init_id	Optional Input	Destination login account
input_dcb->dest_filestore_pw	Optional Input	Login password for the dest_init_id account
input_dcb->dest_account	Optional Input	Identifies who is responsible for accumulated storage charges on destination file; HP-UX responders accept, but ignore, this value.
input_dcb->dest_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for the actions FT_FA_DELETE_FILE (FTAM-1, FTAM-2, FTAM-3, INTAP-1), FT_FA_INSERT (FTAM-2), and FT_FA_REPLACE (FTAM-1, FTAM-3, INTAP-1). Set dest_file_passwords only if the file has an identity (in Ft_access_control_element) that matches dest_init_id and if passwords exist for these actions
input_dcb->create_file_pw	Optional Input	Establishes permission to create files in the destination filestore
input_dcb->overwrite	Optional Input	Determines action taken if the destination file exists

Using High Level, Context Free Functions  
 Copying and Moving FTAM Files (HLCF)

<b>ft_fcopy()</b> <b>Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> src_concur_cntl	Optional Input (Used only with non-HP-UX responders.)	Sets the concurrency control for source_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the source_init_id
input_dcb-> dest_concur_cntl	Optional Input (Used only with non-HP-UX responders.)	Sets the concurrency control for destination_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the dest_init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the attributes that uniquely identify the destination file
inout_dcb->charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_fmove()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fmove(source_dirname, source_filename, destination_dirname,
         destination_filename, ae_label, return_event_name,
         input_dcb, inout_dcb)
Ae_dir_name          source_dirname;
Ft_filename          source_filename;
Ae_dir_name          destination_dirname;
Ft_filename          destination_filename;
Ae_label             *ae_label;
Local_event_name     return_event_name;
struct Ft_fmove_in_dcb *input_dcb;
struct Ft_fmove_out_dcb **inout_dcb;
```

The `ft_fmove()` function moves a file from one filestore to another (i.e., moves `source_filename` in the `source_dirname` filestore to `destination_filename` in the `destination_dirname` filestore). The new file contains the source file attributes and those attributes that are automatically set (e.g., `filesize`).

- If the file you are moving to already exists, the outcome depends on how you set `input_dcb->overwrite`. If `input_dcb->overwrite` is set to `FT_RECREATE_FILE`, FTAM copies the source file over the existing destination file. If the value is `FT_DONT_RECREATE_FILE`, FTAM does not copy the source file over the existing file, the function fails, and you receive an error.

---

### NOTE

---

The following points do not apply to HP-UX FTAM responders. HP-UX FTAM responders do not use these fields. The information is provided here because other implementations may use these fields.

- If the file has access control elements, and an identity field matches the `source_init_id`, the `input_dcb->src_concur_cntl` must be a subset of the values in the `conc_access` field of the access control element.
- If the file has access control elements, and an identity field matches the `dest_init_id`, the `input_dcb->dest_concur_cntl` must be a subset of the values in the `conc_access` field of the access control element.

Using High Level, Context Free Functions  
Copying and Moving FTAM Files (HLCF)

- You must supply matching passwords in `input_dcb->source_file_passwords` for `FT_FA_READ`, `FT_FA_READ_ATTRIBUTE`, and `FT_FA_DELETE_FILE` file actions if the following conditions exist.
  - If the file you are moving contains an identity (in `Ft_access_control_element`) that matches the `source_init_id` AND
  - If passwords exist for these file actions
- You must supply matching passwords in `input_dcb->dest_file_passwords` if the file you moving to exists and contains an identity (in `Ft_access_control_element`) that matches the `dest_init_id` and if passwords exist for these file actions.
  - If the file is an FTAM-1, FTAM-3, or INTAP-1 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_REPLACE`.
  - If the file is an FTAM-2 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_INSERT`.

### **Ft\_fmove\_in\_dcb**

```
struct Ft_fmove_in_dcb {
    Ft_initiator_identity    source_init_id;
    Ft_single_file_pw       source_filestore_pw;
    Ft_account               source_account;
    struct Ft_file_passwords source_file_passwords;
    Ft_initiator_identity    dest_init_id;
    Ft_single_file_pw       dest_filestore_pw;
    Ft_account               dest_account;
    struct Ft_file_passwords dest_file_passwords;
    Ft_single_file_pw       create_file_pw;
    enum Ft_delete_overwrite overwrite;
    struct Ft_concurrency_control *src_concur_cntl;
    struct Ft_concurrency_control *dest_concur_cntl;
};
```

### **Ft\_fmove\_out\_dcb**

```
struct Ft_fmove_out_dcb {
    Uint32 size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

### ft\_fmove() Parameters

<b>ft_fmove() Parameter</b>	<b>Type</b>	<b>Description</b>
source_dirname	Optional Input	Directory distinguished name of the source filestore; NULL implies local filestore
source_filename	Mandatory Input	Path name of the file you are moving; must be in the syntax of the real filestore
destination_dirname	Optional Input	Directory distinguished name of the destination filestore; NULL implies local filestore
destination_filename	Optional Input	Path name of the file to which you are moving; NULL implies source_filename; must be in the syntax of the real filestore
ae_label	Mandatory Input	Uniquely identifier of ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Uniquely identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb-> source_init_id	Optional Input	Source login account
input_dcb-> source_filestore_pw	Optional Input	Login password for the source_init_id account
input_dcb-> source_account	Optional Input	Identifies who is responsible for accumulated storage charges on source file; HP-UX responders accept, but ignore this value

Using High Level, Context Free Functions  
 Copying and Moving FTAM Files (HLCF)

<b>ft_fmove() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->source_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for FT_FA_READ, FT_FA_READ_ATTRIBUTE, and FT_FA_DELETE_FILE. Set source_file_passwords only if the file you are moving has an identity (in Ft_access_control_element) that matches source_init_id and if passwords exist for these three actions
input_dcb->dest_init_id	Optional Input	Destination login account
input_dcb->dest_filestore_pw	Optional Input	Login password for the dest_init_id account
input_dcb->dest_account	Optional Input	Identifies who is responsible for accumulated storage charges on destination file; HP-UX responders accept, but ignore, this value
input_dcb->dest_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for the actions FT_FA_DELETE_FILE (FTAM-1, FTAM-2, FTAM-3, INTAP-1), FT_FA_INSERT (FTAM-2), and FT_FA_REPLACE (FTAM-1, FTAM-3, INTAP-1). Set dest_file_passwords only if the file has an identity (in Ft_access_control_element) that matches dest_init_id, and if passwords exist for these actions
input_dcb->create_file_pw	Optional Input	Establishes permission to create files in the destination filestore
input_dcb->overwrite	Optional Input (Used only with non-HP-UX responders.)	Determines action taken if the destination file exists

<b>ft_fmove() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->src_concur_cntl	Optional Input (Used only with non—HP—UX responders.)	Sets the concurrency control for source_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the source_init_id
input_dcb->dest_concur_cntl	Optional Input (Used only with non—HP—UX responders.)	Sets the concurrency control for destination_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the dest_init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP—UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the attributes that uniquely identify the destination file
inout_dcb->charging	Output	HP—UX responders do not use charging, though it may contain information from other responders
inout_dcb->diagnostic	Output	Provides ISO-specific error information

Using High Level, Context Free Functions  
Copying and Moving FTAM Files (HLCF)

## ft\_fcopy\_aet

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fcopy_aet(source_ascii_ddn, source_filename,
             destination_ascii_ddn, destination_filename,
             ae_label, return_event_name, input_dcb, inout_dcb)
char
Ft_filename                                *source_ascii_ddn;
char                                       source_filename;
Ft_filename                                *destination_ascii_ddn;
char                                       destination_filename;
Ae_label                                   *ae_label;
Local_event_name                          return_event_name;
struct Ft_fcopy_in_dcb                    *input_dcb;
struct Ft_fcopy_out_dcb                   **inout_dcb;
```

The `ft_fcopy_aet()` function copies one FTAM file (contents and attributes) to another FTAM file. The original file remains intact. The new file contains the source file attributes and those attributes that are automatically set (e.g., filesize).

- Both `ft_fcopy_aet()` and `ft_fcopy()` can be used for copying files. However, `ft_fcopy_aet()` sends both the called AE title and calling AE title to the remote system while `ft_fcopy()` sends only the calling AE title. Also, in `ft_fcopy_aet()`, the DDN is passed directly as an ASCII string without any conversion while in `ft_fcopy()`, the DDN must be converted and passed as an `Ae_dir_name`.
- If the file you are copying to already exists, the outcome depends on how you set `input_dcb->overwrite`. If `input_dcb->overwrite` is set to `FT_RECREATE_FILE`, FTAM copies the source file over the existing destination file. If the value is `FT_DONT_RECREATE_FILE`, FTAM does not copy the source file over the existing file, the function fails, and you receive an error message.

---

**NOTE**

The following points do not apply to HP-UX FTAM responders. HP-UX FTAM responders do not use these fields. The information is provided here because other implementations may use these fields.

- If the file has access control elements, and an identity field matches the `source_init_id`, the `input_dcb->src_concur_cntl` must be a subset of the values stored in the `conc_access` field of struct `Ft_access_control_element`.



- If the file has access control elements, and an identity field matches the `dest_init_id`, the `input_dcb->dest_concur_cntl` must be a subset of the values stored in the `conc_access` field of struct `Ft_access_control_element`.
- You must supply matching passwords in `input_dcb->source_file_passwords` for `FT_FA_READ` and `FT_FA_READ_ATTRIBUTE` file actions if the following conditions exist.
  - If the file you are copying from contains an identity (in `Ft_access_control_element`) that matches the `source_init_id` AND
  - If passwords exist for these file actions
- You must supply matching passwords in `input_dcb->dest_file_passwords` if the file you copying to exists and contains an identity (in `Ft_access_control_element`) that matches the `dest_init_id` and if passwords exist for these file actions.
  - If the file is an FTAM-1, FTAM-3, or INTAP-1 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_REPLACE`.
  - If the file is an FTAM-2 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_INSERT`.

### **Ft\_fcopy\_in\_dcb**

```
struct Ft_fcopy_in_dcb {
    Ft_initiator_identity      source_init_id;
    Ft_single_file_pw         source_filestore_pw;
    Ft_account                 source_account;
    struct Ft_file_passwords  source_file_passwords;
    Ft_initiator_identity      dest_init_id;
    Ft_single_file_pw         dest_filestore_pw;
    Ft_account                 dest_account;
    struct Ft_file_passwords  dest_file_passwords;
    Ft_single_file_pw         create_file_pw;
    enum Ft_delete_overwrite  overwrite;
    struct Ft_concurrency_control *src_concur_cntl;
    struct Ft_concurrency_control *dest_concur_cntl;
};
```

Using High Level, Context Free Functions  
**Copying and Moving FTAM Files (HLCF)**

**Ft\_fcopy\_out\_dcb**

```
struct Ft_fcopy_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

**ft\_fcopy\_aet() Parameters**

<b>ft_fcopy_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
source_ascii_ddn	Optional Input	Directory distinguished name of the source filestore in ASCII; NULL implies local filestore
source_filename	Mandatory Input	Path name of the file you are copying; must be in the syntax of the real filestore
destination_ascii_ddn	Optional Input	Directory distinguished name of the destination filestore in ASCII; NULL implies local filestore
destination_filename	Optional Input	Path name of the file to which you are copying; NULL implies source_filename; must be in the syntax of the real filestore
ae_label	Mandatory Input	Unique identifier of ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Unique identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->source_init_id	Optional Input	Source login account

<b>ft_fcopy_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> source_filestore_pw	Optional Input	Login password for the source_init_id account
input_dcb-> source_account	Optional Input	Identifies who is responsible for accumulated storage charges on source file; HP-UX responders accept, but ignore, this value
input_dcb-> source_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for FT_FA_READ and FT_FA_READ_ATTRIBUTE. Set source_file_passwords only if the file you are copying from has an identity (in Ft_access_control_element) that matches source_init_id and if passwords exist for these two actions
input_dcb-> >dest_init_id	Optional Input	Destination login account
input_dcb-> dest_filestore_pw	Optional Input	Login password for the dest_init_id account
input_dcb-> dest_account	Optional Input	Identifies who is responsible for accumulated storage charges on destination file; HP-UX responders accept, but ignore, this value.
input_dcb-> dest_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for the actions FT_FA_DELETE_FILE (FTAM-1, FTAM-2, FTAM-3, INTAP-1).. Set dest_file_passwords only if the file has an identity (in Ft_access_control_element) that matches dest_init_id and if passwords exist for these actions
input_dcb-> create_file_pw	Optional Input	Establishes permission to create files in the destination filestore
input_dcb->overwrite	Optional Input	Determines action taken if the destination file exists

Using High Level, Context Free Functions  
 Copying and Moving FTAM Files (HLCF)

<b>ft_fcopy_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> src_concur_cntl	Optional Input (Used only with non-HP- UX responders.)	Sets the concurrency control for source_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the source_init_id
input_dcb-> dest_concur_cntl	Optional Input (Used only with non-HP- UX responders.)	Sets the concurrency control for destination_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the dest_init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the attributes that uniquely identify the destination file
inout_dcb->charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_fmove\_aet()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fmove_aet(source_ascii_ddn, source_filename,
             destination_ascii_ddn, destination_filename,
             ae_label, return_event_name, input_dcb, inout_dcb)
char
Ft_filename                                *source_dirname;
char                                        *source_filename;
Ft_filename                                *destination_dirname;
char                                        *destination_filename;
Ae_label                                    *ae_label;
Local_event_name                           return_event_name;
struct Ft_fmove_in_dcb                     *input_dcb;
struct Ft_fmove_out_dcb                    **inout_dcb;
```

The `ft_fmove_aet()` function moves a file from one filestore to another (i.e., moves `source_filename` in the `source_dirname` filestore to `destination_filename` in the `destination_dirname` filestore). The new file contains the source file attributes that are automatically set (e.g., `filesize`).

- Both `ft_fmove_aet()` and `ft_fmove()` are used for moving files. However, `ft_fmove_aet()` sends both the Called AE Title and calling AE title to the remote system while `ft_fmove()` sends only the calling AE title. Also, in `ft_fmove_aet()`, the DDN is passed directly as an ASCII string without conversion while in `ft_fmove()`, the DDN has to be converted and passed as an `Ae_dir_name`.
- If the file you are moving to already exists, the outcome depends on how you set `input_dcb->overwrite`. If `input_dcb->overwrite` is set to `FT_RECREATE_FILE`, FTAM copies the source file over the existing destination file. If the value is `FT_DONT_RECREATE_FILE`, FTAM does not copy the source file over the existing file, the function fails, and you receive an error.

---

**NOTE**

The following points do not apply to HP-UX FTAM responders. HP-UX FTAM responders do not use these fields. The information is provided here because other implementations may use these fields.

- If the file has access control elements, and an identity field matches the `source_init_id`, the `input_dcb->src_concur_cntl` must be a subset of the values stored in the `conc_access` field of `struct Ft_access_control_element`.

Using High Level, Context Free Functions  
Copying and Moving FTAM Files (HLCF)

- If the file has access control elements, and an identity field matches the `dest_init_id`, the `input_dcb->dest_concur_cntl` must be a subset of the values stored in the `conc_access` field of struct `Ft_access_control_element`.
- You must supply matching passwords in `input_dcb->source_file_passwords` for `FT_FA_READ`, `FT_FA_READ_ATTRIBUTE`, and `FT_FA_DELETE_FILE` file actions if the following conditions exist.
  - If the file you are moving contains an identity (in `Ft_access_control_element`) that matches the `source_init_id` AND
  - If passwords exist for these file actions
- You must supply matching passwords in `input_dcb->dest_file_passwords` if the file you are moving to exists and contains an identity (in `Ft_access_control_element`) that matches the `dest_init_id` and if passwords exist for these file actions.
  - If the file is an FTAM-1, FTAM-3, or INTAP-1 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_REPLACE`.
  - If the file is an FTAM-2 document type, you must supply matching passwords for `FT_FA_DELETE_FILE` and `FT_FA_INSERT`.

### **Ft\_fmove\_in\_dcb**

```
struct Ft_fmove_in_dcb {
    Ft_initiator_identity    source_init_id;
    Ft_single_file_pw       source_filestore_pw;
    Ft_account              source_account;
    struct Ft_file_passwords source_file_passwords;
    Ft_initiator_identity    dest_init_id;
    Ft_single_file_pw       dest_filestore_pw;
    Ft_account              dest_account;
    struct Ft_file_passwords dest_file_passwords;
    Ft_single_file_pw       create_file_pw;
    enum Ft_delete_overwrite overwrite;
    struct Ft_concurrency_control *src_concur_cntl;
    struct Ft_concurrency_control *dest_concur_cntl;
};
```

### **Ft\_fmove\_out\_dcb**

```

struct Ft_fmove_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    enum Ft_action_result                 action_result;
    struct Ft_attributes                  attributes;
    struct Ft_charging                    *charging;
    struct Ft_diagnostic                  *diagnostic;
};

```

### **ft\_fmove\_aet() Parameters**

<b>ft_fmove_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
source_ascii_ddn	Optional Input	Directory distinguished name of the source filestore in ASCII; NULL implies local filestore
source_filename	Mandatory Input	Path name of the file you are moving; must be in the syntax of the real filestore
destination_ascii_ddn	Optional Input	Directory distinguished name of the destination filestore in ASCII; NULL implies local filestore
destination_filename	Optional Input	Path name of the file to which you are moving; NULL implies source_filename; must be in the syntax of the real filestore
ae_label	Mandatory Input	Uniquely identifier of ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Uniquely identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->source_init_id	Optional Input	Source login account
input_dcb->source_filestore_pw	Optional Input	Login password for the source_init_id account

Using High Level, Context Free Functions  
 Copying and Moving FTAM Files (HLCF)

<b>ft_fmmove_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> source_account	Optional Input	Identifies who is responsible for accumulated storage charges on source file; HP-UX responders accept, but ignore this value
input_dcb-> source_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for FT_FA_READ, FT_FA_READ_ATTRIBUTE, and FT_FA_DELETE_FILE. Set source_file_passwords only if the file you are moving has an identity (in Ft_access_control_element) that matches source_init_id and if passwords exist for these three actions
input_dcb-> dest_init_id	Optional Input	Destination login account
input_dcb-> dest_filestore_pw	Optional Input	Login password for the dest_init_id account
input_dcb-> dest_account	Optional Input	Identifies who is responsible for accumulated storage charges on destination file; HP-UX responders accept, but ignore, this value
input_dcb-> dest_file_passwords	Optional Input (Used only with non-HP-UX responders.)	Structure containing the passwords for the actions FT_FA_DELETE_FILE (FTAM-1, FTAM-2, FTAM-3, INTAP-1), FT_FA_INSERT (FTAM-2), and FT_FA_REPLACE (FTAM-1, FTAM-3, INTAP-1). Set dest_file_passwords only if the file has an identity (in Ft_access_control_element) that matches dest_init_id, and if passwords exist for these actions
input_dcb-> create_file_pw	Optional Input	Establishes permission to create files in the destination filestore
input_dcb-> >overwrite	Optional Input (Used only with non-HP-UX responders.)	Determines action taken if the destination file exists



<b>ft_fmove_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> src_concur_cntl	Optional Input (Used only with non—HP— UX responders.)	Sets the concurrency control for source_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the source_init_id
input_dcb-> dest_concur_cntl	Optional Input (Used only with non—HP— UX responders.)	Sets the concurrency control for destination_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the dest_init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP—UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> attributes	Output	Contains values of the attributes that uniquely identify the destination file
inout_dcb->charging	Output	HP—UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## Reading and Changing Attributes (HLCF)

Use `ft_frattributes()` and `ft_frattributes_aet` to read file attributes, and `ft_fcattributes()` and `ft_fcattributes_aet()` to change file attributes by making only one call.

### `ft_frattributes()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_frattributes (dirname, filename, ae_label,
                return_event_name, input_dcb, inout_dcb)
Ae_dir_name      dirname;
Ft_filename      filename;
Ae_label         *ae_label;
Local_event_name return_event_name;
struct Ft_frattributes_in_dcb *input_dcb;
struct Ft_frattributes_out_dcb **inout_dcb;
```

Use `ft_frattributes()` to read file attributes. Examples of using `ft_frattributes()` are as follows.

You might want to use `ft_frattributes()` to determine the creation dates when deleting files older than a specific date. Another example of using `ft_frattributes()` is to know how to set destination file attributes when copying files with low level calls.

- If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.
- The `input_dcb->attribute_names` field defaults to indicate all attributes in the `attribute_groups` negotiated.

- You must supply matching passwords in `input_dcb->file_passwords` for `FT_FA_READ` and `FT_FA_READ_ATTRIBUTE` file actions if the following conditions exist.
  - If the file contains an identity (in `Ft_access_control_element`) that matches the `init_id` AND
  - If passwords exist for these file actions
- HP-UX responders set `storage_account`, `future_filesize`, `legal_qualifications`, and `private_use` so that the indication no value available (0) returns when you invoke `ft_frattributes()`.

### **Ft\_frattributes\_in\_dcb**

```
struct Ft_frattributes_in_dcb {
    Ft_initiator_identity      init_id;
    Ft_single_file_pw          filestore_pw;
    struct Ft_file_passwords   file_passwords;
    Ft_attribute_names         attribute_names;
    Ft_account                 account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### **Ft\_frattributes\_out\_dcb**

```
struct Ft_frattributes_out_dcb {
    Uint32      size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

## **ft\_frattributes() Parameters**

<b>ft_frattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
dirname	Optional Input	Directory distinguished name of the filestore; NULL implies the local filestore
filename	Mandatory Input	Path name of the file that has the attributes you are reading; must be in the syntax of the real filestore
ae_label	Mandatory Input	Unique identifier for ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Unique identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->init_id	Optional Input	Login account
input_dcb->filestore_pw	Optional Input	Login password for init_id
input_dcb->file_passwords	Optional Input	Structure containing the passwords for FT_FA_READ and FT_FA_READ_ATTRIBUTE. Set file_passwords only if the file has an identity (in Ft_access_control_element) that matches init_id and if passwords exist for these two actions
input_dcb->attribute_names	Optional Input	Mask that indicates which file attributes to read
input_dcb->account	Optional Input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value

<b>ft_frattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb-> concurrency_control	Optional input (Used only with non-HP-UX responders.)	Sets the concurrency control for filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX-specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> attributes	Output	Contains values of the attributes of the file you read
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## ft\_fcattributes()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fcattributes (dirname, filename, ae_label,
                return_event_name, input_dcb, inout_dcb)
Ae_dir_name      dirname;
Ft_filename      filename;
Ae_label         *ae_label;
Local_event_name return_event_name;
struct Ft_fcattributes_in_dcb *input_dcb;
struct Ft_fcattributes_out_dcb **inout_dcb;
```

Use `ft_fcattributes()` to change file attributes. For example, you can change an attribute such as `access_control` to place file access restrictions on a specific user.

- The `attribute_groups` negotiated dictate the attributes you can change.

### Attributes You Can Change

`access_control`  
`file_availability`  
`filename`  
`future_filesize*`  
`legal_qualification*`  
`private_use*`  
`storage_account*`

### Attributes You Cannot Change

`contents_type`  
`date_time_of_attribute_mod`  
`date_time_of_creation`  
`date_time_of_modification`  
`date_time_of_read` `filesize`  
`identity_of_attribute_mod`  
`identity_of_creator`  
`identity_of_modifier`  
`identity_of_reader`  
`permitted_actions`

\* HP-UX responders accept, but ignore, these values.

- If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.

- You must supply matching passwords in `input_dcb->file_passwords` for `FT_FA_READ` and `FT_FA_CHANGE_ATTRIBUTE` file actions if the following conditions exist.
  - If the file contains an identity (in `Ft_access_control_element`) that matches the `init_id` AND
  - If passwords exist for these file actions

### **Ft\_fcattributes\_in\_dcb**

```
struct Ft_fcattributes_in_dcb {
    Ft_initiator_identity      init_id;
    Ft_single_file_pw          filestore_pw;
    struct Ft_file_passwords   file_passwords;
    Ft_attribute_names         attribute_names;
    struct Ft_attributes       attributes;
    Ft_account                 account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### **Ft\_fcattributes\_out\_dcb**

```
struct Ft_fcattributes_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_fcattributes() Parameters**

<b>ft_fcattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>dirname</code>	Optional Input	Directory distinguished name of the filestore; NULL implies the local filestore
<code>filename</code>	Mandatory Input	Path name of the file that has the attributes you are changing; must be in the syntax of the real filestore
<code>ae_label</code>	Mandatory Input	Unique identifier for <code>ftam_init</code> you want to service the request; if pointing to a NULL location, the interface uses a currently activated <code>ftam_init</code> or activates and deactivates a <code>ftam_init</code>

Using High Level, Context Free Functions  
 Reading and Changing Attributes (HLCF)

<b>ft_fattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
	Output	Unique identifies the ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->attribute_names	Mandatory Input	Mask that indicates which file attributes to change
input_dcb->attributes.values	Mandatory Input	Contains values you want to change for the attributes given in attribute_names
input_dcb->init_id	Optional Input	Login account
input_dcb->filestore_pw	Optional Input	Login password for init_id
input_dcb->account	Optional input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value
input_dcb->concurrency_control	Optional input Used only when using access control	Sets the concurrency control for filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the attributes that changed; access_passwords do not return



<b>ft_fcattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information
input_dcb-> file_passwords	Optional Input	Structure containing the passwords for FT_FA_READ and FT_FA_CHANGE_ATTRIBUTE. Set file_passwords only if the file has an identity (in Ft_access_control_element) that matches init_id and if passwords exist for these two actions

## ft\_frattributes\_aet()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_frattributes_aet (ascii_ddn, filename, ae_label,
                    return_event_name, input_dcb, inout_dcb)
char                *ascii_ddn;
Ft_filename         filename;
Ae_label           *ae_label;
Local_event_name   return_event_name;
struct Ft_frattributes_in_dcb *input_dcb;
struct Ft_frattributes_out_dcb **inout_dcb;
```

Use `ft_frattributes_aet()` to read file attributes. Examples of using `ft_frattributes_aet()` are as follows.

You might want to use `ft_frattributes_aet()` to determine the creation dates when deleting files older than a specific date. Another example of using `ft_frattributes_aet()` is to know how to set destination file attributes when copying files with low level calls.

- Both `ft_frattributes_aet()` and `ft_frattributes()` are used for reading file attributes. However, `ft_frattributes_aet()` sends both the Called AE Title and calling AE title to the remote system while `ft_frattributes()` sends only the calling AE title. Also, in `ft_frattributes_aet()`, the DDN is passed directly as an ASCII string without any conversion while in `ft_frattributes()`, the DDN has to be converted and passed as an `Ae_dir_name`.
- If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.
- The `input_dcb->attribute_names` field defaults to indicate all attributes in the `attribute_groups` negotiated.
- You must supply matching passwords in `input_dcb->file_passwords` for `FT_FA_READ` and `FT_FA_READ_ATTRIBUTE` file actions if the following conditions exist.
  - If the file contains an identity (in `Ft_access_control_element`) that matches the `init_id` AND
  - If passwords exist for these file actions

- HP-UX responders set `storage_account`, `future_filesize`, `legal_qualifications`, and `private_use` so that the indication no value available (0) returns when you invoke `ft_frattributes_aet()`.

### **Ft\_frattributes\_in\_dcb**

```
struct Ft_frattributes_in_dcb {
    Ft_initiator_identity    init_id;
    Ft_single_file_pw       filestore_pw;
    struct Ft_file_passwords file_passwords;
    Ft_attribute_names      attribute_names;
    Ft_account              account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### **Ft\_frattributes\_out\_dcb**

```
struct Ft_frattributes_out_dcb {
    Uint32      size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_frattributes\_aet() Parameters**

<b>ft_frattributes_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>ascii_ddn</code>	Optional Input	Directory distinguished name of the filestore in ASCII; NULL implies the local filestore
<code>filename</code>	Mandatory Input	Path name of the file that has the attributes you are reading; must be in the syntax of the real filestore
<code>ae_label</code>	Mandatory Input	Unique identifier for <code>ftam_init</code> you want to service the request; if pointing to a NULL location, the interface uses a currently activated <code>ftam_init</code> or activates and deactivates a <code>ftam_init</code>
	Output	Unique identifier for <code>ftam_init</code> that serviced the request

Using High Level, Context Free Functions  
 Reading and Changing Attributes (HLCF)

<b>ft_frattributes_ae t() Parameter</b>	<b>Type</b>	<b>Description</b>
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->init_id	Optional Input	Login account
input_dcb-> filestore_pw	Optional Input	Login password for init_id
input_dcb-> file_passwords	Optional Input	Structure containing the passwords for FT_FA_READ and FT_FA_READ_ATTRIBUTE. Set file_passwords only if the file has an identity (in Ft_access_control_element) that matches init_id and if passwords exist for these two actions
input_dcb-> attribute_names	Optional Input	Mask that indicates which file attributes to read
input_dcb->account	Optional Input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value
input_dcb-> concurrency_control	Optional input (Used only with non-HP-UX responders.)	Sets the concurrency control for filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the init_id
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX-specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request

<b>ft_frattributes_ae t() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->attributes	Output	Contains values of the attributes of the file you read
inout_dcb->charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_fcattributes\_aet()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fcattributes_aet (ascii_ddn, filename, ae_label,
                    return_event_name, input_dcb, inout_dcb)
char                *ascii_ddn;
Ft_filename         filename;
Ae_label            *ae_label;
Local_event_name    return_event_name;
struct Ft_fcattributes_in_dcb *input_dcb;
struct Ft_fcattributes_out_dcb **inout_dcb;
```

Use `ft_fcattributes_aet()` to change file attributes. For example, you can change an attribute such as `access_control` to place file access restrictions on a specific user.

- Both `ft_fcattributes_aet()` and `ft_fcattributes()` are used for changing file attributes. However, `ft_fcattributes_aet()` sends both the Called AE Title and calling AE title to the remote system while `ft_fcattributes()` sends only the calling AE title. Also, in `ft_fcattributes_aet()`, the DDN is passed directly as an ASCII string without any conversion while in `ft_fcattributes()`, the DDN has to be converted and passed as an `Ae_dir_name`.
- The `attribute_groups` negotiated dictate the attributes you can change.

### Attributes You Can Change

`access_control`  
`file_availability`  
`filename`  
`future_filesize*`  
`legal_qualification*`  
`private_use*`  
`storage_account*`

### Attributes You Cannot Change

`contents_type`  
`date_time_of_attribute_mod`  
`date_time_of_creation`  
`date_time_of_modification`  
`date_time_of_read`  
`filesize`  
`identity_of_attribute_mod`  
`identity_of_creator`  
`identity_of_modifier`  
`identity_of_reader`  
`permitted_actions`

\* HP-UX responders accept, but ignore, these values.

- If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.
- You must supply matching passwords in `input_dcb->file_passwords` for `FT_FA_READ` and `FT_FA_CHANGE_ATTRIBUTE` file actions if the following conditions exist.
  - If the file contains an identity (in `Ft_access_control_element`) that matches the `init_id` AND
  - If passwords exist for these file actions

### **Ft\_fcattributes\_in\_dcb**

```
struct Ft_fcattributes_in_dcb {
    Ft_initiator_identity    init_id;
    Ft_single_file_pw       filestore_pw;
    struct Ft_file_passwords file_passwords;
    Ft_attribute_names      attribute_names;
    struct Ft_attributes     attributes;
    Ft_account              account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### **Ft\_fcattributes\_out\_dcb**

```
struct Ft_fcattributes_out_dcb {
    Uint32    size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

## **ft\_fattributes\_aet() Parameters**

<b>ft_fattributes_aet () Parameter</b>	<b>Type</b>	<b>Description</b>
ascii_ddn	Optional Input	Directory distinguished name of the filestore in ASCII; NULL implies the local filestore
filename	Mandatory Input	Path name of the file that has the attributes you are changing; must be in the syntax of the real filestore
ae_label	Mandatory Input	Uniquely identifier for ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Uniquely identifies the ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->attribute_names	Mandatory Input	Mask that indicates which file attributes to change
input_dcb->attributes.values	Mandatory Input	Contains values you want to change for the attributes given in attribute_names
input_dcb->init_id	Optional Input	Login account
input_dcb->filestore_pw	Optional Input	Login password for init_id
input_dcb->account	Optional input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value
input_dcb->concurrency_control	Optional input (Used only when using access control)	Sets the concurrency control for filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the init_id



<b>ft_fcattributes_aet () Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> attributes	Output	Contains values of the attributes that changed; access_passwords do not return
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information
input_dcb-> file_passwords	Optional Input	Structure containing the passwords for FT_FA_READ and FT_FA_CHANGE_ATTRIBUTE. Set file_passwords only if the file has an identity (in Ft_access_control_element) that matches init_id and if passwords exist for these two actions

## Deleting Files (HLCF)

To delete files, invoke `ft_fdelete()` or `ft_delete_aet()`.

### `ft_fdelete()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fdelete (dirname, filename, ae_label, return_event_name,
            input_dcb, inout_dcb)
Ae_dir_name      dirname;
Ft_filename      filename;
Ae_label         *ae_label;
Local_event_name return_event_name;
struct Ft_fdelete_in_dcb *input_dcb;
struct Ft_fdelete_out_dcb **inout_dcb;
```

The `ft_fdelete()` function deletes files from the filestore. Once you delete the file, you cannot recover it.

If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.

### `Ft_fdelete_in_dcb`

```
struct Ft_fdelete_in_dcb {
    Ft_initiator_identity    init_id;
    Ft_single_file_pw       filestore_pw;
    Ft_single_file_pw       delete_file_pw;
    Ft_account               account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### `Ft_fdelete_out_dcb`

```
struct Ft_fdelete_out_dcb {
    Uint32      size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

### ft\_delete() Parameters

<b>ft_delete() Parameter</b>	<b>Type</b>	<b>Description</b>
dirname	Optional Input	Directory distinguished name of the filestore; NULL implies the local filestore
filename	Mandatory Input	Path name of the file you are deleting; must be in the syntax of the real filestore
ae_label	Mandatory Input	Unique identifier for ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Unique identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->init_id	Optional Input	Login account
input_dcb->filestore_pw	Optional Input	Login password for init_id
input_dcb->delete_file_pw	Optional Input	Establishes permission to delete files from the filestore; HP-UX responders accept, but ignore, this value
input_dcb->account	Optional Input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value
input_dcb->concurrency_control	Optional input (Used only with non-HP-UX responders.)	Sets the concurrency control for filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the init_id

Using High Level, Context Free Functions  
Deleting Files (HLCF)

<b>ft_fdelete() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## ft\_fdelete\_aet()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_fdelete_aet (ascii_ddn, filename, ae_label,
               return_event_name, input_dcb, inout_dcb)
char          *ascii_ddn;
Ft_filename  filename;
Ae_label     *ae_label;
Local_event_name return_event_name;
struct Ft_fdelete_in_dcb *input_dcb;
struct Ft_fdelete_out_dcb **inout_dcb;
```

The `ft_fdelete_aet()` function deletes files from the filestore. Once you delete the file, you cannot recover it.

If the responder supports the `conc_access` field of struct `Ft_access_control_element`, and the file has access control elements, and an identity field matches the `init_id`, the `input_dcb->concurrency_control` must be a subset of the values stored in the `conc_access`. HP-UX responders do not support the `conc_access` field of struct `Ft_access_control_element`.

Both `ft_fdelete_aet()` and `ft_fdelete()` are used for deleting files. However, `ft_fdelete_aet()` sends both the Called AE Title and calling AE title to the remote system while `ft_fdelete()` sends only the calling AE title. Also, in `ft_fdelete_aet()`, the DDN is passed directly as an ASCII string without any conversion while in `ft_fdelete()`, the DDN has to be converted and passed as an `Ae_dir_name`.

## Ft\_fdelete\_in\_dcb

```
struct Ft_fdelete_in_dcb {
    Ft_initiator_identity    init_id;
    Ft_single_file_pw        filestore_pw;
    Ft_single_file_pw        delete_file_pw;
    Ft_account               account;
    struct Ft_concurrency_control *concurrency_control;
};
```

## Ft\_fdelete\_out\_dcb

```
struct Ft_fdelete_out_dcb {
    Uint32                size;
    struct Api_rc          result;
    enum Ft_action_result action_result;
    struct Ft_charging     *charging;
    struct Ft_diagnostic   *diagnostic;
};
```

### **ft\_fdelete\_aet() Parameters**

<b>ft_fdelete_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
ascii_ddn	Optional Input	Directory distinguished name of the filestore in ASCII; NULL implies the local filestore
filename	Mandatory Input	Path name of the file you are deleting; must be in the syntax of the real filestore
ae_label	Mandatory Input	Unique identifier for ftam_init you want to service the request; if pointing to a NULL location, the interface uses a currently activated ftam_init or activates and deactivates a ftam_init
	Output	Unique identifier for ftam_init that serviced the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->init_id	Optional Input	Login account
input_dcb->filestore_pw	Optional Input	Login password for init_id
input_dcb->delete_file_pw	Optional Input	Establishes permission to delete files from the filestore; HP-UX responders accept, but ignore, this value
input_dcb->account	Optional Input	Identifies who is responsible for accumulated file storage charges; HP-UX responders accept, but ignore, this value
input_dcb->concurrency_control	Optional input (Used only with non-HP-UX responders.)	Sets the concurrency control for source_filename; compared to conc_access in struct Ft_access_control_element if an identity field matches the source_init_id

<b>ft_fdelete_aet() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code (MAP 3.0 error) and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

Using High Level, Context Free Functions  
**Deleting Files (HLCF)**





To use context sensitive FTAM operations, you must first activate an application entity (AE) and then establish a connection. The AE that you activate is an FTAM initiator (`ftam_init`). Use connection management functions to control activations and connections. Refer to the “Example Programs” chapter for model programs using the connection management functions.

---

## Chapter Overview

This chapter describes the following functions in the order listed.

<b>Task to Perform</b>	<b>Function Used to Perform the Task</b>	<b>Description</b>
Starting and Stopping Application Entities	ft_aeactivation()	Activate ftam_init
	ft_aedeactivation()	Deactivate ftam_init
Establishing and Removing Connections	ft_connect()	Establish connection from ftam_init to the responder
	ft_rrequest()	Release connection from the responder
	ft_aereset()	Abort all active connections for a ftam_init, and then reset ftam_init
Aborting Connection Requests	ft_abort()	Abort one connection at any time
	ft_ireceive()	Determine who aborted the call and receive abort indication information

## Connection Establishment Process

A typical connection establish and release routine (algorithm) is as follows. You do not have an option regarding the sequence of these events, with the exception of step 4. (Refer to Figure 6-1.)

1. Activate ftam\_init (using ft\_aeactivation()).
2. Establish the connection (using ft\_connect()) between ftam\_init and the responder. Note that during the life of the ftam\_init, you can establish and release connections arbitrarily.
3. Use FTAM operations as needed.
4. Release the connection (using ft\_rrequest()). Alternatively, you can use one of the following functions:
  - ft\_aereset() to release all active connections for that ftam\_init.
  - ft\_abort() used at any time aborts a connection.
5. Deactivate ftam\_init (using ft\_aedeactivation()).

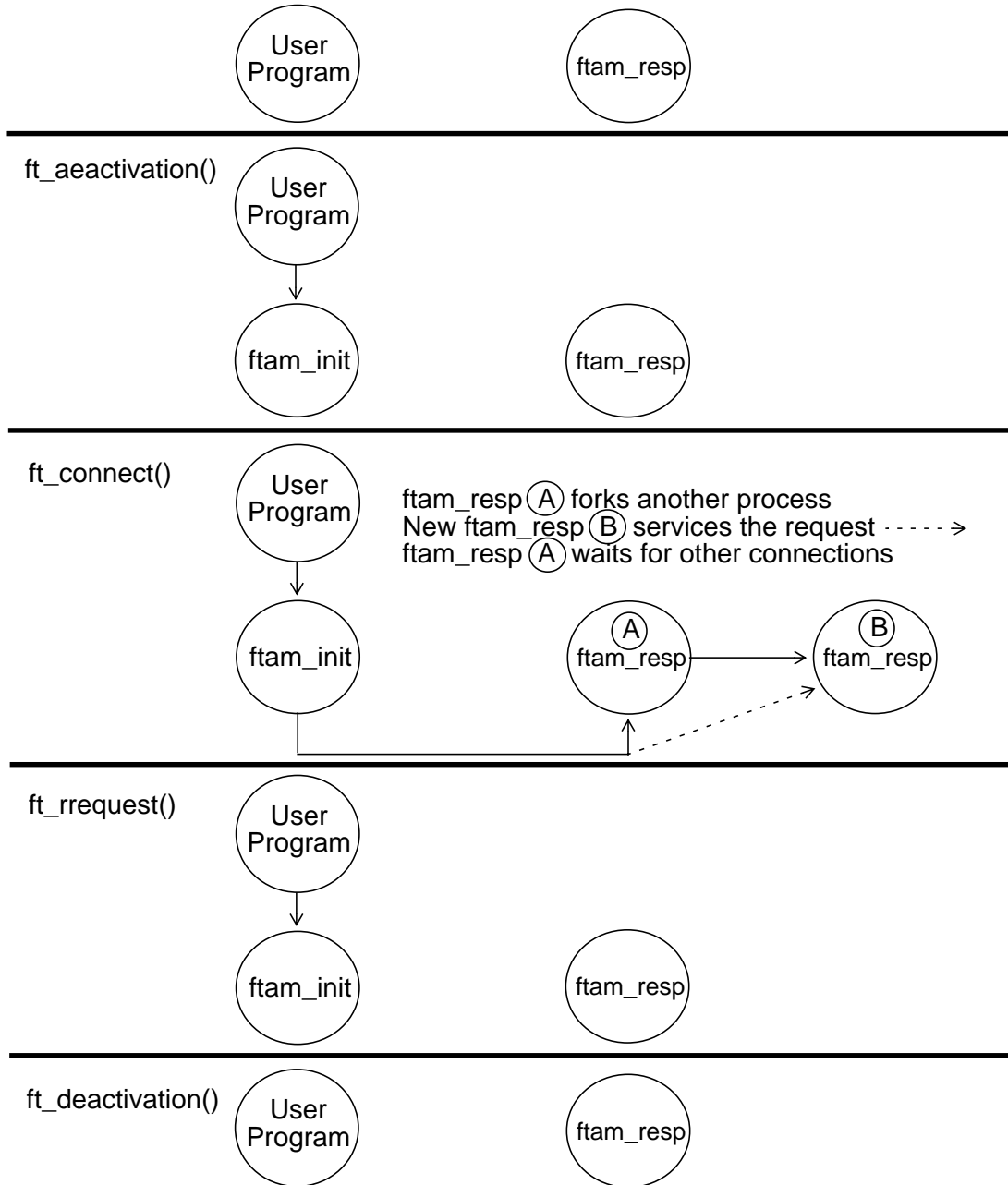
---

**NOTE**

This chapter does not explain the parameter structures. For detailed structure information (parameter settings), refer to the “FTAM Data Structures” chapter.

---

**Figure 6-1 Connection Establishment Process**



## Starting and Stopping Application Entities

Use `ft_aeactivation()` and `ft_aedeactivation()` to respectively start and stop `ftam_init`.

### `ft_aeactivation()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_aeactivation(my_dir_name, return_event_name, input_dcb,
               inout_dcb, ae_label)
Ae_dir_name      my_dir_name;
Local_event_name return_event_name;
struct Ft_aeactivate_in_dcb *input_dcb;
struct Ft_output  **inout_dcb;
Ae_label         *ae_label;
```

You must invoke `ft_aeactivation()` before establishing a connection. By passing `my_dir_name` (the local `ftam_init` directory distinguished name) as input, the interface returns the `ae_label` of the activated `ftam_init`. Note, directory distinguished names are in the Initial Configuration Store (ICS); they are defined during system configuration and when nodes are added to the network.

Using `my_dir_name` requires that you set up `Ae_dir_dn`. You can do so manually (as explained in the “FTAM Data Structures” chapter) or you can call the `convert_ddn` utility. This utility eases the setting of `Ae_dir_dn` by automatically setting it and its sub-structures for you. The source for `convert_ddn` is in `/opt/ftam/demos/cnvrtddd.c`. (For instructions, read the on-line README document in this directory.) The “Example Programs” chapter also contains the source for `convert_ddn`.

### `Ft_aeactivation_in_dcb`

```
struct Ft_aeactivate_in_dcb {
    struct Octet_string      authentication;
    enum Ae_title_option    my_ae_title_option;
    Ae_title                 *my_ae_title;
};
```

## Ft\_output

```
struct Ft_output {
    Uint32          size;
    struct Api_rc   result;
};
```

## ft\_aeactivation() Parameters

<b>ft_aeactivation() Parameter</b>	<b>Type</b>	<b>Description</b>
my_dir_name	Mandatory Input	Directory distinguished name identifying the AE (ftam_init) you want to activate
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->authentication	Optional Input	ftam_init accepts, but ignores, this value
input_dcb->my_ae_title_option	Optional Input	Dictates the interpretation of my_ae_title. If it indicates Dir_dist_name_option, my_dir_name must be usable as an address for the local ftam_init. Default is No_value_option.
input_dcb->my_ae_title	Optional Input	Optional ae_title sent to the peer AE, which can be either a directory distinguished name or an object_id; determined by how you set my_ae_title_option.
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
ae_label	Output	Uniquely identifies the AE (ftam_init) that serviced the request

## ft\_aedeactivation()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_aedeactivation(ae_label, return_event_name, inout_dcb)
Ae_label          *ae_label;
Local_event_name  return_event_name;
struct Ft_output  **inout_dcb;
```

Use `ft_aedeactivation()` to deactivate the AE (`ftam_init`) servicing the request.

### Ft\_output

```
struct Ft_output {
    Uint32          size;
    struct Api_rc   result;
};
```

### ft\_aedeactivation() Parameters

<b>ft_aedeactivation() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>ae_label</code>	Mandatory Input	Uniquely identifies the AE ( <code>ftam_init</code> ) you want to deactivate
<code>return_event_name</code>	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
<code>inout_dcb-&gt;size</code>	Mandatory input if using <code>inout_dcb</code>	Size (in Octets) of the <code>inout_dcb</code> structure and data
<code>inout_dcb-&gt;result</code>	Output	Pointer to the struct <code>Api_rc</code> containing the outcome of the operation: <code>result_code</code> and <code>vendor_code</code> (HP-UX—specific error)



---

## Establishing and Removing Connections

Use the `ft_connect()`, `ft_rrequest()`, and `ft_aeraset()` functions to connect, release connections, and reset connections in an orderly manner.

### `ft_connect()`

```
#include </opt/ftam/include/map.h>
#include </opt/ftam/include/mapftam.h>
Return_code
ft_connect(ae_label, return_event_name, called_dir_name,
           input_dcb, inout_dcb, connection_id)
Ae_label          ae_label;
Local_event_name  return_event_name;
Ae_dir_name       called_dir_name;
struct Ft_connect_in_dcb *input_dcb;
struct Ft_connect_out_dcb *inout_dcb;
Connection_id     *connection_id;
```

Use `ft_connect()` to establish a connection from `ftam_init` to an FTAM responder. Remember, you must first invoke `ft_aeactivation()` to start `ftam_init`.

You will use the returned `connection_id` on all subsequent, context sensitive FTAM calls to uniquely identify the connection to the desired filestore.

- When invoking `ft_connect()`, the document types, functional\_units, and service\_classes are negotiated with the responder. When the request is complete, check these fields in the `inout_dcb` to see if the responder accepted these values, accepted a subset of these values, or rejected these values.
- You must set either the `input_dcb->called_presentation_address` or the `called_dir_name`. If you set both fields, `input_dcb->called_presentation_address` takes precedence. Refer to the “FTAM Data Structures” chapter to set either of these parameters.

Specifying these fields requires that you set up the appropriate structures. You can do so manually (as explained in the “FTAM Data Structures” chapter) or you can call one of the following utilities.

- To have the system set the `P_address`, call `convert_paddr`; the source is in `/opt/ftam/demos/cnvrtr_addr.c`.

Managing HP FTAM/9000 Connections  
Establishing and Removing Connections

- To have the system set the Ae\_dir\_name, call convert\_ddn; the source is in /opt/ftam/demos/cnvrt\_addr.c.

For instructions on using these utilities, read the on-line README document in /opt/ftam/demos/cnvrt\_addr.c. The “Example Programs” chapter also contains the source for convert\_paddr and convert\_ddn.

- For HP-UX responders, the login name (connect\_in\_info.initiator\_identity) must exist on the remote node, and the file\_store\_pw must be that user's password. Other responders may or may not have similar expectations.
- The inout\_dcb structure contains the responder's reply to the requested connection parameters.

### **Ft\_connect\_in\_dcb**

```
struct Ft_connect_in_dcb {
    struct P_address          called_presentation_address;
    enum Ae_title_option     called_ae_title_option;
    Ae_title                 *called_ae_title;
    Uint32                  called_ae_invoke_id;
    Uint32                  called_ap_invoke_id;
    Uint8                   number_of_retry;
    Uint32                  delay_between_retry;
    Uint32                  connection_resource_wait_timer;
    struct Object_id        context_name;
    struct Ft_connect_req_info connect_in_info;
};
struct Ft_connect_req_info {
    Ft_service_class        service_class;
    Ft_functional_units     functional_units;
    Ft_attribute_groups     attribute_groups;
    enum Ft_qos             quality_of_service;
    struct Ft_contents_type_element *contents_type_list;
    Ft_initiator_identity   initiator_identity;
    Ft_account              account;
    Ft_single_file_pw       file_store_pw;
};
```

### **Ft\_connect\_out\_dcb**

```

struct Ft_connect_out_dcb {
    Uint32                                size;
    struct Api_rc                          result;
    struct P_address
responding_presentation_address;
    Ae                                      responding_ae;
    Uint32                                  responding_ae_invoke_id;
    Uint32                                  responding_ap_invoke_id;
    struct Object_id                        context_name;
    struct Ft_connect_cnf_info              *connect_out_info;
};
    
```

```

struct Ft_connect_cnf_info {
    enum Ft_state_result                    state_result;
    enum Ft_action_result                   action_result;
    Ft_service_class                        service_class;
    Ft_functional_units                     functional_units;
    Ft_attribute_groups                     attribute_groups;
    enum Ft_qos                             quality_of_service;
    struct Ft_contents_type_element         contents_type_list;
    struct Ft_diagnostic                    *diagnostic;
    enum Ft_acse_result                     acse_result;
    struct Ft_acse_diagnostic               acse_diag;
};
    
```

### **ft\_connect() Parameters**

<b>ft_connect() Parameter</b>	<b>Type</b>	<b>Description</b>
ae_label	Mandatory Input	Uniquely identifies the AE (ftam_init) you want to service the request
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
called_dir_name	Mandatory Input if no called_presentation_address given; otherwise, Optional Input	Directory distinguished name identifying the responder to which you want to connect

Managing HP FTAM/9000 Connections  
Establishing and Removing Connections

<b>ft_connect() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->called_ presentation_address	Mandatory Input if no called_dir_name given; otherwise, Optional Input	Contains addressing information for the remote responder
input_dcb-> called_ae_title_option	Optional Input	Dictates the interpretation of called_ae_title If it indicates Dir_dist_name_option, called_dir_name must be usable as an address for the remote responder Default is No_value_option
input_dcb-> called_ae_title	Optional Input	Optional ae_title sent to the peer AE Can be either a directory distinguished name or an object_id Determined by how you set called_ae_title_option
input_dcb-> called_ae_invoke_id	Optional Input	HP-UX responders accept, but ignore, this undefined value; default setting is NO_VALUE
input_dcb-> called_ap_invoke_id	Optional Input	HP-UX responders accept, but ignore, this undefined value; default setting is NO_VALUE
input_dcb-> number_of_retry	Optional Input	Number of times (in tenths of seconds) to retry the connection attempt; default is zero
input_dcb-> delay_between_retry	Optional Input	Amount of time (in tenths of seconds) to wait between retrying to establish the connection; default is zero
input_dcb->connection_ resource_wait_timer	Optional Input	Maximum time (in tenths of seconds) to wait for an available association resource; default is zero
input_dcb-> context_name	Mandatory Input	Association Control Service Element (ACSE) context name; you must set this value to 1 0 8571 1 1

<b>ft_connect() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->connect_in_info.service_class	Mandatory Input	Determines whether you transfer, access, or manage a file on this connection
input_dcb->connect_in_info.functional_units	Mandatory Input	Determines which functions are available per connection
input_dcb->connect_in_info.attribute_groups	Mandatory Input	Specifies the set of attributes (level of support) available on the association
input_dcb->connect_in_info.quality_of_service	Optional Input	Determines the level of available recovery; FTAM does not automatically attempt recovery
input_dcb->connect_in_info.contents_type_list	Optional Input	Specifies the allowable document type for the connection
input_dcb->connect_in_info.initiator_identity	Optional Input	Login account name, which must exist on the node to which you are connecting
input_dcb->connect_in_info.account	Optional Input	HP-UX responders accept, but ignore, this value
input_dcb->connect_in_info.file_store_pw	Optional Input	Login password for the initiator_identity
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->responding_presentation_address	Output	presentation_address returned by the responder
inout_dcb->responding_ae	Output	ae_title returned by the responder
inout_dcb->responding_ae_invoke_id	Output	HP-UX responders accept, but ignore this undefined value
inout_dcb->responding_ap_invoke_id	Output	HP-UX responders accept, but ignore this undefined value

Managing HP FTAM/9000 Connections  
Establishing and Removing Connections

<b>ft_connect() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb-> context_name	Output	context_name accepted by the responder
inout_dcb->connect_ out_info->state_result	Output	Indicates whether the request moved through regimes; if a failure returns, you will also receive an action_result of FT_AR_PERMANENT_ERROR
inout_dcb->connect_ out_info->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->connect_ out_info->service_class	Output	Highest service_class the responder supports
inout_dcb-> connect_out_info-> functional_unit	Output	functional_units accepted by the responder
inout_dcb-> connect_out_info-> attribute_groups	Output	attribute_groups accepted by the responder
inout_dcb-> connect_out_info-> quality_of_service	Output	Level of available recovery; FTAM does not automatically attempt recovery
inout_dcb-> connect_out_info-> contents_type_list	Output	contents_types (document types) accepted by the responder
inout_dcb->connect_ out_info->diagnostic	Output	Provides ISO-specific error information
inout_dcb->connect_ out_info->acse_result	Output	Indicates overall result of ACSE processing
inout_dcb-> connect_out_info-> acse_diagnostic	Output	Provides Association Control Service Element (ACSE) diagnostic information
connection_id	Output	Uniquely identifies the connection to the filestore; use this connection_id as input to subsequent calls to identify the connection

## ft\_rrequest()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_rrequest (connection_id, return_event_name, input_dcb,
            inout_dcb)
Connection_id          connection_id;
Local_event_name       return_event_name;
struct Ft_relreq_in_dcb *input_dcb;
struct Ft_relreq_out_dcb **inout_dcb;
```

Use `ft_rrequest` to release a specific connection; afterwards, the connection no longer exists; you must establish a new connection (using `ft_connect()`).

The `ft_rrequest()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.

## Ft\_relreq\_in\_dcb

```
struct Ft_relreq_in_dcb {
    struct Ft_release_req_info  release_in_info;
};
struct Ft_release_req_info {
    char      *ft_rel_info;
};
```

## Ft\_relreq\_out\_dcb

```
struct Ft_relreq_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    struct Ft_release_cnf_info *output_info;
};
struct Ft_release_cnf_info {
    struct Ft_charging *charging;
};
```

## ft\_rrequest() Parameters

<b>ft_rrequest() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies the connection you want to release
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb	Optional Input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->output_info->charging	Output	HP-UX responders do not use charging, though it may contain information from other responders

## ft\_aereset()

```
#include </opt/ftam/include/map.h>
#include </opt/ftam/include/mapftam.h>
Return_code
ft_aereset(ae_label, return_event_name, inout_dcb)
Ae_label          ae_label;
Local_event_name  return_event_name;
struct Ft_output  **inout_dcb;
```

Use `ft_aereset()` to abort all open connections and return `ftam_init` to its initial state.

## Ft\_output

```
struct Ft_output {
    Uint32          size;
    struct Api_rc   result;
};
```



### **ft\_aereset() Parameters**

<b>ft_aereset() Parameter</b>	<b>Type</b>	<b>Description</b>
ae_label	Mandatory Input	Uniquely identifies the AE (ftam_init) you want to reset
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

## Aborting Connections

Use `ft_abort()` to abort a connection and `ft_ireceive()` to determine why you received an abort indication.

### `ft_abort()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_abort(connection_id, return_event_name, input_dcb,
          inout_dcb)
Connection_id          connection_id;
Local_event_name      return_event_name;
struct Ft_abort_in_dcb *input_dcb;
struct Ft_output      **inout_dcb;
```

Use `ft_abort()` to abruptly terminate a connection at any time (i.e., in any regime).

---

#### NOTE

If you abort while writing data, you could lose any or all data written during that transfer; you will not, however, lose the source file.

---

### `Ft_abort_in_dcb`

```
struct Ft_abort_in_dcb {
    struct Ft_abort_req_info info;
};

struct Ft_abort_req_info {
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

### `Ft_output`

```
struct Ft_output {
    Uint32 size;
    struct Api_rc result;
};
```

## ft\_abort() Parameters

<b>ft_abort() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies the specific filestore connection to abort
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->info.action_result	Mandatory Input	Conveys the reason for the abort
input_dcb->info.diagnostic	Optional Input	ISO-specific error information; HP-UX responders accept, but ignore, this value
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

## ft\_ireceive()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_ireceive(connection_id, return_event_name, inout_dcb,
            indication_name)
Connection_id          connection_id;
Local_event_name       return_event_name;
struct Ft_indication_out_dcb *inout_dcb;
Uint32                 *indication_name;
```

If you receive the FTE003\_ABORT\_IND\_RCVD error, call ft\_ireceive()

- to determine whether the abort was caused by the system (initiator, responder, or network) or by a user, and
- to receive diagnostics.

## Aborting Connections

The information regarding the type of abort returns in the `indication_name` parameter and may be useful for troubleshooting purposes.

All diagnostics associated with the abort are returned in the `inout_dcb`.

### **Ft\_indication\_out\_dcb**

```
struct Ft_indication_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    union Ft_specific_indication_info    info;
};

union Ft_specific_indication_info {
    struct Ft_aabort_ind_info            aabort_info;
    struct Ft_pabort_ind_info            pabort_info;
};

struct Ft_aabort_ind_info {
    enum Ft_action_result                action_result;
    struct Ft_diagnostic                 *diagnostic;
};

struct Ft_pabort_ind_info {
    enum Ft_action_result                action_result;
    struct Ft_diagnostic                 *diagnostic;
};
```

### **ft\_ireceive() Parameters**

<b>ft_ireceive() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies the aborted connection
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->info.aabort_info.action_result	Output	Specifies that an application caused the abort Specifies the overall success or failure of the request
inout_dcb->info.aabort_info.diagnostic	Output	Specifies that an application caused the abort; provides ISO-specific error information
inout_dcb->info.pabort_info.action_result	Output	Specifies that the system caused the abort Specifies the overall success or failure of the request
inout_dcb->info.pabort_info.diagnostic	Output	Specifies that the system caused the abort; provides ISO-specific error information
indication_name	Output	Specifies if the abort was a system (FT_PABORT_IND) or application (FT_AABORT_IND) abort; dictates which union Ft_specific_indication_info contains the output information

Managing HP FTAM/9000 Connections  
**Aborting Connections**

---

**7**

**Managing and Accessing HP  
FTAM/9000 Files**

## Managing and Accessing HP FTAM/9000 Files

After establishing connections, you can manage and access FTAM files. Refer to the “Example Programs” chapter for model programs using these functions.



---

## Chapter Overview

This chapter describes the following functions in the order listed.

<b>Task to Perform</b>	<b>Function Used to Perform the Task</b>	<b>Description</b>
Gaining Access to Files	ft_create()	Create files
	ft_select()	Select files
	ft_open()	Open files (LLCS)
	ft_close()	Close files (LLCS)
	ft_delete()	Delete files (LLCS)
	ft_deselect()	Deselect files
Opening and Closing Files	ft_fopen()	Select or create, and then open files (HLCS)
	ft_fclose()	Close and then deselect or delete files (HLCS)
Reading and Changing Attributes	ft_rattributes()	Read attributes (LLCS)
	ft_cattributes()	Change attributes (LLCS)
Locating and Erasing FTAM Files	ft_locate()	Locate FADUs within files
	ft_erase()	Erase files
Grouping FTAM Functions	ft_bgroup()	Begin a group of FTAM calls
	ft_egroup()	End a group of FTAM calls

---

**NOTE** This chapter does not explain the parameter structures. For detailed structure information (parameter settings), refer to the “FTAM Data Structures” chapter.

---

## Gaining Access to FTAM Files

After establishing a connection, either select (`ft_select()`) or create (`ft_create()`) a file for further manipulation. You must always select a file before opening it (`ft_open()`), though the selection automatically occurs if you create the file.

After performing FTAM operations, close (`ft_close()`) the file and then deselect (`ft_deselect()`) or delete (`ft_delete()`) it. The file remains intact if you deselect it; if you delete the file, it no longer exists.

### `ft_create()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
```

Return code

```
ft_create (connection_id, filename, contents_type,
           requested_access, file_status, return_event_name,
           input_dcb, inout_dcb)
```

Connection_id	connection_id;
Ft_filename	filename;
struct Ft_contents_type	contents_type;
Ft_file_actions	requested_access;
enum Ft_file_status	file_status;
Local_event_name	return_event_name;
struct Ft_create_in_dcb	*input_dcb;
struct Ft_create_out_dcb	**inout_dcb;

Use `ft_create()` to create a new file. If successful, `ft_create()` automatically selects the file.

- Invoke `ft_create()` in the FTAM regime to move to the File Selection regime.
- During `ft_connect()`, you must negotiate the `FT_FU_LTD_MGMT` functional\_units to use `ft_create()`.
- If the file already exists, the `file_status` parameter determines whether a new file is created or the request fails. (Refer to the “`Ft_file_status`” section in the “FTAM Data Structures” chapter for options on setting this field.)

- The subset of the `input_dcb->attributes.mask` field (of type struct `Ft_attributes`) that is valid for `ft_create()` is as follows.

<code>FT_AN_ACCESS_CONTROL</code>	<code>FT_AN_LEGAL_QUAL</code>
<code>FT_AN_FILE_AVAILABILITY</code>	<code>FT_AN_PERMITTED_ACTION</code>
<code>FT_AN_FUTURE_FILESIZE</code>	<code>S FT_AN_PRIVATE_USE</code>
<code>FT_AN_ID_OF_CREATOR</code>	<code>FT_AN_STORAGE_ACCOUNT</code>

- If you do not specify `permitted_actions`, FTAM uses the `requested_access` values for the `permitted_actions` values.
- FTAM uses the exposed parameters `filename` and `contents_type` and does not use `input_dcb->attributes.values.filename` and `input_dcb->attributes.values.contents_type`.

### **Ft\_create\_in\_dcb**

```
struct Ft_create_in_dcb {
    struct Ft_attributes          attributes;
    struct Ft_concurrency_control *concurrency_control;
    Ft_single_file_pw            create_file_pw;
    struct Ft_file_passwords     file_passwords;
    Ft_account                   account;
};
```

### **Ft\_create\_out\_dcb**

```
struct Ft_create_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_state_result state_result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_create() Parameters

<b>ft_create() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
filename	Mandatory Input	Path name of the file you are creating; must be in the syntax of the real filestore
contents_type	Mandatory Input	Specifies the document type of the newly created file
requested_access	Mandatory Input	Specifies the actions you want available for the connection (during the File Selection regime and those regimes nested within it)
file_status	Mandatory Input	Determines the action to take if the file already exists
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->attributes	Optional Input	Uniquely identifies the file you are creating; do not use input_dcb->attributes.filename and input_dcb->attributes.contents_type
input_dcb->concurrency_control	Optional Input	File access locks required on the actions specified in requested_access
input_dcb->create_file_pw	Optional Input	Establishes permission to create files in the current filestore; HP-UX responders ignore this value
input_dcb->file_passwords	Optional Input	Passwords associated with the actions specified by requested_access
input_dcb->account	Optional Input	HP-UX responders accept, but ignore, this value
input_dcb->size	Mandatory input if using input_dcb	Size (in Octets) of the input_dcb structure and data

<b>ft_create() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->state_result	Output	Indicates whether the request moved from the FTAM regime to the File Selection regime; if a failure returns, you will also receive an action_result of FT_AR_PERMANENT_ERROR
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the attributes of the file you created (those attributes accepted by the responder)
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_select()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
```

```
Return_code
ft_select (connection_id, filename, requested_access,
          return_event_name, input_dcb, inout_dcb)
```

```
Connection_id          connection_id;
Ft_filename            filename;
Ft_file_actions       requested_access;
Local_event_name      return_event_name;
struct Ft_select_in_dcb *input_dcb;
struct Ft_select_out_dcb **inout_dcb;
```

Use `ft_select()` to choose an existing file for file management or access (e.g., reading attributes, opening a file). You select a file based only on its filename.

- Invoke `ft_select()` in the FTAM regime to move to the File Selection regime.

### Gaining Access to FTAM Files

- FTAM uses the exposed parameters `filename` and `contents_type` and does not use the corresponding values stored in `input_dcb->attributes`.
- If the file you are selecting has access control, the following conditions exist.
  - The `requested_access` parameter must be a subset of the values stored in `action_list` field in struct `Ft_access_control_element`.
  - FTAM will detect conflicts between the `initiator_id` and the `identity` field of the `access_control_element`.
  - The HP-UX implementation of FTAM does not use the `file_passwords` parameter. Other implementations might.

### **Ft\_select\_in\_dcb**

```
struct Ft_select_in_dcb {
    struct Ft_attributes          attributes;
    struct Ft_file_passwords     file_passwords;
    Ft_account                   account;
    struct Ft_concurrency_control *concurrency_control;
};
```

### **Ft\_select\_out\_dcb**

```
struct Ft_select_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_state_result state_result;
    enum Ft_action_result action_result;
    struct Ft_attributes attributes;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_select() Parameters**

<b>ft_select() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>connection_id</code>	Mandatory Input	Uniquely identifies a specific connection to the filestore
<code>filename</code>	Mandatory Input	Path name of the file you are selecting; must be in the syntax of the real filestore

<b>ft_select() Parameter</b>	<b>Type</b>	<b>Description</b>
requested_access	Mandatory Input	Specifies the actions you want available for the connection (during the File Selection regime and those regimes nested within it)
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->attributes	Optional input	FTAM does not use this field; provided for future use
input_dcb->file_passwords	Optional Input	Passwords associated with the actions specified by requested_access
input_dcb->account	Optional Input	HP-UX responders accept, but ignore, this value
input_dcb->concurrency_control	Optional Input	File access locks required on the actions specified in requested_access
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->state_result	Output	Indicates whether the request moved from the FTAM to the File Selection regime; if a failure returns, you will also receive an action_result of FT_AR_PERMANENT_ERROR
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	Contains values of the filename attribute of the file you selected HP-UX responders accept, but ignore, this value
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_open() (LLCS)

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_open (connection_id, processing_mode, contents_type,
        return_event_name, input_dcb, inout_dcb)

Connection_id          connection_id;
Ft_processing_mode     processing_mode;
struct Ft_contents_type contents_type;
Local_event_name       return_event_name;
struct Ft_open_in_dcb  *input_dcb;
struct Ft_open_out_dcb **inout_dcb;
```

Use `ft_open()` after selecting or creating a file to access operations.

- Invoke `ft_open()` in the File Selection regime to move to the File Open regime.
- During `ft_connect()`, you must negotiate the `FT_FU_READ` or `FT_FU_WRITE` functional\_units to use `ft_open()`.
- The `input_dcb->concurrency_control` parameters must be the same or more restrictive than the `input_dcb->concurrency_control` parameters set on `ft_create()` and `ft_select()`.
- If the file you are opening has access control, the following conditions exist:
  - The `processing_mode` parameter must be a subset of the values stored in `action_list` field in the `access_control_element` corresponding to the `initiator_id` used to establish the connection.
  - The HP-UX implementation of FTAM does not use the `file_passwords` parameter. Other implementations might.

## Ft\_open\_in\_dcb

```
struct Ft_open_in_dcb {
    struct Ft_concurrency_control *concurrency_control;
};
```



## **Ft\_open\_out\_dcb**

```

struct Ft_open_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    enum Ft_state_result                  state_result;
    enum Ft_action_result                  action_result;
    struct Ft_contents_type                contents_type;
    struct Ft_concurrency_control          *concurrency_control;
    struct Ft_diagnostic                   *diagnostic;
};

```

## **ft\_open() Parameters**

<b>ft_open() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
processing_mode	Mandatory Input	Establishes a valid subset of actions negotiated in the File Selection regime
contents_type	Mandatory Input	Specifies the document type
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->concurrency_control	Optional Input	File access locks required on the actions specified in requested_access
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

<b>ft_open() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->state_result	Output	Indicates whether the request moved from the File Selection to the File Open regime; if a failure returns, you will also receive an action_result of FT_AR_PERMANENT_ERROR
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->contents_type	Output	contents_type of the opened file (as accepted by the responder)
inout_dcb->concurrency_control	Output	concurrency_control of the opened file (as accepted by the responder)
inout_dcb->diagnostic	Output	Provides ISO-specific error information

### **ft\_close() (LLCS)**

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
```

```
Return_code
ft_close (connection_id, return_event_name, input_dcb,
          inout_dcb)
```

```
Connection_id          connection_id;
Local_event_name       return_event_name;
char                   *input_dcb;
struct Ft_close_out_dcb **inout_dcb;
```

Use `ft_close()` to close a file when you finish processing it. After closing the file, the only actions you can perform on the file without re-opening it are `ft_deselect()`, `ft_delete()`, `ft_rattributes()`, and `ft_cattributes()`.

- The `ft_close()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.
- Invoke `ft_close()` in the File Open regime to return to the File Selection regime.

## **Ft\_close\_out\_dcb**

```
struct Ft_close_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    enum Ft_action_result                 action_result;
    struct Ft_diagnostic                  *diagnostic;
};
```

## **ft\_close() Parameters**

<b>ft_close() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb	Optional input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_delete() (LLCS)

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_delete (connection_id, return_event_name, input_dcb,
           inout_dcb)

Connection_id          connection_id;
Local_event_name      return_event_name;
char                   *input_dcb;
struct Ft_delete_out_dcb **inout_dcb;
```

Use `ft_delete()` to remove a file from the filestore.

- The `ft_delete()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.
- Invoke `ft_delete()` in the File Selection regime to move back to the FTAM regime after deleting the file.
- During `ft_connect()`, you must negotiate the `FT_FU_LTD_MGMT` `functional_units` to use `ft_delete()`.
- During `ft_select()`, `ft_create()`, and `ft_fopen()` you must specify `FT_FA_DELETE_FILE` in `requested_access` to use `ft_delete()`.

## Ft\_delete\_out\_dcb

```
struct Ft_delete_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_delete() Parameters

<b>ft_close() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb	Optional input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## ft\_deselect()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_deselect (connection_id, return_event_name, input_dcb,
            inout_dcb)

Connection_id          connection_id;
Local_event_name      return_event_name;
char                   *input_dcb;
struct Ft_deselect_out_dcb **inout_dcb;
```

Use `ft_deselect()` to deselect a file when you finish processing, but do not want to remove it. You must re-select the file to perform further actions on it.

- The `ft_deselect()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.
- Invoke `ft_deselect()` in the File Selection regime to move back to the FTAM regime.

## Ft\_deselect\_out\_dcb

```
struct Ft_deselect_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_charging *charging;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_deselect() Parameters

<b>ft_deselect() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>connection_id</code>	Mandatory Input	Uniquely identifies a specific connection to the filestore
<code>return_event_name</code>	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
<code>input_dcb</code>	Optional input	Contains no information; pass a NULL pointer

<b>ft_deselect() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> charging	Output	HP-UX responders do not use charging, though it may contain information from other responders
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## Opening and Closing Files (HLCS)

Use `ft_fopen()` and `ft_fclose()` to open and close files, respectively, using only one high level call.

<code>ft_fopen()</code>	Either selects or creates, and then opens a file
<code>ft_fclose()</code>	Closes a file, and then either deselects or deletes a file

### `ft_fopen()` (HLCS)

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_fopen (connection_id, filename, file_status,
          requested_access, contents_type, return_event_name,
          input_dcb, inout_dcb)

Connection_id          connection_id;
Ft_filename            filename;
enum Ft_file_status    file_status;
Ft_file_actions        requested_access;
struct Ft_contents_type contents_type;
Local_event_name       return_event_name;
struct Ft_fopen_in_dcb *input_dcb;
struct Ft_fopen_out_dcb **inout_dcb;
```

Use `ft_fopen()` to create or select a file and then open it with one call (rather than invoking the low level calls `ft_select()` or `ft_create()` with `ft_open()`).

- Invoke `ft_fopen()` in the FTAM regime to move to the File Open regime.
- If the file already exists, the `file_status` parameter determines whether a new file is created or the request fails. (Refer to the “`Ft_file_status`” section in the “FTAM Data Structures” chapter for options on setting this field.)



- If the file you are selecting and opening has access control, the following conditions exist.
  - The `requested_access` parameter must be a subset of the values stored in `action_list` field in the `access_control_element` corresponding to the `initiator_id` used to establish the connection.
  - FTAM will detect conflicts between the `initiator_id` and the `identity` field of the `access_control_element`.
  - The HP-UX implementation of FTAM does not use the `file_passwords` parameter. Other implementations might.
- If you create a new file, the returned file attributes are in the `inout_dcb->attributes`.
- FTAM uses the exposed parameters `filename` and `contents_type` and does not use `input_dcb->attributes.values.filename` and `input_dcb->attributes.values.contents_type`.
- If you do not specify `permitted_actions`, FTAM uses `requested_access` for `permitted_actions`.
- The `processing_mode` parameter used when the file is opened includes all file actions set in `requested_access` that are valid in the File Open regime.
- If you are selecting and opening a file, but do not know what the document type is, set `contents_type.contents_form` to `FT_CONTENTS_UNKNOWN`.

### **Ft\_fopen\_in\_dcb**

```
struct Ft_fopen_in_dcb {
    struct Ft_attributes          attributes;
    struct Ft_concurrency_control *concurrency_control;
    struct Ft_file_passwords     file_passwords;
    Ft_account                   account;
};
```

### **Ft\_fopen\_out\_dcb**

```

struct Ft_fopen_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    enum Ft_state_result                  state_result;
    enum Ft_action_result                  action_result;
    struct Ft_attributes                   attributes;
    struct Ft_concurrency_control          *concurrency_control;
    struct Ft_diagnostic                    *diagnostic;
};

```

### **ft\_fopen() Parameters**

<b>ft_fopen() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
filename	Mandatory Input	Path name of the file you are opening; must be in the syntax of the real filestore
file_status	Mandatory Input	Determines the action to take if the file already exists
requested_access	Mandatory Input	Specifies the actions you want to perform on the file
contents_type	Mandatory Input	Specifies the document type of the file you are creating or opening
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->attributes	Optional	Uniquely identifies the file you are creating and opening; do not use input_dcb->attributes.values.filename or input_dcb->attributes.values.contents_type
input_dcb->concurrency_control	Optional	File access locks required on the actions specified in requested_access
input_dcb->file_passwords	Optional	Passwords associated with the actions specified by requested_access

<b>ft_fopen() Parameter</b>	<b>Type</b>	<b>Description</b>
input_dcb->account	Optional	HP-UX responders accept, but ignore, this value
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->state_result	Output	Indicates whether the request moved through regimes; if a failure returns, you will also receive an action_result of FT_AR_PERMANENT_ERROR
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->attributes	Output	If you created the file, contains values of the attributes of the file (those attributes accepted by the responder); if you selected the file, contains values of the filename and contents_type attributes
inout_dcb->concurrency_control	Output	concurrency_control of the opened file (as accepted by the responder)
inout_dcb->diagnostic	Output	Provides ISO-specific error information

Managing and Accessing HP FTAM/9000 Files  
Opening and Closing Files (HLCS)

## **ft\_fclose() (HLCS)**

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_fclose (connection_id, delete_action,
           return_event_name, input_dcb, inout_dcb)

Connection_id          connection_id;
enum Ft_delete_action  delete_action;
Local_event_name      return_event_name;
char                   *input_dcb;
struct Ft_fclose_out_dcb **inout_dcb;
```

Use `ft_fclose()` to close a file, and then either deselect or delete it (rather than invoking the low level `ft_close()` function with `ft_deselect()` or `ft_delete()`).

- Invoke `ft_fclose()` in the File Open regime to move back to the FTAM regime.
- The `delete_action` parameter determines whether the file is deleted or deselected.
- The `ft_close()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.

## **Ft\_fclose\_out\_dcb**

```
struct Ft_fclose_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_fclose() Parameters

<b>ft_fclose() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
delete_action	Mandatory Input	Determines whether to delete or deselect the file
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb	Optional input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## Reading and Changing Attributes (LLCS)

Use the `ft_rattributes()` and `ft_cattributes()` to read and change attributes using low level calls.

If you are already in the File Selection regime, it is faster and easier to use low level calls to read and change attributes than to use high level calls (which re-connect and re-select). In other cases, the high level calls may be easier to use.

### `ft_rattributes()` (LLCS)

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_rattributes (connection_id, return_event_name, input_dcb,
               inout_dcb)

Connection_id           connection_id;
Local_event_name       return_event_name;
char                   *input_dcb;
struct Ft_rattributes_out_dcb **inout_dcb;
```

Use `ft_rattributes()` to read attributes.

- Invoke `ft_rattributes()` in the File Selection regime; you do not move to another regime.
- During `ft_connect()`, you must negotiate the `FT_FU_LTD_MGMT` functional\_units to use `ft_rattributes()`.
- During `ft_select()`, you must specify `FT_FA_READ_ATTRIBUTE` in `requested_access`.
- Use `input_dcb->attribute_names` to specify which attributes you want to read. The attributes you read return in `inout_dcb->attributes`.
- HP-UX responders return the indication no value available (0) for the attributes `legal_qualification`, `private_use`, `future_filesize`, and `storage_account`.

### **Ft\_rattributes\_in\_dcb**

```
struct Ft_rattributes_in_dcb {
    Ft_attribute_names  attribute_names;
};
```

### **Ft\_rattributes\_out\_dcb**

```
struct Ft_rattributes_out_dcb {
    Uint32                size;
    struct Api_rc         result;
    enum Ft_action_result action_result;
    struct Ft_attributes  attributes;
    struct Ft_diagnostic  *diagnostic;
};
```

### **ft\_rattributes() Parameters**

<b>ft_rattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
input_dcb-> attribute_names	Optional Input	Indicates the file attributes to read; default is all file attributes associated with the attribute groups negotiated on ft_connect()
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation

<b>ft_rattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> attributes	Output	Contains values of the attributes of the file you read (those attributes accepted by the responder)
inout_dcb-> diagnostic	Output	Holds ISO-specific error information

### **ft\_cattributes() (LLCS)**

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>

Return_code
ft_cattributes (connection_id, return_event_name, inout_dcb,
               inout_dcb)

Connection_id           connection_id;
Local_event_name       return_event_name;
struct Ft_cattributes_in_dcb *input_dcb;
struct Ft_cattributes_out_dcb **inout_dcb;
```

Use `ft_cattributes()` to change file attributes. For example, you can change an attribute such as `access_control` to allow a specific user to access a file.

- Invoke `ft_cattributes()` in the File Selection regime; you do not move to another regime.
- During `ft_connect()`, you must negotiate the `FT_FU_ENH_MGMT_functional_units` to use `ft_cattributes()`.
- During `ft_select()`, you must specify `FT_FA_CHANGE_ATTRIBUTE` in `requested_access`.
- The `attributes_groups` negotiated on the `ft_connect()` request dictates the attributes you can change.
- The `input_dcb->attribute_names` parameters takes precedence over the `input_dcb->attributes.mask` field.



### Attributes You Can Change

access\_control  
file\_availability  
filename  
future\_filesize\*  
legal\_qualification\*  
private\_use\*  
storage\_account\*

### Attributes You Cannot Change

contents\_type  
date\_time\_of\_attribute\_mod  
date\_time\_of\_creation  
date\_time\_of\_modification  
date\_time\_of\_read  
filesize  
identity\_of\_attribute\_mod  
identity\_of\_creator  
identity\_of\_modifier  
identity\_of\_reader  
permitted\_actions

\* HP-UX responders accept, but ignore, these values.

### Ft\_cattributes\_in\_dcb

```
struct Ft_cattributes_in_dcb {  
    Ft_attribute_names      attribute_names;  
    struct Ft_attributes    attributes;  
};
```

### Ft\_cattributes\_out\_dcb

```
struct Ft_cattributes_out_dcb {  
    Uint32      size;  
    struct Api_rc result;  
    enum Ft_action_result action_result;  
    struct Ft_attributes attributes;  
    struct Ft_diagnostic *diagnostic;  
};
```

## **ft\_cattributes() Parameters**

<b>ft_cattributes() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb-> attribute_names	Mandatory Input	Mask that indicates which file attributes to change
input_dcb-> attributes	Mandatory Input	Specifies the values of the attributes to change; attributes.values contains new values for each attribute specified in attribute_names
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> attributes	Output	Contains values of the attributes you changed (those attributes accepted by the responder)
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

---

## Locating and Erasing FTAM Files

Use the `ft_locate()` to locate FADUs (FTAM-2) and `ft_erase()` to erase FTAM files.

### `ft_locate()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_locate (connection_id, fadu_identity,
           return_event_name, input_dcb, inout_dcb)
Connection_id      connection_id;
Ft_fadu_identity   fadu_identity;
Local_event_name   return_event_name;
char               *input_dcb;
struct Ft_locate_out_dcb **inout_dcb;
```

Use `ft_locate()` to locate a specific FADU within a file. Hence, use `ft_locate()` only on FTAM-2 document types. For example, you might want to locate the current FADU in an FTAM-2 file so that you can read it.

- The `ft_locate()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.
- Invoke `ft_locate()` in the File Open regime; you do not move to another regime.
- During `ft_connect()`, you must negotiate the `FT_FU_FILE_ACCESS` functional\_units to use `ft_locate()`.

### `Ft_locate_out_dcb`

```
struct Ft_locate_out_dcb {
    Uint32      size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_fadu_identity *fadu_identity;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_locate() Parameters**

<b>ft_locate() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
fadu_identity	Mandatory Input	Identifies the FADU to locate (FTAM-2 only)
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb	Optional input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->fadu_identity	Output	fadu_identity of the FADU you located (FTAM-2 only)
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_erase()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_erase (connection_id, fadu_identity, return_event_name,
         input_dcb, inout_dcb)
Connection_id          connection_id;
Ft_fadu_identity       fadu_identity;
Local_event_name       return_event_name;
char                   *input_dcb;
struct Ft_erase_out_dcb **inout_dcb;
```

Use `ft_erase()` to erase a file's contents; attributes are not erased.

- To erase the entire file's contents, the `fadu_identity.fadu_location` must be `FT_FIRST` for FTAM-1, FTAM-3, and INTAP-1 and must be `FT_BEGIN` for FTAM-2.
- The `ft_erase()` `input_dcb` parameter does not contain information; therefore, pass a `NULL` pointer for the `input_dcb` parameter.
- Invoke `ft_erase()` in the File Open regime; you do not move to another regime.
- During `ft_connect()`, you must negotiate the `FT_FU_FILE_ACCESS` `functional_units` to use `ft_erase()`.
- During `ft_open()`, you must specify `FT_PM_ERASE` in `processing_mode` to use `ft_erase()`.

## Ft\_erase\_out\_dcb

```
struct Ft_erase_out_dcb {
    Uint16          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_erase() Parameters**

<b>ft_locate() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
fadu_identity	Mandatory Input	Identifies the FADU to erase
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero.
input_dcb	Optional input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb-> diagnostic	Output	Provides ISO-specific error information

## Grouping LLCS FTAM Functions

Use `ft_bgroup()` and `ft_egroup()` to begin and end the grouping of low level FTAM functions and thus, increase performance. The `ftam_init` places these functions into one presentation service data unit (PSDU) before sending it across the network.

For a group to be successful, all functions moving from one regime to another must be successful (regardless of whether the calls that do not move through regimes succeed).

If a failure occurs on a function that crosses regimes and if the SUCCESS count does not equal or exceed threshold, no remaining calls, except `ft_egroup()`, are processed and you receive an error.

## Allowable Groupings

Refer to the following guidelines for grouping FTAM functions and for reading the following table. The table lists the legal groupings with the associated sequence of LLCS functions (from left to right).

**EXAMPLE:** The first grouping (#1) in the table reads as follows.

1. Call `ft_bgroup()`.
2. Call either `ft_create()` if the file does not exist or `ft_select()` if it does.
3. Call either or both `ft_rattributes()` and `ft_cattributes()`.
4. Call `ft_open()` followed by `ft_egroup()`.

- All groups must begin with `ft_bgroup()` and end with `ft_egroup()`.
- The first function you want to call after `ft_bgroup()` determines which regime you must be in to call `ft_bgroup()`.
- You must call all functions within the group asynchronously except for `ft_egroup()`, which can be synchronous or asynchronous.
- If more than one function is listed in a "Mandatory" column, you must perform both functions in the order listed.

Managing and Accessing HP FTAM/9000 Files  
**Grouping LLCS FTAM Functions**

- You can specify either or both optional functions, or can choose not to use them. If you call both optional functions, you must do so in the order given in the table.

<b>Mandatory Beginning Functions</b>	<b>Must Have One of these Functions</b>	<b>Optional Functions</b>	<b>Must Have One of these Functions</b>	<b>Mandatory Ending Functions</b>
#1 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()	ft_open()	ft_egroup()
#2 ft_bgroup()	ft_close()	ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()
#3 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()
#4 ft_bgroup()	ft_select() ft_create()	ft_rattributes() ft_cattributes()		ft_egroup()
#5 ft_bgroup()		ft_rattributes() ft_cattributes()	ft_deselect() ft_delete()	ft_egroup()

The legal groupings are restricted by the service\_class. Refer to the above table and the following list to determine these restrictions.

- If you select FT\_SC\_TRANSFER, you must use the #1 or #2 groupings.
- If you select FT\_SC\_MANAGEMENT, you must use the #3 grouping.
- If you select FT\_SC\_TRANSFER\_AND\_MGMT, you must use the #1, #2, or #3 groupings.
- If you select FT\_SC\_ACCESS, you can use any of the groupings.

If you select the service\_class FT\_SC\_TRANSFER, FT\_SC\_MANAGEMENT, or FT\_SC\_TRANSFER\_AND\_MGT, you must use FT\_FU\_GROUPING in the functional\_units parameter.



## ft\_bgroup()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_bgroup (connection_id, threshold, return_event_name,
           input_dcb, inout_dcb)
Connection_id          connection_id;
Uint32                 threshold;
Local_event_name      return_event_name;
char                   *input_dcb;
struct Ft_bgroup_out_dcb **inout_dcb;
```

Issue an asynchronous `ft_bgroup()` to start a set of grouped functions.

- You cannot nest an `ft_bgroup()` function within another `ft_bgroup()` function (i.e., you cannot invoke `ft_bgroup()` within another set of grouped functions).
- During `ft_connect()`, you must negotiate the `FT_FU_GROUPING` functional\_units to use `ft_bgroup()`.

## Ft\_bgroup\_out\_dcb

```
struct Ft_bgroup_out_dcb {
    Uint32      size;
    struct Api_rc result;
};
```

## ft\_bgroup() Parameters

<b>ft_bgroup() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
threshold	Mandatory Input	Specifies how many functions that change regimes must be successful for the group to be successful; set to a value between 1 and 5

<b>ft_bgroup()</b> Parameter	<b>Type</b>	<b>Description</b>
return_event_name	Mandatory Input	Uniquely identifies the asynchronous call
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

## ft\_egroup()

```
#include %</opt/ftam/include/map.h>  
#include %</opt/ftam/include/mapftam.h>
```

```
Return_code  
ft_egroup (connection_id, return_event_name, input_dcb,  
          inout_dcb)
```

```
Connection_id          connection_id;  
Local_event_name      return_event_name;  
char                   *input_dcb;  
struct Ft_egroup_out_dcb **inout_dcb;
```

Use `ft_egroup()`, either synchronously or asynchronously, to end a set of grouped functions.

- During `ft_connect()`, you must negotiate the `FT_FU_GROUPING functional_units` to use `ft_egroup()`.
- An `ft_egroup()` is successful only if `ft_bgroup()` succeeds and if the threshold number of requests succeeds.

## Ft\_egroup\_out\_dcb

```
struct Ft_egroup_out_dcb {  
    Uint32      size;  
    struct Api_rc result;  
};
```

## ft\_egroup() Parameters

<b>ft_egroup() parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

Managing and Accessing HP FTAM/9000 Files  
**Grouping LLCS FTAM Functions**



## Transferring HP FTAM/9000 Data

After opening a file, you can send and receive FTAM data. Refer to the “Example Programs” chapter for model programs using the functions described in the chapter.

---

## Chapter Overview

This chapter describes the sequence of transferring data and describes the following functions in the order listed.

<b>Task to Perform</b>	<b>Function Used to Perform the Task</b>	<b>Description</b>
Reading Data	ft_read()	Specify the information to be read
	ft_rdata()	Receive data that was requested in ft_read() and QOS=0
	ft_rdataqos()	Receive data that was requested in ft_read() and QOS>0
Writing Data	ft_write()	Specify the information to write
	ft_sdata()	Send data that was requested in ft_write()
Ending Data Transfer	ft_edata()	End the sending of data
	ft_ettransfer()	End data transfer
	ft_cancel()	Cancel the transferring of data
	ft_rcancel()	Acknowledge the cancel of data being transferred

---

**NOTE** This chapter does not explain the parameter structures. For detailed structure information (parameter settings), refer to the “FTAM Data Structures” chapter.

---

## Sequence of Transferring Data

Sequences to use when transferring data are as follows.

- |                                       |                                       |
|---------------------------------------|---------------------------------------|
| 1. <code>ft_read()</code>             | 1. <code>ft_write()</code>            |
| 2. <code>ft_rdata()</code> (multiple) | 2. <code>ft_sdata()</code> (multiple) |
| 3. <code>ft_ettransfer()</code>       | 3. <code>ft_edata()</code>            |
|                                       | 4. <code>ft_ettransfer()</code>       |
- If you invoke `ft_read()` or `ft_write()`, you must exit the regime by invoking `ft_ettransfer()`.
  - If you invoke `ft_sdata()`, you must end the sending of data by invoking `ft_edata()`.
  - To prematurely terminate a data transfer after issuing `ft_read()` or `ft_write()`, call `ft_cancel()`.
  - You can receive a cancel indication any time; if you receive one, acknowledge it by calling `ft_rcancel()`. If you receive a cancel indication, you automatically exit the Data Transfer regime and move into the File Open regime.



---

## Reading Data

Use `ft_read()` to identify what you want to read. To receive the identified information from the responder, call `ft_rdata()` to tell `ftam_init` the number of data elements you want to receive. The `ftam_init` gathers and returns the information requested.

### `ft_read()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_read (connection_id, fadu_identity, access_context,
         return_event_name, input_dcb, inout_dcb)
Connection_id
struct Ft_fadu_identity          connection_id;
enum Ft_access_context          fadu_identity;
Local_event_name                access_context;
char                            return_event_name;
struct Ft_read_out_dcb          *input_dcb;
                                **inout_dcb;
```

Use `ft_read()` to identify the FADUs you want to read and to request that data be sent. Remember, FADUs may contain structure and data. Since FTAM-1, FTAM-3, and NBS-9 file type files contain only single FADUs, you must read the entire file. You can read individual FADUs within FTAM-2 files. (Refer to the “HP-UX FTAM Overview” chapter for a description of FADUs. Refer to the “FTAM Data Structures” chapter for information on `Ft_fadu_identity`.)

- Invoke `ft_read()` in the File Open regime to move to the Data Transfer regime.
- During `ft_connect()`, you must negotiate the `FT_FU_READ` functional\_units to use `ft_read()`.
- During `ft_create()` and `ft_select()`, you must request `FT_FA_READ` in requested\_access.
- The `ft_read()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.
- To read an FTAM-1, FTAM-3, INTAP-1, or NBS-9 file, set `fadu_identity` of `FT_FIRST`. To read an entire FTAM-2 file, set `fadu_identity` to `FT_BEGIN`. To read a FADU within an FTAM-2 file, set `fadu_identity` to `FT_FIRST` or `FT_NEXT`.

**Reading Data**

- If you specify `FT_UNSTRUCTURED_ALL_DATA_UNITS` in `access_context` for FTAM-2 files, you receive only the data elements, not the node descriptors (structuring information). If you specify `FT_FLAT_ALL_DATA_UNITS` in `access_context` for FTAM-2 files, you receive both data elements and node descriptors.

**Ft\_read\_out\_dcb**

```
struct    Ft_read_out_dcb {
        Uint32      size;
        struct      Api_rc    result;
};
```

**ft\_read() Parameters**

<b>ft_read() Parameter</b>	<b>Type</b>	<b>Description</b>
<code>connection_id</code>	Mandatory Input	Uniquely identifies a specific connection to the filestore
<code>fadu_identity</code>	Mandatory Input	Identifies the FADU to read
<code>access_context</code>	Mandatory Input	Specifies the access context; depends on the document type
<code>return_event_name</code>	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
<code>input_dcb</code>	Optional Input	Contains no information; pass a NULL pointer
<code>inout_dcb-&gt;size</code>	Mandatory input if using <code>inout_dcb</code>	Size (in Octets) of the <code>inout_dcb</code> structure and data
<code>inout_dcb-&gt;result</code>	Output	Pointer to the struct <code>Api_rc</code> containing the outcome of the operation: <code>result_code</code> and <code>vendor_code</code> (HP-UX—specific error)

## ft\_rdata()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_rdata (connection_id, des_requested, return_event_name,
          inout_dcb)
Connection_id          connection_id;
Uint16                 des_requested;
Local_event_name      return_event_name;
struct Ft_rdata_out_dcb **inout_dcb;
```

Use `ft_rdata()` to receive the data you requested on `ft_read()`. Issuing `ft_rdata()` controls the number of data elements you read (i.e., controls the flow of information). You may have to call `ft_rdata()` several times to read the entire contents of a file. The information returns in the `inout_dcb->data_unit` parameter.

Since FTAM-1, FTAM-3, and INTAP-1 files contain only single FADUs, you must read the entire file. You can read individual FADUs within FTAM-2 files. (Refer to the “FTAM Data Structures” chapter for `Ft_data_unit` information.)

- Invoke `ft_rdata()` in the Data Transfer regime after invoking `ft_read()`; you do not move to another regime.
- The `des_requested` parameter specifies the maximum number of data elements you want to receive.
- During `ft_connect()`, you must negotiate the `FT_FU_READ` functional\_units to use `ft_rdata()`.
- If the `inout_dcb->data_unit->structure_id` equals `FT_DATA_END`, you reached the end of the file.
- If the `inout_dcb->data_unit->structure_id` equals `FT_CANCEL_IND`, the responder cancelled the data transfer. Call `ft_rcancel()` to respond to the cancel indication; do not send another `ft_rdata()`.

## Ft\_rdata\_out\_dcb

```
struct Ft_rdata_out_dcb {
    Uint16          size;
    struct Api_rc   result;
    Uint32          des_received;
    struct Ft_data_unit *data_unit;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

### **ft\_rdata() Parameters**

<b>ft_rdata() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
des_requested	Mandatory Input	Maximum number of data elements you want to receive
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb-> des_received	Output	Number of data elements received by this request if it is successful
inout_dcb->data_unit	Output	Contains the returned data
inout_dcb-> action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_rdataqos()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_rdataqos (src_connection_id, dest_connection_id,
             des_requested, return_event_name, inout_dcb)
Connection_id          src_connection_id;
Connection_id          dest_connection_id;
UInt16                 des_requested;
Local_event_name       return_event_name;
struct Ft_rdata_out_dcb **inout_dcb;
```

Use `ft_rdataqos()` to receive the data you requested on `ft_read()`. Issuing `ft_rdataqos()` controls the number of data elements you read (i.e., controls the flow of information). You may have to call `ft_rdataqos()` several times to read the entire contents of a file. The information returns in the `inout_dcb->data_unit` parameter.

Since FTAM-1, FTAM-3, and INTAP-1 files contain only single FADUs, you must read the entire file. You can read individual FADUs within FTAM-2 files. (Refer to the “FTAM Data Structures” chapter for `Ft_data_unit` information.)

- Both `ft_rdataqos()` and `ft_rdata()` perform the same function. However, `ft_rdataqos()` is used if QOS is set to 1 or 2 while `ft_rdata()` is used if QOS is set to 0. Also, in `ft_rdataqos()`, both the source and destination `connection_ids` are passed while in `ft_rdata()`, only one `connection_id` is passed.
- Invoke `ft_rdataqos()` in the Data Transfer regime after invoking `ft_read()`; you do not move to another regime.
- The `des_requested` parameter specifies the maximum number of data elements you want to receive.
- During `ft_connect()`, you must negotiate the `FT_FU_READ` `functional_units` to use `ft_rdataqos()`.
- If the `inout_dcb->data_unit->structure_id` equals `FT_DATA_END`, you reached the end of the file.
- If the `inout_dcb->data_unit->structure_id` equals `FT_CANCEL_IND`, the responder cancelled the data transfer. Call `ft_rcancel()` to respond to the cancel indication; do not send another `ft_rdataqos()`.

Transferring HP FTAM/9000 Data  
**Reading Data**

**Ft\_rdataqos\_out\_dcb**

```
struct Ft_rdataqos_out_dcb {
    Uint16      size;
    struct Api_rc result;
    Uint32      des_received;
    struct Ft_data_unit *data_unit;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

**ft\_rdataqos() Parameters**

<b>ft_rdataqos() Parameter</b>	<b>Type</b>	<b>Description</b>
src_connection_id	Mandatory Input	Uniquely identifies the source connection to the filestore
dest_connection_id	Mandatory Input	Uniquely identifies the destination connection to the filestore
des_requested	Mandatory Input	Maximum number of data elements you want to receive
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->des_received	Output	Number of data elements received by this request if it is successful
inout_dcb->data_unit	Output	Contains the returned data
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

---

## Writing Data

Use `ft_write()` and `ft_sdata()` to write data.

- Use `ft_write()` to request permission to write information to a file and to specify what to write (from the initiator to the responder).
- Use `ft_sdata()` to send data.

### `ft_write()`

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_write (connection_id, fadu_identity, fadu_operation,
          return_event_name, input_dcb, inout_dcb)
Connection_id          connection_id;
struct Ft_fadu_identity fadu_identity;
enum Ft_fadu_operation fadu_operation;
Local_event_name      return_event_name;
char                   *input_dcb;
struct Ft_write_out_dcb **inout_dcb;
```

Use `ft_write()` to write data to a specific filestore by identifying a FADU to which you want to write. You can only write to a file as follows.

#### **FTAM-1, FTAM-3, and INTAP-1**

- To write over the existing FADU, `fadu_identity.fadu_location` must be `FT_FIRST` and `fadu_operation` must be `FT_REPLACE`.
- To write to the end of the FADU, `fadu_identity.fadu_location` must be `FT_FIRST` and `fadu_operation` must be `FT_EXTEND`.

#### **FTAM-2**

You can only write to the end of the file. The `fadu_identity.fadu_location` must be `FT_END` and `fadu_operation` must be `FT_INSERT`.

Transferring HP FTAM/9000 Data

### Writing Data

- Invoke `ft_write()` in the File Open regime to move to the Data Transfer regime.
- During `ft_connect()`, you must negotiate the `FT_FU_WRITE` `functional_units` to use `ft_write()`.
- During `ft_create()` and `ft_select()`, you must set `requested_access` according to the document type.

#### **FTAM-1, FTAM-3, and INTAP-1**

`FT_FA_EXTEND`  
`FT_FA_REPLACE`

#### **FTAM-2**

`FT_FA_INSERT`

- During `ft_open()`, you must set `processing_mode` according to the document type.

#### **FTAM-1 and FTAM-3**

`FT_PM_EXTEND`  
`FT_PM_REPLACE`

#### **FTAM-2**

`FT_PM_INSERT`

- The `ft_write()` `input_dcb` parameter does not contain information; therefore, pass a `NULL` pointer for the `input_dcb` parameter.

### **Ft\_write\_out\_dcb**

```
struct    Ft_write_out_dcb {
    Uint32    size;
    struct    Api_rc    result;
};
```



## ft\_write() Parameters

<b>ft_write() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
fadu_identity	Mandatory Input	Identifies the FADU to which you write
fadu_operation	Mandatory Input	Specifies the action you are taking on a file; depends on the document type
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
input_dcb	Optional Input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

## ft\_sdata()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_sdata (connection_id, data_unit, return_event_name,
          inout_dcb)
Connection_id          connection_id;
struct Ft_data_unit    *data_unit;
Local_event_name       return_event_name;
struct Ft_sdata_out_dcb **inout_dcb;
```

## Writing Data

Use `ft_sdata()` to send data units to a file. You can send a maximum of 14 data units linked together for each `ft_sdata()` function call.

- Invoke `ft_sdata()` in the Data Transfer regime; you do not move to another regime.
- During `ft_connect()`, you must negotiate the `FT_FU_WRITE` `functional_units` to use `ft_sdata()`.
- During `ft_create()` and `ft_select()`, you must set `requested_access` according to the document type.

### FTAM-1, FTAM-3, and INTAP-1

`FT_FA_EXTEND`  
`FT_FA_REPLACE`

### FTAM-2

`FT_FA_INSERT`

- If you call `ft_sdata()` asynchronously and then receive the `FTE008_NO_CON_RESOURCES` error, either call `ft_nwcleared()` to determine when the resources are free or re-issue `ft_sdata()`.
- The total amount of data sent per `ft_sdata()` call is limited to about 7K octets. The function will return `FTE101_BUFF_TOO_BIG` if this threshold is exceeded.

## Ft\_sdata\_out\_dcb

```
struct Ft_sdata_out_dcb {
    Uint32                               size;
    struct Api_rc                         result;
    enum Ft_action_result                 action_result;
    struct Ft_diagnostic                  *diagnostic;
};
```

## **ft\_sdata() Parameters**

<b>ft_sdata() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
data_unit	Mandatory Input	Linked list containing the data units to send; use data_unit to control the flow of data
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## Ending Data Transfer

Use `ft_edata()` and `ft_ettransfer()` to terminate data transfers in an orderly manner. Use `ft_cancel()` to prematurely terminate a data transfer after issuing an `ft_read()` or `ft_write()` request, and use `ft_rcancel()` to respond to a cancel indication.

### `ft_edata()`

```
#include </opt/ftam/include/map.h>
#include </opt/ftam/include/mapftam.h>
Return_code
ft_edata (connection_id, return_event_name, input_dcb,
          inout_dcb)
Connection_id          connection_id;
Local_event_name       return_event_name;
char                   *input_dcb;
struct Ft_edata_out_dcb **inout_dcb;
```

After issuing the necessary number of `ft_sdata()` functions, call `ft_edata()` to signal the responder you are no longer sending data (i.e., you completed sending the data unit).

You must call `ft_edata()` before calling `ft_ettransfer()`; after calling `ft_edata()`, you can no longer call `ft_sdata()` until you call `ft_ettransfer()` followed by `ft_write()`.

- Invoke `ft_edata()` in the Data Transfer regime; you do not move to another regime.

### `Ft_edata_in_dcb`

```
struct Ft_edata_in_dcb {
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

### `Ft_edata_out_dcb`

```
struct Ft_edata_out_dcb {
    Uint32 size;
    struct Api_rc result;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_edata() Parameters

<b>ft_edata() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
input_dcb->action_result	Mandatory Input	Specifies that all data was sent
input_dcb->diagnostic	Optional Input	ISO-specific error information; HP-UX responders accept, but ignore this value
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

**Ending Data Transfer****ft\_ettransfer()**

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_ettransfer (connection_id, return_event_name, input_dcb,
              inout_dcb)
Connection_id          connection_id;
Local_event_name       return_event_name;
char                   *input_dcb;
struct Ft_ettransfer_out_dcb **inout_dcb;
```

Use `ft_ettransfer()` to end the transferring of data (`ft_read()` or `ft_write()`).  
Use `ft_ettransfer()` in one of the following sequences.

```
ft_read() ft_rdata()          ft_write() ft_sdata() ft_edata()
ft_ettransfer()              ft_ettransfer()
```

- Invoking `ft_ettransfer()` in the Data Transfer regime moves you back to the File Open regime.
- The `ft_ettransfer()` `input_dcb` parameter does not contain information; therefore, pass a NULL pointer for the `input_dcb` parameter.

**Ft\_ettransfer\_out\_dcb**

```
struct Ft_ettransfer_out_dcb {
    Uint32          size;
    struct Api_rc   result;
    enum Ft_action_result action_result;
    struct Ft_diagnostic *diagnostic;
};
```

## ft\_ettransfer() Parameters

<b>ft_ettransfer() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
input_dcb	Optional Input	Contains no information; pass a NULL pointer
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

## ft\_cancel()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_cancel (connection_id, return_event_name, input_dcb,
           inout_dcb)
Connection_id           connection_id;
Local_event_name       return_event_name;
struct Ft_cancel_in_dcb *input_dcb;
struct Ft_cancel_out_dcb **inout_dcb;
```

**Ending Data Transfer**

Use `ft_cancel()` to prematurely terminate the transferring of data. Using `ft_cancel()` does not abort the entire connection (as does `ft_abort()`). An example of using `ft_cancel()` could be when you are reading data from one FTAM file and writing to another, and you encounter an error when writing the data. You may want to cancel reading data from the FTAM file.

Call `ft_cancel()` after `ft_read()` or `ft_write()`, rather than calling `ft_etransfer()`.

- If you are reading a file, you will not lose data. If you are writing to a file, you may or may not lose the data you wrote.
- Invoking `ft_cancel()` in the Data Transfer regime moves you back to the File Open regime.

**Ft\_cancel\_in\_dcb**

```
struct Ft_cancel_in_dcb {
    enum Ft_action_result    action_result;
    struct Ft_diagnostic     *diagnostic;
};
```

**Ft\_cancel\_out\_dcb**

```
struct Ft_cancel_out_dcb {
    Uint32                    size;
    struct Api_rc             result;
    enum Ft_action_result     action_result;
    struct Ft_diagnostic     *diagnostic;
};
```



## ft\_cancel() Parameters

<b>ft_cancel() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is 0 (zero)
input_dcb->action_result	Mandatory Input	Indicates failure of the data transfer; must be set to FT_AR_PERMANENT_ERROR
input_dcb->diagnostic	Optional Input	ISO-specific error information; HP-UX responders accept, but ignore, this value
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)
inout_dcb->action_result	Output	Specifies the overall success or failure of the request
inout_dcb->diagnostic	Output	Provides ISO-specific error information

Transferring HP FTAM/9000 Data  
Ending Data Transfer

## ft\_rcancel()

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
Return_code
ft_rcancel (connection_id, return_event_name, input_dcb,
            inout_dcb)
Connection_id          connection_id;
Local_event_name       return_event_name;
struct Ft_rcancel_in_dcb *input_dcb;
struct Ft_rcancel_out_dcb **inout_dcb;
```

If an error occurs during a data transfer, the responder sends you a cancel indication. To acknowledge you received this indication, call `ft_rcancel()`. A cancel indication returns either as an

- `inout_dcb->result.return_code` of `FTE175_FCANCEL_IN_RECEIVED` on `ft_sdata()` or
- as an `inout_dcb->data_unit.structure_id` of `FT_CANCEL_IND` on `ft_rdata()`.

If you do not send `ft_rcancel()`, you receive a protocol error which results in an abort.

- Invoking `ft_rcancel()` in the Data Transfer regime moves you back to the File Open regime.

## Ft\_rcancel\_in\_dcb

```
struct Ft_rcancel_in_dcb {
    enum Ft_action_result  action_result;
    struct Ft_diagnostic   *diagnostic;
};
```

## Ft\_rcancel\_out\_dcb

```
struct Ft_rcancel_out_dcb {
    Uint32      size;
    struct Api_rc result;
};
```

## ft\_rcancel() Parameters

<b>ft_rcancel() Parameter</b>	<b>Type</b>	<b>Description</b>
connection_id	Mandatory Input	Uniquely identifies a specific connection to the filestore
return_event_name	Mandatory Input	If the call is asynchronous, uniquely identifies the function call; if the call is synchronous, the value is zero
input_dcb-> action_result	Mandatory Input	Indicates failure of the data transfer; set to the action_result received from the cancel indication
input_dcb->diagnostic	Optional Input	ISO-specific error information; set to the diagnostic received from the cancel indication
inout_dcb->size	Mandatory input if using inout_dcb	Size (in Octets) of the inout_dcb structure and data
inout_dcb->result	Output	Pointer to the struct Api_rc containing the outcome of the operation: result_code and vendor_code (HP-UX—specific error)

Transferring HP FTAM/9000 Data  
**Ending Data Transfer**



## Handling Errors

- Synchronous and Asynchronous calls
- Context Free and Context Sensitive calls
- High and Low level services (HLS and LLS)

You should also understand the various failure types and the differences in errors occurring on synchronous versus asynchronous calls. This chapter covers both of these topics.

## Chapter Overview

This chapter first describes general troubleshooting procedures for FTAM. The remaining sections explain the actions listed in these general guidelines. The specific topics are as follows.

- General Approach to Troubleshooting FTAM
- Interpreting Errors
  - Function Return Values
  - Output Errors
  - MAP 3.0 FTAM Errors
  - HP-UX-specific Errors
  - FTAM Protocol Machine (FPM) and Virtual Filestore (VFS) Errors
    - action\_result
    - state\_result
    - diagnostic
- Example Program Checking for Errors
- Logging Errors
- Using API Tracing

---

**NOTE**

Refer to the *HP FTAM/9000 Reference Manual* for lists of errors, their probable causes, and possible recovery actions.

---

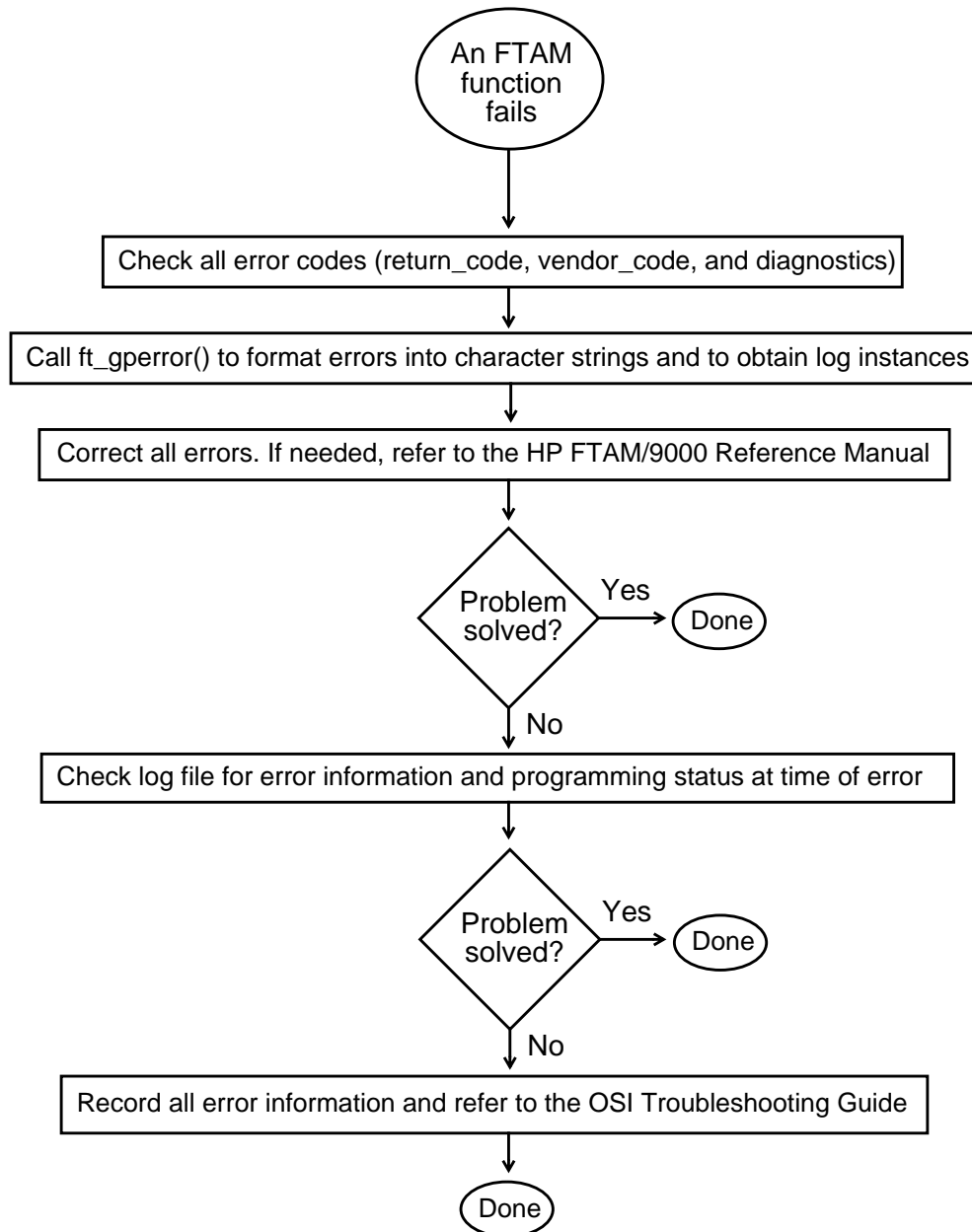
## General Approach to Troubleshooting FTAM

To write a supportable program, you must check for errors. If you receive an error, use the following guidelines and Figure 9-1 to help you resolve the problem.

1. Check all error codes (return\_codes, vendor\_codes, diagnostics). The “Interpreting Errors” section describes the values you interpret.
2. Use ft\_gperror() to format errors into printable character strings and to obtain log instances.
3. Correct the errors as indicated by these values. If needed, look up the error code in the *HP FTAM/9000 Reference Manual* to determine probable causes and corrective actions.
4. If the error still occurs, check the log file (described in the “Logging Errors” section) for error information, such as error codes, program status, and log instance. If you do not have a log instance, find the logged error by searching on the error string in the log file and by searching on the relative time when the error occurred.
5. If the error still occurs, record the following error information and refer to the OSI Planning and Troubleshooting Guide for corrective actions.
  - Specific function call that failed
  - Input parameters passed to the function
  - Function return value
  - inout\_dcb->result.return\_code
  - inout\_dcb->result.vendor\_code
  - inout\_dcb->diagnostic
  - return\_string returned by ft\_gperror()
  - vendor\_string returned by ft\_gperror()
  - Log instance (upper 16 bits of the vendor\_code)



**Figure 9-1**      **General Approach to Troubleshooting FTAM**



## Interpreting Errors

Your program detects errors in two locations.

- Function Return Values (includes Event Management (EM) values)
- Output parameters

### Function Return Values

Function return values indicate whether an error occurred.

- Function return values return in the program as integers.
- These values may occur because of errors on previous calls. For example, if the first call within a group fails, thereafter every function within the group fails.
- The value zero indicates SUCCESS.

Refer to the *HP FTAM/9000 Reference Manual* for a list of function return values.

- |                   |   |
|-------------------|---|
| Synchronous Calls | <ul style="list-style-type: none"><li>• If the value is SUCCESS, the request successfully passed to the application interface, ftam_init, and responder, and returned back to the user program.</li><li>• If the value indicates an error, the application interface, ftam_init, or responder detected the error.</li></ul> |
|-------------------|---|

- |                    |  |
|--------------------|--|
| Asynchronous Calls | <ul style="list-style-type: none"><li>• If the value is SUCCESS, the request successfully passed from the application interface to ftam_init.</li><li>• If the value indicates an error, the application interface detected the error.</li></ul> |
|--------------------|--|

## Output Errors

Information returned in the `inout_dcb` fields may be defined by MAP 3.0 FTAM, HP-UX FTAM, FTAM Protocol Machine (FPM), or the VFS.

<b>This Error</b>	<b>Returns As</b>
MAP 3.0 FTAM error	<code>result.return_code</code>
HP-UX FTAM specific error	<code>result.vendor_code</code>
FPM or VFS error	<code>action_result</code> <code>state_result</code> <code>diagnostic</code>

### MAP 3.0 FTAM Errors

Errors detected by the interface, an `ftam_init`, or an FTAM responder are mapped to MAP 3.0 FTAM errors and returned to the user program.

- The `inout_dcb->result.return_code` contains the MAP 3.0 FTAM error.
- To receive a printable character string and a log instance, call `ft_gperror()`.

Refer to the *HP FTAM/9000 Reference Manual* for a list of MAP 3.0 FTAM errors and possible corrective actions.

## Handling Errors

### Interpreting Errors

- |                    |  |
|--------------------|--|
| Synchronous Calls  | <ul style="list-style-type: none"><li>• The MAP 3.0 FTAM <code>result.return_code</code> is always the same as the function return value.</li><li>• If <code>result.return_code</code> is <code>SUCCESS</code>, the request successfully passed to the application interface, <code>ftam_init</code>, and responder, and returned back to the user program.</li></ul>  |
| Asynchronous Calls | <ul style="list-style-type: none"><li>• If <code>result.return_code</code> indicates an error, the application interface, <code>ftam_init</code>, or responder detected the error.</li><li>• The <code>em_wait()</code> function returns a value for a specific event. If an error occurs in <code>ftam_init</code> or the responder, the function return value of <code>em_wait()</code> indicates <code>SUCCESS</code> and the <code>result.return_code</code> of the original function identifies the error.</li><li>• The <code>inout_dcb</code> pointer and structure are not valid until the <code>em_wait()</code> request returns successfully. If you do not call <code>em_wait()</code>, you will not know if the request completed.</li></ul> |

### HP-UX-Specific Errors

Errors detected by the interface, an `ftam_init`, or an FTAM responder may contain HP-UX FTAM implementation-specific error information that is returned to the user program. The `vendor_code` contains both HP-UX—specific error information (lower 16 bits) and the log instance (upper 16 bits).

- The `inout_dcb->result.vendor_code` contains the vendor errors; however, not all errors returning in `inout_dcb->result` have an accompanying `vendor_code`.
- To receive a printable character string and a log instance, call `ft_gperror()`.
- The `vendor_code` frequently provides additional information when errors return on HLCF functions.

Many `vendor_codes` indicate internal errors that require HP support assistance. Refer first, however, to the *HP FTAM/9000 Reference Manual* for a list of HP-UX—specific errors and possible corrective actions.

## FPM and VFS Errors

FPM errors are generated in the FTAM protocol machine and VFS errors are generated in the Virtual Filestore. Both FPM or VFS errors return an `action_result`, and may optionally return a `state_result` or diagnostic.

**action\_result.** The `action_result` reflects the overall success or failure of the request and returns as either `FT_AR_SUCCESS` or `FT_AR_PERMANENT_ERROR`, respectfully.

```
enum Ft_action_result {  
    FT_AR_SUCCESS=0,  
    FT_AR_TRANSIENT_ERROR=1,  
    FT_AR_PERMANENT_ERROR=2  
};
```

**state\_result.** The `state_result` indicates whether the FPM made the regime change and returns as either `FT_SR_SUCCESS` or `FT_SR_FAILURE`.

```
enum Ft_state_result {  
    FT_SR_SUCCESS=0,  
    FT_SR_FAILURE=1  
};
```

**diagnostic.** The diagnostic reflects the information defined by ISO/IS 8571. The diagnostic may be informational only or it may be error information that elaborates on the `action_result`. (Refer to Figure 9-2.)

- The `diagnostic->error_id` is the error identifier that returns in the `Ft_diagnostic` structure.
- For a character string describing the error, check the `diagnostic->further_details` field for additional information.
- Multiple ISO errors may be represented by a single MAP error; more than one diagnostic may return.
- Multiple FPM errors may occur on one call and are represented by a linked list of `inout_dcb->diagnostic` structures.

Refer to the *HP FTAM/9000 Reference Manual* for a list of `diagnostic->error_ids`.

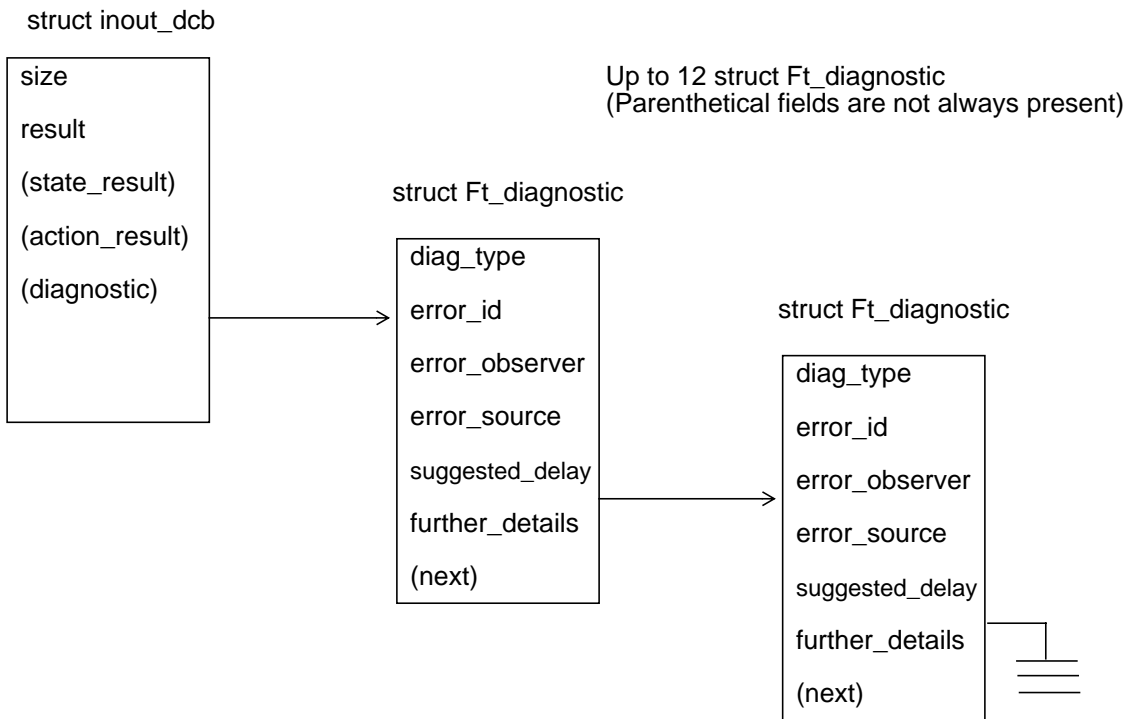
Handling Errors  
**Interpreting Errors**

```

struct Ft_diagnostic {
    struct Ft_diagnostic *next;
    enum Ft_diag_type    diag_type;
    Uint16               error_id;
    enum Ft_entity_ref   error_observer;
    enum Ft_entity_ref   error_source;
    Uint16               suggested_delay;
    char                 *further_details;
};

```

**Figure 9-2** inout\_dcb->diagnostic



diagnostic Field	Description
next	Pointer to another Ft_diagnostic structure or NULL to indicate end of linked list; refer to mapftam.h.
diag_type	A description of the error type; refer to mapftam.h. HP-UX responders do not return the FT_TRANSIENT value.

```
enum FT_diag_type {
    FT_INFORMATIVE      = 0,
    FT_TRANSIENT        = 1,
    FT_PERMANENT        = 2
};
```

**error\_id**            The actual error as defined by ISO/IS 8571. Refer to the *HP FTAM/9000 Reference Manual* for a list of the error\_ids, causes, and corrective actions.

**error\_observer**    The entity that detects the error; refer to mapftam.h.

```
enum FT_entity_ref {
    FT_NOT_CATEGORIZED           = 0,
    FT_INITIATING_FILE_SERVICE_USER = 1,
    FT_INITIATING_FILE_PROTOCOL_MACHINE = 2,
    FT_SERVICE_SUPPORTING_FILE_PROTOCOL_MACHINE = 3,
    FT_RESPONDING_FILE_PROTOCOL_MACHINE = 4,
    FT_RESPONDING_FILE_SERVICE_USER = 5
};
```

**diagnostic  
Field**

**Description**

**error\_source**        Where the error is perceived to have originated.

```
enum Ft_entity_ref {
    FT_NOT_CATEGORIZED           = 0,
    FT_INITIATING_FILE_SERVICE_USER = 1,
    FT_INITIATING_FILE_PROTOCOL_MACHINE = 2,
    FT_SERVICE_SUPPORTING_FILE_PROTOCOL_MACHINE = 3,
    FT_RESPONDING_FILE_PROTOCOL_MACHINE = 4,
    FT_RESPONDING_FILE_SERVICE_USER = 5
};
```

**suggested\_delay**    Recommended delay before calling the function again; useful only for transient errors. HP-UX responders do not return suggested\_delay information.

**further\_details**    Additional information that may help you better understand the diagnostic->error\_id; refer to mapftam.h.

## Example Program Checking for Errors

This example checks the `result.return_code`, `result.vendor_code`, and `Ft_diagnostic` structures. This example also uses `printf()` statements to print the results.

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include %<stdio.h>
/*
**
**          error_handler
**
**  DESCRIPTION:
**    This routine is an example of handling failures experienced
**    during the FTAM functions. This routine will display the
**    error messages.
**
**  PARAMETERS:
**    INPUT:
**      result :          the MAP and HP error information
**      diag   :          the ISO error information
**
**/
void
error_handler(result, diag)
  Api_rc          result;
  struct Ft_diagnostic *diag;
{
  Return_code     res;
  Api_rc          outcome;
  Octet           *return_string;
  Octet           *vendor_string;
  struct Ft_diagnostic *ft_diag = NULL;
  /*
  **   Initialize variables.
  **/
  return_string = NULL;
  vendor_string = NULL;
}
```



```
    /*
    ** Call ft_gperror to get the printable strings for
    ** the return_code and vendor_code. Print the
    ** strings returned from ft_gperror.
    */
    res = ft_gperror(& result, & return_string, & vendor_string, &
outcome);
    if (res == SUCCESS) {
        if (return_string != NULL) {
            (void)printf("FTAM failure: return_code = %s\n", return_string);
            res = ft_fdmemory(return_string, & outcome);
            if (res != SUCCESS)
                error_handler(outcome, ft_diag);
        }
        /*
        ** Print the vendor_code if it exists.
        */
        if (vendor_string != NULL) {
            (void)printf("FTAM failure: vendor_code = %s\n", vendor_string);
            res = ft_fdmemory(vendor_string, & outcome);
            if (res != SUCCESS)
                error_handler(outcome, ft_diag);
        }
    }
    /*
    ** Traverse the diagnostic list and print the further_details
strings.
    */
    while(diag != NULL) {
        (void)printf("FTAM failure: diagnostic number = %d\n",
            diag->error_id);
        if(diag->further_details != NULL)
            (void)printf("FTAM failure: diagnostic = %s\n",
                diag->further_details);
        diag = diag->next;
    }
    exit(1);
}
```

## Logging Errors

All errors return both in your programs and in the log files. When any of the OSI layers have an error, the error propagates up the stack, creating an error at each layer. The number that identifies the error is called a log instance. For example, if an error occurs at the Presentation layer, a related error also occurs at the Application layer. The same log instance occurs at least twice, identifying the related errors at both layers.

If you are unable to resolve errors returned in the user program, refer to the log file for additional error information, such as program status. This information may help you resolve the error or you may use it when calling an HP support representative or referencing the OSI Planning and Troubleshooting Guide.

Log classes include DISASTER, ERROR, WARNING, and INFORMATIVE cases.

---

## Example Program Containing Error Information

Given a `vendor_code`, the following example creates a log filter file (`/tmp/input_file`) that the troubleshooting tool `osidump` uses to check for errors (via log instances in the log file). The file created records a time period (during which the error is assumed to have occurred), the log class to filter on, and the log instance. Following the program is sample output.

```
/* This routine creates a input FORMAT file for use by the "osidump"
 * command. The FORMAT file is created with two time lines, time_from
 * and time_to. A log class of ERROR is used to search on, and if a
 * log instance is present (upper 16 bits of the vendor code), then
 * the instance will also be added to the filter file.
 */
search_log(vcode)
int vcode;
{
FILE *fp, *fopen();
long now, time();
int log_instance;
struct tm *localtime();
struct tm *clock;
extern long timezone;
/* open a /tmp file to store the FORMAT data */
if ((fp = fopen("/tmp/input_file", "w")) == NULL) {
    fprintf(stderr, "fopen error\n");
    exit(1);
}
/* get current time */
now = time ((long *) 0);
clock = localtime(& now);
/* get the upper 16 bits of the vendor_code */
log_instance = vcode >> 16;
/* write log filter file */
fprintf(fp, "FORMATTER\tfilter\ttime_from\t");
if (clock->tm_min > 2)
    fprintf(fp, "%d:%02d:%02d\t", clock->tm_hour,
        clock->tm_min - 3, clock->tm_sec);
else
    fprintf(fp, "%d:%02d:%02d\t", (clock->tm_hour + 23)%24,
        (clock->tm_min + 57)%60, clock->tm_sec);
}
```

## Handling Errors

### Example Program Containing Error Information

---

```
fprintf(fp, "%d/%d/%d\n", clock->tm_mon+1,
        clock->tm_mday, clock->tm_year);
fprintf(fp, "FORMATTER\tfilter\ttime_to \t");
fprintf(fp, "%d:%02d:%02d\t", clock->tm_hour,
        clock->tm_min, clock->tm_sec);
fprintf(fp, "%d/%d/%d\n", clock->tm_mon+1,
        clock->tm_mday, clock->tm_year);
/* set filter "class" to ERROR */
fprintf(fp, "FORMATTER\tfilter\tclass          \tERROR\n");
if (log_instance) /* log instance present */
    fprintf(fp, "FORMATTER\tfilter\tlog_instance\t%d\n",
           log_instance);
fclose(fp);
/* Run osidump with the input FORMAT file just created */
system("osidump -c /tmp/input_file > /tmp/log_file");
}
```

The following text is an example of what you might find in `/tmp/input_file`.

```
FORMATTER      filter  time_from      14:02:16      4/13/89
FORMATTER      filter  time_to        14:05:16      4/13/89
FORMATTER      filter  class          ERROR
FORMATTER      filter  log_instance   9284
```

---

## Using API Tracing

API tracing allows you to get detailed information about your program's interactions with the HP FTAM API without your having to access your source code. It provides the return value of each API call made by your program as well as the input parameters to each call.

### API Tracing Variables

HP FTAM API tracing is controlled by the following three global variables.

- `ft_trace` indicates the trace level to be used. The default value of `ft_trace` is 0, which indicates no tracing, but the value may instead be defined in the `%<api_trace.h>` file as either of the following:
  - `API_TR_ENTRY_EXIT` — procedure entry and exit are traced. No parameter information is displayed. This is useful if you are interested only in seeing what FTAM calls your program is making. This value is defined as 0x1.
  - `API_TR_INPUT` — input parameters to the FTAM functions, as well as procedure entry and exit, are traced. This is useful if you want to verify that FTAM is actually receiving the values you expect. This value is defined as 0x2.
- `ft_trace_fp` is a file pointer that is used to write the trace records. The default is `stderr`. You can redirect the trace information to be saved to a specific file, and then use `fopen` to open the file.
- `ft_trace_max_udata` indicates the maximum amount of user data (in bytes) to be displayed during tracing. The default is 16.

### Enabling API Tracing

The following steps show how to enable API tracing in your program.

1. To include the appropriate definitions, add lines similar to the following to your program:

```
#include %<api_trace.h>
extern int ft_trace;
extern FILE *ft_trace_fp;
extern int ft_trace_max_udata
```

## Handling Errors

### Using API Tracing

2. Within your program, enable tracing by changing the value of the `ft_trace` variable from 0 to either `API_TR_ENTRY_EXIT` or `API_TR_INPUT`. (The “Resolving FTAM Problems” chapter in the *HP FTAM/9000 User's Guide* provides information on enabling and disabling API tracing during an interactive FTAM session.)
3. If you want the trace output to be written to a file, redirect the trace output from `stderr` as follows.

```
ft_trace_fp = fopen("/tmp/my_ft_trace", "a");
```

4. If you want more than the first 16 bytes of data to be displayed, increase the value of the `ft_trace_max_udata` variable. For example:

```
ft_trace_max_udata = 256
```

5. If you want API tracing to be a runtime option, use the `-t` option shown in the following sample code, and enter `program_name -t` at runtime.

The following example shows how to use the variables.

```
#include %<api_trace.h>
extern int ft_trace;
extern FILE *ft_trace_fp;
extern int ft_trace_max_udata;
..

main (argc,argv)
Int argc;
char **argv;
{
/* To run, enter "program name -t" to enable tracing*/
/* Check if tracing needs to be turned on */

if ((argc ==2) && (strcmp(argv{1}, "-t") == 0))
{
/* Turn tracing on, create trace file */

ft_trace = API_TR_INPUT;
ft_trace_fp = fopen("/tmp/my_trace_file", "a");
ft_trace_max_udata = 256

else
ft_trace = 0 /*To be safe, always initialize it*/

/*Then do whatever you have to do*/
}
```

## Interpreting the Trace File

Every time a routine is entered and exited, the time (in hh:mm:ss format) and the name of the call are shown. First the `->` arrow indicates the beginning of the call, then the input parameters are shown, and then the `%<-` arrow indicates the exit. The number following the `%<-` arrow is

the integer value of the return code for that call. A value of 0 indicates that there were no errors. If an integer other than 0 is returned, refer to the `/opt/ftam/includei/mapftam.h` file to determine which return code corresponds to the integer, and then refer to the “FTAM return\_codes” chapter of the *HP FTAM/9000 Reference Manual* for the cause of the return code and corrective actions you can take.

The following is an example trace file for an HP FTAM session.

```
Tue Sep 1 08:55:14 1992 ->ft_aeactivation()
  my_ae_name = 0x5a718
  return_event_name = 0
  input_dcb->authentication = NULL
  input_dcb->my_ae_title_option = User_object_id_option
  input_dcb->my_ae_title = 0x5a908
Tue Sep 1 08:55:15 1992 %<-ft_aeactivation() = 0
Tue Sep 1 08:55:15 1992 ->ft_connect()
  ae_label = 0x202c0006
  return_event_name = 0
  called_ae_name = 0x5a9bc
  input_dcb->called_presentation_address = 0x0
  input_dcb->called_ae_title_option = User_object_id_option
  input_dcb->called_ae_title = 0x5ac14
  input_dcb->called_ae_invoke_id = 0x80000000
  input_dcb->called_ap_invoke_id = 0x80000000
  input_dcb->number_of_retry = 0
  input_dcb->delay_between_retry = 0
  input_dcb->connection_resource_wait_timer = 0
  input_dcb->context_name = 1 0 8571 1 1
  input_dcb->connect_in_info.service_class:
    FT_SC_TRANSFER
    FT_SC_TRANSFER_AND_MGMT
  input_dcb->connect_in_info.functional_units:
    FT_FU_READ
    FT_FU_WRITE
  input_dcb->connect_in_info.attribute_groups
    FT_AG_STORAGE
    FT_AG_PRIVATE
  input_dcb->connect_in_info.quality_of_service = FT_NO_RECOVERY
  input_dcb->connect_in_info.contents_type_list: 0x5ac74
  input_dcb->connect_in_info.contents_type_list.contents_type.connects_form =
    FT_DOCUMENT_TYPE
  contents_type->contents_info.document.name.length = 5
  contents_type->contents_info.document.name.element: 1 0 8571 5 3
  input_dcb->connect_in_info.initiator_identity = "root"
  input_dcb->connect_in_info.account = "account"
  input_dcb->connect_in_info.file_store_pw = "ftam"
Tue Sep 1 08:55:17 1992 %<-ft_connect() = 0
```

## Handling Errors Using API Tracing

---

```
Tue Sep 1 08:55:18 1992 ->ft_open()
  connection_id = 65535
  processing_mode:
    FT_PM_READ
    FT_PM_REPLACE
    FT_PM_EXTEND
    FT_PM_ERASE
  contents_type->contents_info.document.name.length = 5
  contents_type->contents_info.document.name.element: 1 0 8571 5 3
  return_event_name = 0
  input_dcb->concurrency_control = NULL
Tue Sep 1 08:55:18 1992 %-ft_open() = 0
Tue Sep 1 08:55:18 1992 ->ft_read()
  connection_id = 65536
  fadu_identity->fadu_form = FT_FADU_LOCATION
  fadu_identity->fadu_info.fadu_location = FT_FIRST
  access_context = UNSTRUCTURED_ALL_DATA_UNITS
  return_event_name = 0
Tue Sep 1 08:55:18 1992 %-ft_read() = 0
Tue Sep 1 08:55:19 1992 ->ft_close()
  connection_id = 65535
  return_event_name = 2
Tue Sep 1 08:55:19 1992 %-ft_close() = 0
```





## Former Introduction ... need to change

You can learn from the code examples in this chapter, and pattern your applications after them. The examples are in the following order.

Using HLCF Functions Example	This routine synchronously calls the HLCF functions <code>ft_fcopy()</code> , <code>ft_fattributes()</code> , and <code>ft_fdelete()</code> . Additionally, the routine uses <code>ft_dfpcb()</code> to free dynamically allocated data control blocks (DCBs).
Managing FTAM Connections Example	This routine activates <code>ftam_init</code> and synchronously establishes a connection to a responder. The routine then releases the connection and deactivates the <code>ftam_init</code> .
Using LLCS Functions Example	This routine transfers a source file to a destination directory using LLCS functions.
Setting <code>Ae_dir_dn</code> and <code>P_address</code> Utility Example	This routine sets the directory distinguished name and presentation address for the first three examples.
Checking for Errors Example	This routine checks the function return value, <code>result.return_code</code> , <code>result.vendor_code</code> , and diagnostic for the first three examples.
Common Code Example	This routine contains global definitions and common code used by the first three examples.

---

**NOTE**

For an on-line copy of the example that sets the `Ae_dir_name` and `P_address`, refer to `/opt/osi/lib/demos`. For on-line copies of all the other examples, refer to `/opt/ftam/demos`. (See the README files in both directories.)

Additionally, you can refer to the on-line file `/opt/ftam/demos/makefile` for an example Makefile used to compile the example programs.

## Using HLCF Functions Example

This example program synchronously calls the high level context free functions `ft_fcopy()`, `ft_fattributes()`, `ft_fdelete()`. Additionally, the program uses `ft_dfdcb()` to free dynamic memory. Refer to the following three sections for associated routines: “Setting `Ae_dir_name` and `P` address Utility Example,” “Checking for Errors Example,” and “Common Code Example.”

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
main()
/*
**
**                               ftm_hlcopy
**
**  DESCRIPTION:
**    This example program uses high level, context free functions
**    synchronously. This program copies a file, changes the
**    attributes of the destination file and deletes the source file.
**
**/
{
Ae_label                ae_label;
Ae_dir_name             source_dirname;
Ft_filename             source_filename;
Ae_dir_name             destination_dirname;
Ft_filename             destination_filename;
Local_event_name       return_event_name;
struct Ft_fcopy_in_dcb *fco_input_dcb;
struct Ft_fcopy_out_dcb *fco_inout_dcb;
struct Ft_fattributes_in_dcb *fca_input_dcb;
struct Ft_fattributes_out_dcb *fca_inout_dcb;
struct Ft_fdelete_in_dcb *fde_input_dcb;
struct Ft_fdelete_out_dcb *fde_inout_dcb;
Return_code             res;
Api_rc                 outcome;
struct Ft_diagnostic    *diag = NULL;
(void)printf("ftm_fcopy: starting\n");
/*
**    Copy the source file to the destination directory.
**
**    Get the parameters for ft_fcopy.
**    Call ft_fcopy and verify the outcome.
**/
(void)printf("Copying the source file to the destination directory...\n");
fco_parm_in( & source_dirname, & source_filename,
             & destination_dirname, & destination_filename,
             & ae_label, & return_event_name, & fco_input_dcb,
             & fco_inout_dcb );
res = ft_fcopy(source_dirname, source_filename, destination_dirname,
              destination_filename, & ae_label,
              return_event_name, fco_input_dcb, & fco_inout_dcb);
if (res != SUCCESS)
    error_handler(fco_inout_dcb->result, fco_inout_dcb->diagnostic);
```

## Example Programs

### Using HLCF Functions Example

---

```
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)fco_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)fco_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Change the attributes of the destination file.
**   Get the parameters for ft_fcattributes.
**   Call ft_fcattributes and verify the outcome.
*/
(void)printf("Changing attributes of the destination file...\n");
fca_parm_in( & destination_dirname, & destination_filename,
            & ae_label, & return_event_name, & fca_input_dcb,
            & fca_inout_dcb );
res = ft_fcattributes(destination_dirname, destination_filename,
                    & ae_label, return_event_name, fca_input_dcb,
                    & fca_inout_dcb);
if (res != SUCCESS)
    error_handler(fca_inout_dcb->result, fca_inout_dcb->diagnostic);
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)fca_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)fca_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Delete the source file.
**   Get the parameters for ft_fdelete.
**   Call ft_fdelete and verify the outcome.
*/
(void)printf("Deleting the source file...\n");
fde_parm_in( & source_dirname, & source_filename,
            & ae_label, & return_event_name, & fde_input_dcb,
            & fde_inout_dcb );
res = ft_fdelete(source_dirname, source_filename, & ae_label,
                return_event_name, fde_input_dcb, & fde_inout_dcb);
if (res != SUCCESS)
    error_handler(fde_inout_dcb->result, fde_inout_dcb->diagnostic);
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)fde_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)fde_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(void)printf("ftm_hlcopy: finished\n");
return(SUCCESS);
}

```

---

## Managing FTAM Connections Example

This example uses the `ft_aeactivation()`, `ft_connect()`, `ft_rrequest()`, `ft_aedeactivation()`, and `ft_dfdbc()` functions. Refer to the following sections for associated routines: “Setting Ae\_dir\_dn and P\_address Utility Example,” “Checking for Errors Example,” and “Common Code Example.”

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
main()
/*
**                               ftm_connect
**
**  DESCRIPTION:
**    This example program illustrates how to establish
**    a connection with an active "ftam_init" synchronously.
**    This program also releases the connection and deactivates
**    "ftam_init".
**
**
**
*/
{
Ae_label                ae_label;
Connection_id          connection_id;
Ae_dir_name            my_ae_name;
Ae_dir_name            called_ae_name;
Local_event_name       return_event_name;
struct Ft_aeactivate_in_dcb *aea_input_dcb;
struct Ft_connect_in_dcb *con_input_dcb;
struct Ft_relreq_in_dcb *rre_input_dcb;
struct Ft_output       *aea_inout_dcb;
struct Ft_connect_out_dcb *con_inout_dcb;
struct Ft_relreq_out_dcb *rre_inout_dcb;
struct Ft_output       *aed_inout_dcb;
Return_code            res;
Api_rc                outcome;
struct Ft_diagnostic   *diag = NULL;
    (void)printf("ftm_connect: starting\n");
```

## Example Programs

### Managing FTAM Connections Example

---

```
/*
**  Activate "ftam_init".
**
**  Get the parameters for ft_aeactivation.
**  Call ft_aeactivation and verify the outcome.
*/
(void)printf("Activating ftam_init...\n");
aea_parm_in( & my_ae_name, & return_event_name, & aea_input_dcb,
             & aea_inout_dcb, & ae_label );
res = ft_aeactivation(my_ae_name, return_event_name, aea_input_dcb,
                    & aea_inout_dcb, & ae_label);
if (res != SUCCESS)
    error_handler(aea_inout_dcb->result, diag);
/*
**  Free memory.
*/
res = ft_dfdcdb((Octet *)aea_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Establish a connection with the responder.
**
**  Get the parameters for ft_connect.
**  Call ft_connect and verify the outcome.
*/
(void)printf("Establishing a connection with the responder...\n");
con_parm_in( & return_event_name, & called_ae_name, & con_input_dcb,
             & con_inout_dcb, & connection_id );
res = ft_connect(ae_label, return_event_name, called_ae_name,
                con_input_dcb, & con_inout_dcb, & connection_id);
if (res != SUCCESS)
    error_handler(con_inout_dcb->result,
                con_inout_dcb->connect_out_info->diagnostic);
/*
**  Free memory.
*/
res = ft_dfdcdb((Octet *)con_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcdb((Octet *)con_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Release the connection with the responder.
**
**  Get the parameters for ft_rrequest.
**  Call ft_rrequest and verify the outcome.
*/
(void)printf("Releasing the connection with the responder...\n");
rre_parm_in( & return_event_name, & rre_input_dcb, & rre_inout_dcb );
res = ft_rrequest(connection_id, return_event_name, rre_input_dcb,
                 & rre_inout_dcb);
if (res != SUCCESS)
    error_handler(rre_inout_dcb->result, diag);
/*
**  Free memory.
*/
res = ft_dfdcdb((Octet *)rre_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
```

```
/*
**  Deactivate "ftam_init".
**
**  Get the parameters for ft_aedeactivation.
**  Call ft_aedeactivation and verify the outcome.
*/
(void)printf("Deactivating ftam_init...\n");
aed_parm_in( & return_event_name, & aed_inout_dcb );
res = ft_aedeactivation(ae_label, return_event_name, & aed_inout_dcb);
if (res != SUCCESS)
    error_handler(aed_inout_dcb->result, diag);
/*
**  Free memory.
*/
res = ft_dfdc((Octet *)aed_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(void)printf("ftm_connect: finished\n");
return(SUCCESS);
}
```

## Using LLCS Functions Example

This example uses the following functions. Refer to the following sections for associated routines: “Setting Ae\_dir\_dn and P\_address Utility Example,” “Checking for Errors Example,” and “Common Code Example.”

- em\_wait()
- ft\_aeactivation()
- ft\_dfdcb()
- ft\_connect()
- ft\_open()
- ft\_read()
- ft\_write()
- ft\_rdata()
- ft\_sdata()
- ft\_nwcleared()
- ft\_edata()
- ft\_ettransfer()
- ft\_bgroup()
  - ft\_select()
  - ft\_rattributes()
- ft\_egroup()
- ft\_bgroup()
  - ft\_create()
  - ft\_open()
- ft\_egroup()
- ft\_bgroup()
  - ft\_close()
  - ft\_deselect()
- ft\_egroup()



```

#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
main()
/*
**
**          ftm_llcopy
**
** DESCRIPTION:
** This example program illustrates how to transfer a file
** using low level, context sensitive FTAM functions. This
** program transfers the source file to the destination directory.
**
**
*/
{
Ae_dir_name          my_dirname;
Ae_dir_name          dirname;
Ae_label             ae_label;
Local_event_name     return_event_name;
Connection_id        conn_id[TWO_CONNECTIONS];
Ft_filename          src_filename;
Ft_filename          dst_filename;
Ft_processing_mode   processing_mode;
Ft_file_actions      requested_access;
enum Ft_file_status  file_status;
enum Ft_access_context access_context;
enum Ft_fadu_operation fadu_operation;
struct Ft_fadu_identity fadu_identity;
struct Ft_contents_type contents_type;
Uint16               threshold;
Uint16               des_requested;
struct Ft_data_unit  *data_unit;
struct Ft_aeactivate_in_dcb *aea_input_dcb;
struct Ft_output     *aea_inout_dcb;
struct Ft_connect_in_dcb *con_input_dcb;
struct Ft_connect_out_dcb *con_inout_dcb;
struct Ft_relreq_in_dcb *rre_input_dcb;
struct Ft_relreq_out_dcb *rre_inout_dcb;
struct Ft_output     *aed_inout_dcb;
char                 *bgr_input_dcb;
struct Ft_bgroup_out_dcb *bgr_inout_dcb;
char                 *egr_input_dcb;
struct Ft_egrout_out_dcb *egr_inout_dcb;
struct Ft_select_in_dcb *sel_input_dcb;
struct Ft_select_out_dcb *sel_inout_dcb;
struct Ft_create_in_dcb *cre_input_dcb;
struct Ft_create_out_dcb *cre_inout_dcb;
struct Ft_rattributes_in_dcb *rat_input_dcb;
struct Ft_rattributes_out_dcb *rat_inout_dcb;
struct Ft_open_in_dcb *ope_input_dcb;
struct Ft_open_out_dcb *ope_inout_dcb;
char                 *rea_input_dcb;
struct Ft_read_out_dcb *rea_inout_dcb;
char                 *wri_input_dcb;
struct Ft_write_out_dcb *wri_inout_dcb;
struct Ft_rdata_out_dcb *rda_inout_dcb;
struct Ft_sdata_out_dcb *sda_inout_dcb;
struct Ft_edata_in_dcb *eda_input_dcb;
struct Ft_edata_out_dcb *eda_inout_dcb;

```

## Example Programs

### Using LLCS Functions Example

---

```

char                *etr_input_dcb;
struct Ft_ettransfer_out_dcb *etr_inout_dcb;
struct Ft_nwcleared_out_dcb *nwc_inout_dcb;
char                *clo_input_dcb;
struct Ft_close_out_dcb   *clo_inout_dcb;
char                *des_input_dcb;
struct Ft_deselect_out_dcb *des_inout_dcb;
Em_t                time;
Api_rc              result;
struct Ft_attributes    attr_src;
Return_code         res;
Api_rc              outcome;
int                 i, j, index;
Bool                done;
struct Ft_diagnostic    *diag = NULL;
(void)printf("ftm_llcopy: starting\n");
/*
**  Activate "ftam_init".
**
**  Get the parameters for ft_aeactivation.
**  Call ft_aeactivation and verify the outcome.
*/
(void)printf("Activating ftam_init...\n");
aea_parm_in( & my_dirname, & return_event_name, & aea_input_dcb,
             & aea_inout_dcb, & ae_label );
res = ft_aeactivation(my_dirname, return_event_name, aea_input_dcb,
                    & aea_inout_dcb, & ae_label);
if (res != SUCCESS)
    error_handler(aea_inout_dcb->result, diag);
/*
**  Free memory.
*/
res = ft_dfdcdb((Octet *)aea_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Establish connections between ftam_init and the responder
**  for the source file and between ftam_init and the responder
**  for the destination file.
**
**  Get the parameters for ft_connect.
**  Call ft_connect and verify the outcome.
*/
(void)printf("Establishing connections for the source");
(void)printf(" & destination files...\n");
con_source_parm_in( & return_event_name, & dirname, & con_input_dcb,
                  & con_inout_dcb, & conn_id[0] );
res = ft_connect(ae_label, return_event_name, dirname,
                con_input_dcb, & con_inout_dcb, & conn_id[0]);
if (res != SUCCESS)
    error_handler(con_inout_dcb->result,
                con_inout_dcb->connect_out_info->diagnostic);

```

---

```

/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)con_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)con_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
con_parm_in( & return_event_name, & dirname, & con_input_dcb,
            & con_inout_dcb, & conn_id[1] );
res = ft_connect(ae_label, return_event_name, dirname,
                con_input_dcb, & con_inout_dcb, & conn_id[1]);
if (res != SUCCESS)
    error_handler(con_inout_dcb->result,
                con_inout_dcb->connect_out_info->diagnostic);

/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)con_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)con_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);

/*
**   Perform a grouped activity on the source file.
**   ft_bgroup
**   ft_select
**   ft_rattributes
**   ft_egroup
**
**   Get the parameters for ft_bgroup.
**   Call ft_bgroup and verify the outcome.
*/
(void)printf("Selecting and reading attributes of the source file...\n");
bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
            & bgr_inout_dcb );
res = ft_bgroup(conn_id[Src], threshold, return_event_name,
                bgr_input_dcb, & bgr_inout_dcb);
if (res != SUCCESS)
    error_handler(bgr_inout_dcb->result, diag);

/*
**   Get the parameters for ft_select.
**   Call ft_select and verify the outcome.
*/
sel_parm_in( & src_filename, & requested_access,
            & return_event_name, & sel_input_dcb, & sel_inout_dcb );
res = ft_select(conn_id[Src], src_filename, requested_access,
                return_event_name, sel_input_dcb, & sel_inout_dcb);
if (res != SUCCESS)
    error_handler(sel_inout_dcb->result, sel_inout_dcb->diagnostic);

/*
**   Get the parameters for ft_rattributes.
**   Call ft_rattributes and verify the outcome.
*/
rat_parm_in( & return_event_name, & rat_input_dcb, & rat_inout_dcb );
res = ft_rattributes(conn_id[Src], return_event_name, rat_input_dcb,
                    & rat_inout_dcb);
if (res != SUCCESS)
    error_handler(rat_inout_dcb->result, rat_inout_dcb->diagnostic);

```

## Example Programs

### Using LLCS Functions Example

---

```
/*
**  Get the parameters for ft_egroup.
**  Call ft_egroup and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egroup(conn_id[SRC], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb->result, diag);
/*
**  Wait on the asynchronous events in the group.
*/
for (i = 1; i %<M=> 3; ++i) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb->result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb->result, diag);
            break;
        case 2:
            if (sel_inout_dcb->result.return_code != SUCCESS)
                error_handler(sel_inout_dcb->result,
                             sel_inout_dcb->diagnostic);
            break;
        case 3:
            if (rat_inout_dcb->result.return_code != SUCCESS)
                error_handler(rat_inout_dcb->result,
                             rat_inout_dcb->diagnostic);
            break;
        default:
            break;
    }
}
/*
**  Save the attributes of the source file.
*/
attr_src = rat_inout_dcb->attributes;
/*
**  Free memory.
*/
res = ft_dfdbc((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)sel_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)sel_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
```

---

```
/*
**  Open the source file.
**
**  Get the parameters for ft_open.
**  Call ft_open and verify the outcome.
*/
(void)printf("Opening the source file...\n");
ope_parm_in( & processing_mode, & contents_type,
             & return_event_name, & ope_input_dcb, & ope_inout_dcb);
res = ft_open(conn_id[SRC], processing_mode, contents_type,
              return_event_name, ope_input_dcb, & ope_inout_dcb);
if (res != SUCCESS)
    error_handler(ope_inout_dcb->result, ope_inout_dcb->diagnostic);
/*
**  Free memory.
*/
res = ft_dfdcb((Octet *)ope_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)ope_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Perform a grouped activity on the destination file.
**  ft_bgroup
**  ft_create
**  ft_open
**  ft_egroup
**
**  Get the parameters for ft_bgroup.
**  Call ft_bgroup and verify the outcome.
*/
(void)printf("Creating and opening the destination file...\n");
bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
             & bgr_inout_dcb );
res = ft_bgroup(conn_id[DST], threshold, return_event_name,
                bgr_input_dcb, & bgr_inout_dcb);
if (res != SUCCESS)
    error_handler(bgr_inout_dcb->result, diag);
/*
**  Get the parameters for ft_create.
**  Call ft_create and verify the outcome.
*/
cre_parm_in( & dst_filename, & contents_type,
             & requested_access, & file_status, & return_event_name,
             & cre_input_dcb, & cre_inout_dcb );
/*
**  Set the attributes and contents_type of the destination file
**  to be identical with those of the source file.
*/
cre_input_dcb->attributes = attr_src;
contents_type = attr_src.values.contents_type;
res = ft_create(conn_id[DST], dst_filename, contents_type,
                requested_access, file_status, return_event_name,
                cre_input_dcb, & cre_inout_dcb);
if (res != SUCCESS)
    error_handler(cre_inout_dcb->result, cre_inout_dcb->diagnostic);
```

## Example Programs

### Using LLCS Functions Example

---

```
/*
**  Get the parameters for ft_open.
**  Call ft_open and verify the outcome.
*/
ope_parm_in( & processing_mode, & contents_type,
             & return_event_name, & ope_input_dcb, & ope_inout_dcb );
contents_type = attr_src.values.contents_type;
return_event_name = 3;
res = ft_open(conn_id[DST], processing_mode, contents_type,
             return_event_name, ope_input_dcb, & ope_inout_dcb);
if (res != SUCCESS)
    error_handler(ope_inout_dcb->result, ope_inout_dcb->diagnostic);
/*
**  Get the parameters for ft_egrout.
**  Call ft_egrout and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egrout(conn_id[DST], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb->result, diag);
/*
**  Wait on the asynchronous events in the group.
*/
for (i = 1; i %M=> 3; ++i) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb->result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb->result, diag);
            break;
        case 2:
            if (cre_inout_dcb->result.return_code != SUCCESS)
                error_handler(cre_inout_dcb->result,
                             cre_inout_dcb->diagnostic);
            break;
        case 3:
            if (ope_inout_dcb->result.return_code != SUCCESS)
                error_handler(ope_inout_dcb->result,
                             ope_inout_dcb->diagnostic);
            break;
        default:
            break;
    }
}
```

---

```

/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)rat_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)rat_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)cre_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)cre_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)ope_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)ope_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Receive data from the source file.
**
**   Get the parameters for ft_read.
**   Call ft_read and verify the outcome.
*/
(void)printf("Reading data from the source file");
(void)printf("and writing data to the destination file...\n");
rea_parm_in( & fadu_identity, & access_context, & return_event_name,
            & rea_input_dcb, & rea_inout_dcb );
res = ft_read(conn_id[SRC], fadu_identity, access_context,
            return_event_name, rea_input_dcb, & rea_inout_dcb);
if (res != SUCCESS)
    error_handler(rea_inout_dcb->result, diag);
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)rea_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Write to the destination file.
**
**   Get the parameters for ft_write.
**   Call ft_write and verify the outcome.
*/
wri_parm_in( & fadu_identity, & fadu_operation, & return_event_name,
            & wri_input_dcb, & wri_inout_dcb );
res = ft_write(conn_id[DST], fadu_identity, fadu_operation,
            return_event_name, wri_input_dcb, & wri_inout_dcb);
if (res != SUCCESS)
    error_handler(wri_inout_dcb->result, diag);

```

## Example Programs

### Using LLCS Functions Example

---

```
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)wri_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Replace the contents of destination file with the contents
**   of source file.
*/
done = FALSE;
while ( ! done ) {
    /*
    **   Read from the source file.
    **   Get the parameters for ft_rdata.
    **   Call ft_rdata and verify the outcome.
    */
    rda_parm_in(& des_requested, & return_event_name, & rda_inout_dcb);
    res = ft_rdata(conn_id[SRC], des_requested, return_event_name,
                  & rda_inout_dcb);
    if (res != SUCCESS)
        error_handler(rda_inout_dcb->result,
                      rda_inout_dcb->diagnostic);
    /*
    **   If a cancel indication returned, stop
    **   transferring data.
    */
    data_unit = rda_inout_dcb->data_unit;
    for (index = 1; index rda_inout_dcb->des_received; index++)
        data_unit = data_unit->next;
    if (data_unit->structure_id == FT_CANCEL_IND) {
        rda_inout_dcb->data_unit = NULL;
        done = TRUE;
    }
    /*
    **   If only a data end indication was returned,
    **   stop transferring data.
    */
    if (rda_inout_dcb->des_received == 1 & &
        rda_inout_dcb->data_unit->structure_id == FT_DATA_END_IND) {
        rda_inout_dcb->data_unit = NULL;
        done = TRUE;
    }
    /*
    **   Move the data_element next pointer to the next to the
    **   last data_element and check for a data end indication
    **   on the last data_element. If one exists, set the next pointer
    **   to NULL so the last data_element is not written
    **   during the ft_sdata call.
    */
    if ( ! done ) {
        data_unit = rda_inout_dcb->data_unit;
        for (index = 2; index rda_inout_dcb->des_received; index++)
            data_unit = data_unit->next;
        if (data_unit->next->structure_id == FT_DATA_END_IND) {
            data_unit->next = NULL;
            done = TRUE;
        }
    }
}
```



---

```

/*
** Send data to the destination file.
**
** Get the parameters for ft_sdata.
** Call ft_sdata and verify the outcome.
*/
if (rda_inout_dcb ->data_unit != NULL) {
sda_parm_in(& return_event_name, & sda_inout_dcb);
res = ft_sdata(conn_id[DST], rda_inout_dcb->data_unit,
              return_event_name, & sda_inout_dcb);
if (res != SUCCESS) {
/*
** If a "No connection resources" error was returned,
** wait until the resources are free.
*/
if (res == FTE008_NO_CON_RESOURCES) {
/*
** Get the parameters for ft_nwcleared.
** Call ft_nwcleared and verify the outcome.
*/
nwc_parm_in( & return_event_name, & nwc_inout_dcb );
res = ft_nwcleared(conn_id[DST], return_event_name,
                  & nwc_inout_dcb);
if (res != SUCCESS)
error_handler(nwc_inout_dcb->result, diag);
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)nwc_inout_dcb, & outcome);
if (res != SUCCESS)
error_handler(outcome, diag);
}
else {
error_handler(sda_inout_dcb->result,
              sda_inout_dcb->diagnostic);
}
}
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)sda_inout_dcb, & outcome);
if (res != SUCCESS)
error_handler(outcome, diag);
}
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)rda_inout_dcb, & outcome);
if (res != SUCCESS)
error_handler(outcome, diag);
}
/*
** Send a data end to the destination file.
** Get the parameters for ft_edata.
** Call ft_edata and verify the outcome.
*/
eda_parm_in(& return_event_name, & eda_input_dcb, & eda_inout_dcb);
res = ft_edata(conn_id[DST], return_event_name, eda_input_dcb,
              & eda_inout_dcb);
if (res != SUCCESS)
error_handler(eda_inout_dcb->result, eda_inout_dcb->diagnostic);

```

## Example Programs

### Using LLCS Functions Example

---

```
/*
**   Free memory.
*/
res = ft_dfdbc((Octet *)eda_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)eda_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   End the transfer of data to the source and destination files.
**
**   Get the parameters for ft_ettransfer.
**   Call ft_ettransfer and verify the outcome.
*/
(void)printf("Ending data transfer for the source file");
(void)printf(" & the destination file...\n");
for (i = 0; i TWO_CONNECTIONS; ++i) {
    etr_parm_in( & return_event_name, & etr_input_dcb, & etr_inout_dcb );
    res = ft_ettransfer(conn_id[i], return_event_name, etr_input_dcb,
        & etr_inout_dcb);
    if (res != SUCCESS)
        error_handler(etr_inout_dcb->result, etr_inout_dcb->diagnostic);
    /*
    **   Free memory.
    */
    res = ft_dfdbc((Octet *)etr_inout_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
}
/*
**   Perform a grouped activity on the source and destination files.
**   ft_bgroup
**       ft_close
**       ft_deselect
**   ft_egroup
**
**   Get the parameters for ft_bgroup.
**   Call ft_bgroup and verify the outcome.
*/
(void)printf("Closing and deselecting the source file");
(void)printf(" & the destination file...\n");
for (i = 0; i TWO_CONNECTIONS; ++i) {
    bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
        & bgr_inout_dcb );
    res = ft_bgroup(conn_id[i], threshold, return_event_name,
        bgr_input_dcb, & bgr_inout_dcb);
    if (res != SUCCESS)
        error_handler(bgr_inout_dcb->result, diag);
    /*
    **   Get the parameters for ft_close.
    **   Call ft_close and verify the outcome.
    */
    clo_parm_in( & return_event_name, & clo_input_dcb, & clo_inout_dcb );
    res = ft_close(conn_id[i], return_event_name, clo_input_dcb,
        & clo_inout_dcb);
    if (res != SUCCESS)
        error_handler(clo_inout_dcb->result, clo_inout_dcb->diagnostic);
}
```

---

```

/*
**  Get the parameters for ft_deselect.
**  Call ft_deselect and verify the outcome.
*/
des_parm_in( & return_event_name, & des_input_dcb, & des_inout_dcb );
res = ft_deselect(conn_id[i], return_event_name, des_input_dcb,
                 & des_inout_dcb);
if (res != SUCCESS)
    error_handler(des_inout_dcb->result, des_inout_dcb->diagnostic);
/*
**  Get the parameters for ft_egroup.
**  Call ft_egroup and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egroup(conn_id[i], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb->result, diag);
/*
**  Wait on the asynchronous events in the group.
*/
for (j = 1; j %<M=> 3; ++j) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb->result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb->result, diag);
            break;
        case 2:
            if (clo_inout_dcb->result.return_code != SUCCESS)
                error_handler(clo_inout_dcb->result,
                             clo_inout_dcb->diagnostic);
            break;
        case 3:
            if (des_inout_dcb->result.return_code != SUCCESS)
                error_handler(des_inout_dcb->result,
                             des_inout_dcb->diagnostic);
            break;
        default:
            break;
    }
}
/*
**  Free memory.
*/
res = ft_dfdbc((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)clo_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)des_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
}

```

## Example Programs

### Using LLCS Functions Example

---

```
/*
** Release the connections between ftam_init and the
** responders for the source file and the destination file.
**
** Get the parameters for ft_rrequest.
** Call ft_rrequest and verify the outcome.
*/
(void)printf("Releasing connections...\n");
for (i = 0; i TWO_CONNECTIONS; ++i) {
    rre_parm_in( & return_event_name, & rre_input_dcb, & rre_inout_dcb );
    res = ft_rrequest(conn_id[i], return_event_name, rre_input_dcb,
        & rre_inout_dcb);
    if (res != SUCCESS)
        error_handler(rre_inout_dcb->result, diag);
    /*
    ** Free memory.
    */
    res = ft_dfddb((Octet *)rre_inout_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
}
/*
** Deactivate ftam_init
**
** Get the parameters for ft_aedeactivation.
** Call ft_aedeactivation and verify the outcome.
*/
(void)printf("Deactivating ftam_init...\n");
aed_parm_in( & return_event_name, & aed_inout_dcb );
res = ft_aedeactivation(ae_label, return_event_name, & aed_inout_dcb);
if (res != SUCCESS)
    error_handler(aed_inout_dcb->result, diag);
/*
** Free memory.
*/
res = ft_dfddb((Octet *)aed_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(void)printf("ftm_llcopy: finished\n");
return(SUCCESS);
}
```

---

## Setting Ae\_dir\_dn and P\_address Utility Example

This routine sets the directory distinguished name and presentation address for the first three examples.

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
#include %<malloc.h>
/*
**
**                                convert_ddn
**
** DESCRIPTION:
**   This routine scans a character string and sets up
**   an Ae_dir_name.
**
** PARAMETERS:
**   Inputs:
**     name:                string identifying the Directory Distinguished
**                          Name
**
**   Outputs:
**     dirname:            address of the pointer to the Dir_dn structure
**
**
** */
void
convert_ddn(dirname, name)
Ae_dir_name      *dirname;
char             *name;
{
  char           *attr;
  char           *value;
  int            i=0, j=0, k=0, index=0;
  Bool          val_flag=0, null_flag=0;
  /*
  **   Determine the number of rdns.
  ** */
  while (name[j] != NULL) {
    if (name[j] == '/')
      ++index;
    ++j;
  }
  /*
  **   Malloc memory.
  ** */
  attr = (char *)malloc(strlen(name) + 1);
  value = (char *)malloc(strlen(name) + 1);
```

## Example Programs

### Setting Ae\_dir\_dn and P\_address Utility Example

---

```
*dirname = (struct Dir_dn *)malloc(sizeof(struct Dir_dn));
(*dirname)->rdn = (struct Dir_rdn *)malloc(index*sizeof(struct Dir_rdn));
/*
** Set the Ae_dir_name.
*/
(*dirname)->n = index;
/*
** Parse the string and set up the Dir_dn structure.
**
** NOTE: index is set to "-1" since the string MUST start with
** a delimiter "/".
*/
j = 0;
index = -1;
while ( !null_flag ) {
    /*
    ** Have we reached the end of the string?
    */
    if (name[j] == NULL)
        null_flag = TRUE;
    /*
    ** "/" is the signal for another Dir_rdn structure.
    */
    if ((name[j] != '/') && ( !null_flag )) {
        /*
        ** Is the attribute or the value of the attribute being parsed?
        */
        if (name[j] == '=') {
            val_flag = TRUE;
        } else if (val_flag) {
            value[i] = name[j];
            ++i;
        } else {
            attr[k] = name[j];
            ++k;
        }
        ++j;
    }
    value[i] = attr[k] = '\0';
    /*
    ** Has the end of the string or the delimiter for
    ** another rdn structure been reached?
    */
    if ((null_flag) || (name[j] == '/')) {
        if ((i != 0) && (k != 0)) {
            (*dirname)->rdn[index].n = 1;
            (*dirname)->rdn[index].avas =
                (struct Dir_ava *)malloc(sizeof(struct Dir_ava));
            (*dirname)->rdn[index].avas->attr_id.length = 1;
            (*dirname)->rdn[index].avas->attr_id.element =
                (Sint32 *)malloc(sizeof(Sint32));
            /*
            ** Set up the attr_value.
            */
            (*dirname)->rdn[index].avas->attr_value.pointer =
                (Octet *)malloc(i*sizeof(Octet));
            memcpy((char *)(*dirname)->rdn[index].avas->attr_value.pointer,
                value,i);
            (*dirname)->rdn[index].avas->attr_value.length = i;
        }
    }
}
```

---

```

/*
** Set up the syntax_id to NULL.    -- For Future Use --
*/
(*dirname)->rdn[index].avas->syntax_id.length = 0;
(*dirname)->rdn[index].avas->syntax_id.element = NULL;
/*
** Which attribute is this? Set the fourth element of the
** attr_id accordingly.
*/
if (strcmp(attr,"C") == SUCCESS) {
    /*
    ** C = Country
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 6;
} else if (strcmp(attr,"L") == SUCCESS) {
    /*
    ** L = Locality
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 7;
} else if (strcmp(attr,"O") == SUCCESS) {
    /*
    ** O = Organization
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 10;
} else if (strcmp(attr,"OU") == SUCCESS) {
    /*
    ** OU = Organizational Unit
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 11;
} else if ((strcmp(attr,"CN") == SUCCESS) ||
           (strcmp(attr,"AE") == SUCCESS) ||
           (strcmp(attr,"AP") == SUCCESS)) {
    /*
    ** CN = Common Name
    ** AE = Application Entity
    ** AP = Application Process
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 3;
} else {
    /*
    ** Unsupported attr_id.
    */
    (*dirname)->rdn[index].avas->attr_id.element[0] = 9999;
}
/*
** Reset counters and flags for the next rdn structure
*/
i = 0;
k = 0;
++j;
++index;
val_flag = 0;
/*
** If there is another delimiter, move to the next
** Dir_rdn structure and the next character in the string.
*/
while (name[j] == '/') {
    ++j;
    ++index;
}
}

```

Example Programs  
**Setting Ae\_dir\_dn and P\_address Utility Example**

---

```

        else {
            /*
            ** Move to the next character in the string and the next
            ** Dir_rdn structure in the Dir_dn structure.
            */
            ++j;
            ++index;
        }
    }
}
/*
**
** convert_paddr
**
** DESCRIPTION:
** This routine scans a character string and sets up
** a P_address.
**
** PARAMETERS:
** Inputs:
** name: string for P_address
**
** Outputs:
** p_addr: pointer to the P_address
**
**
*/
void
convert_paddr(p_addr, name)
struct P_address *p_addr;
char *name;
{
    char *temp;
    int i=0, j=0, index=0;
    Bool null_flag=0;
    /*
    ** Malloc memory.
    */
    temp = (char *)malloc(strlen(name) + 1);
    p_addr->p_selector = (struct Octet_string *)malloc
        (sizeof(struct Octet_string));
    p_addr->s_selector = (struct Octet_string *)malloc
        (sizeof(struct Octet_string));
    p_addr->t_selector = (struct Octet_string *)malloc
        (sizeof(struct Octet_string));
    p_addr->nsaps = (struct Octet_string *)malloc
        (8*sizeof(struct Octet_string));
    /*
    ** How many nsaps?
    */
    while (name[j] != NULL) {
        if (name[j] == '.')
            ++index;
        ++j;
    }
    p_addr->n_nsaps = index - 2;
}

```



---

```

/*
**  Set the p_addr.
*/
j = 0;
index = 0;
while ( !null_flag ) {
    /*
    **  Has the end of the string been reached?
    */
    if (name[j] == NULL)
        null_flag = TRUE;
    /*
    **  Parse the next field until a delimiter or the
    **  end of the string has been reached.
    */
    if ((name[j] != '.') && ( !null_flag )) {
        temp[i] = name[j];
        ++i;
        ++j;
    }
    /*
    **  Another field has been parsed; set up the correct p_addr field.
    */
    if ((null_flag) || (name[j] == '.')) {
        if (i != 0) {
            ++index;
            switch (index) {
                case 1:
                    /*  p_selector
                    */
                    p_addr->p_selector->pointer = (Octet *)malloc
                        (i*sizeof(Octet));
                    char_to_hex(temp, i, (char *)p_addr->p_selector->pointer,
                        & (p_addr->p_selector->length));
                    break;
                case 2:
                    /*  s_selector
                    */
                    p_addr->s_selector->pointer = (Octet *)malloc
                        (i*sizeof(Octet));
                    char_to_hex(temp, i, (char *)p_addr->s_selector->pointer,
                        & (p_addr->s_selector->length));
                    break;
                case 3:
                    /*  t_selector
                    */
                    p_addr->t_selector->pointer = (Octet *)malloc
                        (i*sizeof(Octet));
                    char_to_hex(temp, i, (char *)p_addr->t_selector->pointer,
                        & (p_addr->t_selector->length));
                    break;
                default:
                    /*  nsaps ... could be multiple nsaps.
                    */
                    p_addr->nsaps[index - 4].pointer = (Octet *)malloc
                        (i*sizeof(Octet));
                    char_to_hex(temp, i, (char *)p_addr->nsaps[index -
4].pointer,
                        & (p_addr->nsaps[index - 4].length));
                    break;
            }
        }
    }
}

```

Example Programs  
Setting Ae\_dir\_dn and P\_address Utility Example

---

```
/*
**   Reset the counters and flags for the next field.
*/
i = 0;
++j;
while (name[j] == '.') {
    /*
    **   Move to the next field in the p_addr
    **   and the next character in the string.
    */
    ++index;
    ++j;
}
}
else {
    /*
    **   Move to the next field in the p_addr and the next
    **   character in the string.
    */
    ++j;
    ++index;
}
}
}
}
}
/*
**
**   char_to_hex
**
**   DESCRIPTION:
**   Convert a character string of hex characters into a p_addr
**   string.
**
**   PARAMETERS
**   Inputs:
**   ch_string:          char hex string
**   ch_length:         char string length
**
**   Outputs:
**   padr_string:       P_address byte string
**   padr_length:       P_address string length
**
**   ALGORITHM:
**   Fill the P_address byte string with zeros.
**   For each two hex characters in the character string:
**   Convert the first character to a 4 bit value.
**   Put the value in the 1st 4 bits of the next P_address byte.
**   Convert the second character to a 4 bit value.
**   Put the value in the last 4 bits of the P_address byte.
**   Set the length of the P_address byte string.
**
*/
```

---

```
void
char_to_hex(ch_string, ch_length, padr_string, padr_length)
char          *ch_string;
int           ch_length;
char          *padr_string;
Uint32       *padr_length;
{
    static char hex[] = "0123456789ABCDEF";
    int         padr_index;
    int         ch_index;
    char        ch;
    char        *position;
    if (ch_length % 2 != 0) {
        (void)printf("Character string won't fit into P_address field\n");
        exit(ERROR);
    }
    /*
    **   Fill the p_addr byte with zeros.
    */
    for (padr_index = 0; padr_index <% (ch_length/2); ++padr_index)
        padr_string[padr_index] = 0x00;
    padr_index = 0;
    ch_index = 0;
    while (ch_index <% ch_length) {
        /*
        **   Put the first hex char into first half of byte.
        */
        ch = ch_string[ch_index];
        if (ch == '\0') {
            break;
        }
        position = strchr(hex, ch);
        if (position == NULL) {
            (void)printf("Invalid hex character in P_address\n");
            exit(ERROR);
        }
        padr_string[padr_index] = (position-hex) <%<% 4;
        ++ch_index;
        /*
        **   Put the second hex char into second half of byte.
        */
        ch = ch_string[ch_index];
        if (ch == '\0') {
            break;
        }
        position = strchr(hex, ch);
        if (position == NULL) {
            (void)printf("Invalid hex character in P_address\n");
            exit(ERROR);
        }
        padr_string[padr_index] |= ((position-hex) & 0x0F);
        ++ch_index;
        ++padr_index;
    }
    *padr_length = padr_index;
}
```

## Checking for Errors Example

This example checks the function return value and the `result.return_code`, `result.vendor_code`, and diagnostic structures. This example also uses `printf()` statements to print the results. Use this routine in conjunction with the first three examples in this chapter: [Using HLCF Functions Example](#), [Managing FTAM Connections Example](#), and [Using LLCs Example](#).

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include %<stdio.h>
/*
**          error_handler
**
**  DESCRIPTION:
**    This routine is an example of handling failures experienced
**    during the FTAM functions. This routine will display the
**    error messages.
**
**  PARAMETERS:
**    INPUT:
**      result :          the MAP and HP error information
**      diag  :          the ISO error information
**
**/
void
error_handler(result, diag)
  Api_rc          result;
  struct Ft_diagnostic *diag;
{
  Return_code     res;
  Api_rc          outcome;
  Octet           *return_string;
  Octet           *vendor_string;
  struct Ft_diagnostic *ft_diag = NULL;
  /*
  **    Initialize variables.
  **/
  return_string = NULL;
  vendor_string = NULL;
}
```

```
/*
** Call ft_gpperror to get the printable strings for
** the return_code and vendor_code. Print the
** strings returned from ft_gpperror.
*/
res = ft_gpperror( & result, & return_string, & vendor_string, & outcome
);
if (res == SUCCESS) {
    if (return_string != NULL) {
        (void)printf("FTAM failure: return_code = %s\n", return_string);
        res = ft_fdmemory(return_string, & outcome);
        if (res != SUCCESS)
            error_handler(outcome, ft_diag);
    }
    /*
    ** Print the vendor_code if it exists.
    */
    if (vendor_string != NULL) {
        (void)printf("FTAM failure: vendor_code = %s\n", vendor_string);
        res = ft_fdmemory(vendor_string, & outcome);
        if (res != SUCCESS)
            error_handler(outcome, ft_diag);
    }
}
/*
** Traverse the diagnostic list and print the further_details strings.
*/
while(diag != NULL) {
    (void)printf("FTAM failure: diagnostic number = %d\n",
                diag->error_id);
    if(diag->further_details != NULL)
        (void)printf("FTAM failure: diagnostic = %s\n",
                    diag->further_details);
    diag = diag->next;
}
exit(-1);
}
```

---

## Common Code Example

This example contains global definitions and common code for the first three examples in this chapter: Using HLCF Functions Example, Managing FTAM Connections Example, and Using LLCS Functions Example.

```
/*
**          ftm_globs.h
**
**  DESCRIPTION:
**    This file contains global definitions for the example
**    programs illustrating HLCF calls, connection management,
**    and LLCS calls.
**
**/
#include %<sys/types.h>
/*
**  Constants
**/
#define ERROR                -1
#define TWO_CONNECTIONS     2
#define SRC                  0
#define DST                  1
#define SRC_ID              "src_id"
#define SRC_ID_LEN          6
#define SRC_PW              "src_pw"
#define SRC_PW_LEN         6
#define SRC_FNAME           "/tmp/src_fname"
#define DEST_FNAME          "/tmp/dest_fname"
#define DEST_ID             "dest_id"
#define DEST_ID_LEN        7
#define DEST_PW             "dest_pw"
#define DEST_PW_LEN        7
#define DEL_PW              "del_pw"
#define DEL_PW_LEN         6
#define CREAT_ID            "creator"
#define CREAT_PW            "cre_pw"
#define CREAT_PW_LEN        6
#define LOCAL_INIT          "/C=us/O=company/OU=division/AP=node1/"
#define AE=ftam_init"
#define LOCAL_RESP          "/C=us/O=company/OU=division/AP=node1/"
#define AE=ftam_resp"
#define REMOTE_RESP         "/C=us/O=company/OU=division/AP=node2/"
#define AE=ftam_resp"
#define SRC_ACCOUNT         "account"
#define DEST_ACCOUNT        "account"
#define STORAGE_ACCOUNT     "stor_acct"
#define SERVICE_CLASS       (FT_SC_UNCONSTRAINED | FT_SC_MANAGEMENT \
                             | FT_SC_TRANSFER | FT_SC_TRANSFER_AND_MGMT \
                             | FT_SC_ACCESS)
#define FUNCTIONAL_UNITS   (FT_FU_READ | FT_FU_WRITE \
                             | FT_FU_FILE_ACCESS | FT_FU_LTD_MGMT \
                             | FT_FU_ENH_MGMT | FT_FU_GROUPING)
```

---

```
#define      ATTRIBUTE_GROUPS (FT_AG_STORAGE | FT_AG_SECURITY | FT_AG_PRIVATE)
/** Define functions that return values other than integers.
 */
void exit();
/**
 ** Define the "void" routines in the example routines.
 */
void convert_ddn();
void convert_paddr();
void char_to_hex();
void error_handler();
void passwords_in();
void cat_attribute_in();
void attribute_in();
void ftam_3();
void concur_cntl_in();
void fadu_ident_in();
void wait_parm_in();
void aea_parm_in();
void aed_parm_in();
void con_parm_in();
void rre_parm_in();
void sel_parm_in();
void des_parm_in();
void ope_parm_in();
void clo_parm_in();
void rat_parm_in();
void cre_parm_in();
void bgr_parm_in();
void egr_parm_in();
void nwc_parm_in();
void rea_parm_in();
void wri_parm_in();
void sda_parm_in();
void rda_parm_in();
void eda_parm_in();
void etr_parm_in();
void fca_parm_in();
void fde_parm_in();
void fco_parm_in();
void exit();
size_t strlen();
char *strchr();
void *memcpy();
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
```

## Example Programs

### Common Code Example

---

```
main()
/*
**                                     ftm_hlcopy
**
** DESCRIPTION:
** This example program uses high level, context free functions
** synchronously. This program copies a file, changes the
** attributes of the destination file and deletes the source file.
**
*/
{

Ae_label                ae_label;
Ae_dir_name             source_dirname;
Ft_filename            source_filename;
Ae_dir_name            destination_dirname;
Ft_filename            destination_filename;
Local_event_name       return_event_name;
struct Ft_fcopy_in_dcb *fco_input_dcb;
struct Ft_fcopy_out_dcb *fco_inout_dcb;
struct Ft_fcattributes_in_dcb *fca_input_dcb;
struct Ft_fcattributes_out_dcb *fca_inout_dcb;
struct Ft_fdelete_in_dcb *fde_input_dcb;
struct Ft_fdelete_out_dcb *fde_inout_dcb;
Return_code            res;
Api_rc                outcome;
struct Ft_diagnostic    *diag = NULL;
    (void)printf("ftm_fcopy: starting\n");
    /*
    ** Copy the source file to the destination directory.
    **
    ** Get the parameters for ft_fcopy.
    ** Call ft_fcopy and verify the outcome.
    */
    (void)printf("Copying the source file to the destination directory...\n");
    fco_parm_in( & source_dirname, & source_filename,
                & destination_dirname, & destination_filename,
                & ae_label, & return_event_name, & fco_input_dcb,
                & fco_inout_dcb );
    res = ft_fcopy(source_dirname, source_filename, destination_dirname,
                  destination_filename, & ae_label,
                  return_event_name, fco_input_dcb, & fco_inout_dcb);
    if (res != SUCCESS)
        error_handler(fco_inout_dcb-result, fco_inout_dcb-diagnostic);
    /*
    ** Free memory.
    */
    res = ft_dfdbc((Octet *)fco_input_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
    res = ft_dfdbc((Octet *)fco_inout_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
```



```
/*
**  Change the attributes of the destination file.
**
**  Get the parameters for ft_fcattributes.
**  Call ft_fcattributes and verify the outcome.
*/
(void)printf("Changing attributes of the destination file...\n");
fca_parm_in( & destination_dirname, & destination_filename,
            & ae_label, & return_event_name, & fca_input_dcb,
            & fca_inout_dcb );
res = ft_fcattributes(destination_dirname, destination_filename,
                    & ae_label, return_event_name, fca_input_dcb,
                    & fca_inout_dcb);

if (res != SUCCESS)
    error_handler(fca_inout_dcb-result, fca_inout_dcb-diagnostic);
/*
**  Free memory.
*/
res = ft_dfdcb((Octet *)fca_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)fca_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Delete the source file.
**
**  Get the parameters for ft_fdelete.
**  Call ft_fdelete and verify the outcome.
*/
(void)printf("Deleting the source file...\n");
fde_parm_in( & source_dirname, & source_filename,
            & ae_label, & return_event_name, & fde_input_dcb,
            & fde_inout_dcb );
res = ft_fdelete(source_dirname, source_filename, & ae_label,
                return_event_name, fde_input_dcb, & fde_inout_dcb);
if (res != SUCCESS)
    error_handler(fde_inout_dcb-result, fde_inout_dcb-diagnostic);
/*
**  Free memory.
*/
res = ft_dfdcb((Octet *)fde_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)fde_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(void)printf("ftm_hlcopy: finished\n");
return(SUCCESS);
}
```

## Example Programs

### Common Code Example

```

#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
main()
/*
**                                     ftm_llcopy
** DESCRIPTION:
** This example program illustrates how to transfer a file
** using low level, context sensitive FTAM functions. This
** program transfers the source file to the destination directory.
**
*/
{
Ae_dir_name          my_dirname;
Ae_dir_name          dirname;
Ae_label             ae_label;
Local_event_name     return_event_name;
Connection_id        conn_id[TWO_CONNECTIONS];
Ft_filename          src_filename;
Ft_filename          dst_filename;
Ft_processing_mode   processing_mode;
Ft_file_actions      requested_access;
enum Ft_file_status  file_status;
enum Ft_access_context access_context;
enum Ft_fadu_operation fadu_operation;
struct Ft_fadu_identity fadu_identity;
struct Ft_contents_type contents_type;
Uint16               threshold;
Uint16               des_requested;
struct Ft_data_unit  *data_unit;
struct Ft_aeactivate_in_dcb *aea_input_dcb;
struct Ft_output     *aea_inout_dcb;
struct Ft_connect_in_dcb *con_input_dcb;
struct Ft_connect_out_dcb *con_inout_dcb;
struct Ft_relreq_in_dcb *rre_input_dcb;
struct Ft_relreq_out_dcb *rre_inout_dcb;
struct Ft_output     *aed_inout_dcb;
char                 *bgr_input_dcb;
struct Ft_bgroup_out_dcb *bgr_inout_dcb;
char                 *egr_input_dcb;
struct Ft_egroup_out_dcb *egr_inout_dcb;
struct Ft_select_in_dcb *sel_input_dcb;
struct Ft_select_out_dcb *sel_inout_dcb;
struct Ft_create_in_dcb *cre_input_dcb;
struct Ft_create_out_dcb *cre_inout_dcb;
struct Ft_rattributes_in_dcb *rat_input_dcb;
struct Ft_rattributes_out_dcb *rat_inout_dcb;
struct Ft_open_in_dcb *ope_input_dcb;
struct Ft_open_out_dcb *ope_inout_dcb;
char                 *rea_input_dcb;
struct Ft_read_out_dcb *rea_inout_dcb;
char                 *wri_input_dcb;
struct Ft_write_out_dcb *wri_inout_dcb;
struct Ft_rdata_out_dcb *rda_inout_dcb;
struct Ft_sdata_out_dcb *sda_inout_dcb;
struct Ft_edata_in_dcb *eda_input_dcb;
struct Ft_edata_out_dcb *eda_inout_dcb;
char                 *etr_input_dcb;
struct Ft_ettransfer_out_dcb *etr_inout_dcb;

```

---

```

struct Ft_nwcleared_out_dcb    *nwc_inout_dcb;
char                          *clo_input_dcb;
struct Ft_close_out_dcb      *clo_inout_dcb;
char                          *des_input_dcb;
struct Ft_deselect_out_dcb   *des_inout_dcb;
Em_t                          time;
Api_rc                        result;
struct Ft_attributes         attr_src;
Return_code                   res;
Api_rc                        outcome;
int                            i, j, index;
Bool                          done;
struct Ft_diagnostic         *diag = NULL;
(void)printf("ftm_llcopy: starting\n");
/*
**   Activate "ftam_init".
**
**   Get the parameters for ft_aeactivation.
**   Call ft_aeactivation and verify the outcome.
*/
(void)printf("Activating ftam_init...\n");
aea_parm_in( & my_dirname, & return_event_name, & aea_input_dcb,
             & aea_inout_dcb, & ae_label );
res = ft_aeactivation(my_dirname, return_event_name, aea_input_dcb,
                    & aea_inout_dcb, & ae_label);
if (res != SUCCESS)
    error_handler(aea_inout_dcb-result, diag);
/*
**   Free memory.
*/
res = ft_dfdcdb((Octet *)aea_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Establish connections between ftam_init and the responder
**   for the source file and between ftam_init and the responder
**   for the destination file.
**
**   Get the parameters for ft_connect.
**   Call ft_connect and verify the outcome.
*/
(void)printf("Establishing connections for the source");
(void)printf(" & destination files...\n");
con_source_parm_in( & return_event_name, & dirname, & con_input_dcb,
                  & con_inout_dcb, & conn_id[0] );
res = ft_connect(ae_label, return_event_name, dirname,
                con_input_dcb, & con_inout_dcb, & conn_id[0]);
if (res != SUCCESS)
    error_handler(con_inout_dcb-result,
                con_inout_dcb-connect_out_info-diagnostic);

```

## Example Programs

### Common Code Example

---

```
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)con_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)con_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
con_parm_in( & return_event_name, & dirname, & con_input_dcb,
            & con_inout_dcb, & conn_id[1] );
res = ft_connect(ae_label, return_event_name, dirname,
                con_input_dcb, & con_inout_dcb, & conn_id[1]);
if (res != SUCCESS)
    error_handler(con_inout_dcb-result,
                con_inout_dcb-connect_out_info-diagnostic);

/*
** Free memory.
*/
res = ft_dfdbc((Octet *)con_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)con_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);

/*
** Perform a grouped activity on the source file.
** ft_bgroup
** ft_select
** ft_rattributes
** ft_egroup
** Get the parameters for ft_bgroup.
** Call ft_bgroup and verify the outcome.
*/
(void)printf("Selecting and reading attributes of the source file...\n");
bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
            & bgr_inout_dcb );
res = ft_bgroup(conn_id[SRC], threshold, return_event_name,
                bgr_input_dcb, & bgr_inout_dcb);
if (res != SUCCESS)
    error_handler(bgr_inout_dcb-result, diag);

/*
** Get the parameters for ft_select.
** Call ft_select and verify the outcome.
*/
sel_parm_in( & src_filename, & requested_access,
            & return_event_name, & sel_input_dcb, & sel_inout_dcb );
res = ft_select(conn_id[SRC], src_filename, requested_access,
                return_event_name, sel_input_dcb, & sel_inout_dcb);
if (res != SUCCESS)
    error_handler(sel_inout_dcb-result, sel_inout_dcb-diagnostic);

/*
** Get the parameters for ft_rattributes.
** Call ft_rattributes and verify the outcome.
*/
rat_parm_in( & return_event_name, & rat_input_dcb, & rat_inout_dcb );
res = ft_rattributes(conn_id[SRC], return_event_name, rat_input_dcb,
                    & rat_inout_dcb);
if (res != SUCCESS)
    error_handler(rat_inout_dcb-result, rat_inout_dcb-diagnostic);
```

---

```
/*
** Get the parameters for ft_egroup.
** Call ft_egroup and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egroup(conn_id[SRC], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb-result, diag);
/*
** Wait on the asynchronous events in the group.
*/
for (i = 1; i %<= 3; ++i) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb-result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb-result, diag);
            break;
        case 2:
            if (sel_inout_dcb-result.return_code != SUCCESS)
                error_handler(sel_inout_dcb-result,
                             sel_inout_dcb-diagnostic);
            break;
        case 3:
            if (rat_inout_dcb-result.return_code != SUCCESS)
                error_handler(rat_inout_dcb-result,
                             rat_inout_dcb-diagnostic);
            break;
        default:
            break;
    }
}
/*
** Save the attributes of the source file.
*/
attr_src = rat_inout_dcb-attributes;
/*
** Free memory.
*/
res = ft_dfdcb((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)sel_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)sel_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
```

## Example Programs

### Common Code Example

---

```
/*
**  Open the source file.
**
**  Get the parameters for ft_open.
**  Call ft_open and verify the outcome.
*/
(void)printf("Opening the source file...\n");
ope_parm_in( & processing_mode, & contents_type,
             & return_event_name, & ope_input_dcb, & ope_inout_dcb);
res = ft_open(conn_id[SRC], processing_mode, contents_type,
              return_event_name, ope_input_dcb, & ope_inout_dcb);
if (res != SUCCESS)
    error_handler(ope_inout_dcb-result, ope_inout_dcb-diagnostic);
/*
**  Free memory.
*/
res = ft_dfdcb((Octet *)ope_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdcb((Octet *)ope_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**  Perform a grouped activity on the destination file.
**  ft_bgroup
**  ft_create
**  ft_open
**  ft_egroup
**
**  Get the parameters for ft_bgroup.
**  Call ft_bgroup and verify the outcome.
*/
(void)printf("Creating and opening the destination file...\n");
bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
             & bgr_inout_dcb );
res = ft_bgroup(conn_id[DST], threshold, return_event_name,
                bgr_input_dcb, & bgr_inout_dcb);
if (res != SUCCESS)
    error_handler(bgr_inout_dcb-result, diag);
/*
**  Get the parameters for ft_create.
**  Call ft_create and verify the outcome.
*/
cre_parm_in( & dst_filename, & contents_type,
             & requested_access, & file_status, & return_event_name,
             & cre_input_dcb, & cre_inout_dcb );
/*
**  Set the attributes and contents_type of the destination file
**  to be identical with those of the source file.
*/
cre_input_dcb-attributes = attr_src;
contents_type = attr_src.values.contents_type;
res = ft_create(conn_id[DST], dst_filename, contents_type,
                requested_access, file_status, return_event_name,
                cre_input_dcb, & cre_inout_dcb);
if (res != SUCCESS)
    error_handler(cre_inout_dcb-result, cre_inout_dcb-diagnostic);
```

---

```
/*
**  Get the parameters for ft_open.
**  Call ft_open and verify the outcome.
*/
ope_parm_in( & processing_mode, & contents_type,
             & return_event_name, & ope_input_dcb, & ope_inout_dcb );
contents_type = attr_src.values.contents_type;
return_event_name = 3;
res = ft_open(conn_id[DST], processing_mode, contents_type,
             return_event_name, ope_input_dcb, & ope_inout_dcb);
if (res != SUCCESS)
    error_handler(ope_inout_dcb-result, ope_inout_dcb-diagnostic);
/*
**  Get the parameters for ft_egroup.
**  Call ft_egroup and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egroup(conn_id[DST], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb-result, diag);
/*
**  Wait on the asynchronous events in the group.
*/
for (i = 1; i %<= 3; ++i) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb-result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb-result, diag);
            break;
        case 2:
            if (cre_inout_dcb-result.return_code != SUCCESS)
                error_handler(cre_inout_dcb-result,
                            cre_inout_dcb-diagnostic);
            break;
        case 3:
            if (ope_inout_dcb-result.return_code != SUCCESS)
                error_handler(ope_inout_dcb-result,
                            ope_inout_dcb-diagnostic);
            break;
        default:
            break;
    }
}
```

## Example Programs

### Common Code Example

---

```
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)rat_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)rat_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)cre_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)cre_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)ope_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)ope_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
** Receive data from the source file.
**
** Get the parameters for ft_read.
** Call ft_read and verify the outcome.
*/
(void)printf("Reading data from the source file");
(void)printf("and writing data to the destination file...\n");
rea_parm_in( & fadu_identity, & access_context, & return_event_name,
            & rea_input_dcb, & rea_inout_dcb );
res = ft_read(conn_id[Src], fadu_identity, access_context,
            return_event_name, rea_input_dcb, & rea_inout_dcb);
if (res != SUCCESS)
    error_handler(rea_inout_dcb-result, diag);
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)rea_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
** Write to the destination file.
**
** Get the parameters for ft_write.
** Call ft_write and verify the outcome.
*/
wri_parm_in( & fadu_identity, & fadu_operation, & return_event_name,
            & wri_input_dcb, & wri_inout_dcb );
res = ft_write(conn_id[DST], fadu_identity, fadu_operation,
            return_event_name, wri_input_dcb, & wri_inout_dcb);
if (res != SUCCESS)
    error_handler(wri_inout_dcb-result, diag);
```



---

```

/*
** Free memory.
*/
res = ft_dfdcb((Octet *)wri_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
** Replace the contents of destination file with the contents
** of source file.
*/
done = FALSE;
while ( ! done ) {
    /*
    ** Read from the source file.
    ** Get the parameters for ft_rdata.
    ** Call ft_rdata and verify the outcome.
    */
    rda_parm_in(& des_requested, & return_event_name, & rda_inout_dcb);
    res = ft_rdata(conn_id[SRC], des_requested, return_event_name,
                  & rda_inout_dcb);
    if (res != SUCCESS)
        error_handler(rda_inout_dcb-result,
                    rda_inout_dcb-diagnostic);
    /*
    ** If a cancel indication returned, stop
    ** transferring data.
    */
    data_unit = rda_inout_dcb-data_unit;
    for (index = 1; index %< rda_inout_dcb-des_received; index++)
        data_unit = data_unit-next;
    if (data_unit-structure_id == FT_CANCEL_IND) {
        rda_inout_dcb -data_unit = NULL;
        done = TRUE;
    }
    /*
    ** If only a data end indication was returned,
    ** stop transferring data.
    */
    if (rda_inout_dcb-des_received == 1 & &
        rda_inout_dcb-data_unit-structure_id == FT_DATA_END_IND) {
        rda_inout_dcb -data_unit = NULL;
        done = TRUE;
    }
    /*
    ** Move the data_element next pointer to the next to the
    ** last data_element and check for a data end indication
    ** on the last data_element. If one exists, set the next pointer
    ** to NULL so the last data_element is not written
    ** during the ft_sdata call.
    */
    if ( ! done ) {
        data_unit = rda_inout_dcb-data_unit;
        for (index = 2; index %< rda_inout_dcb-des_received; index ++ )
            data_unit = data_unit-next;
        if (data_unit-next-structure_id == FT_DATA_END_IND) {
            data_unit-next = NULL;
            done = TRUE;
        }
    }
}

```

## Example Programs

### Common Code Example

---

```
/* Send data to the destination file.
**
** Get the parameters for ft_sdata.
** Call ft_sdata and verify the outcome.
*/
if (rda_inout_dcb -data_unit != NULL) {
    sda_parm_in(& return_event_name, & sda_inout_dcb);
    res = ft_sdata(conn_id[DST], rda_inout_dcb-data_unit,
                  return_event_name, & sda_inout_dcb);
    if (res != SUCCESS) {
        /*
        ** If a "No connection resources" error was returned,
        ** wait until the resources are free.
        */
        if (res == FTE008_NO_CON_RESOURCES) {
            /*
            ** Get the parameters for ft_nwcleared.
            ** Call ft_nwcleared and verify the outcome.
            */
            nwc_parm_in( & return_event_name, & nwc_inout_dcb );
            res = ft_nwcleared(conn_id[DST], return_event_name,
                              & nwc_inout_dcb);
            if (res != SUCCESS)
                error_handler(nwc_inout_dcb-result, diag);
            /*
            ** Free memory.
            */
            res = ft_dfdcb((Octet *)nwc_inout_dcb, & outcome);
            if (res != SUCCESS)
                error_handler(outcome, diag);
        }
        else {
            error_handler(sda_inout_dcb-result,
                          sda_inout_dcb-diagnostic);
        }
    }
    /* Free memory.
    */
    res = ft_dfdcb((Octet *)sda_inout_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
}
/*
** Free memory.
*/
res = ft_dfdcb((Octet *)rda_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
}
/*
** Send a data end to the destination file.
**
** Get the parameters for ft_edata.
** Call ft_edata and verify the outcome.
*/
eda_parm_in(& return_event_name, & eda_input_dcb, & eda_inout_dcb);
res = ft_edata(conn_id[DST], return_event_name, eda_input_dcb,
               & eda_inout_dcb);
if (res != SUCCESS)
    error_handler(eda_inout_dcb-result, eda_inout_dcb-diagnostic);
```

---

```
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)eda_input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)eda_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
** End the transfer of data to the source and destination files.
** Get the parameters for ft_ettransfer.
** Call ft_ettransfer and verify the outcome.
*/
(void)printf("Ending data transfer for the source file");
(void)printf(" & the destination file...\n");
for (i = 0; i %< TWO_CONNECTIONS; ++i) {
    etr_parm_in( & return_event_name, & etr_input_dcb, & etr_inout_dcb );
    res = ft_ettransfer(conn_id[i], return_event_name, etr_input_dcb,
        & etr_inout_dcb);
    if (res != SUCCESS)
        error_handler(etr_inout_dcb-result, etr_inout_dcb-diagnostic);
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)etr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
}
/*
** Perform a grouped activity on the source and destination files.
** ft_bgroup
** ft_close
** ft_deselect
** ft_egroup
**
** Get the parameters for ft_bgroup.
** Call ft_bgroup and verify the outcome.
*/
(void)printf("Closing and deselecting the source file");
(void)printf(" & the destination file...\n");
for (i = 0; i %< TWO_CONNECTIONS; ++i) {
    bgr_parm_in( & threshold, & return_event_name, & bgr_input_dcb,
        & bgr_inout_dcb );
    res = ft_bgroup(conn_id[i], threshold, return_event_name,
        bgr_input_dcb, & bgr_inout_dcb);
    if (res != SUCCESS)
        error_handler(bgr_inout_dcb-result, diag);
/*
** Get the parameters for ft_close.
** Call ft_close and verify the outcome.
*/
clo_parm_in( & return_event_name, & clo_input_dcb, & clo_inout_dcb );
res = ft_close(conn_id[i], return_event_name, clo_input_dcb,
    & clo_inout_dcb);
if (res != SUCCESS)
    error_handler(clo_inout_dcb-result, clo_inout_dcb-diagnostic);
```

## Example Programs

### Common Code Example

---

```
/* Get the parameters for ft_deselect.
** Call ft_deselect and verify the outcome.
*/
des_parm_in( & return_event_name, & des_input_dcb, & des_inout_dcb );
res = ft_deselect(conn_id[i], return_event_name, des_input_dcb,
                 & des_inout_dcb);
if (res != SUCCESS)
    error_handler(des_inout_dcb-result, des_inout_dcb-diagnostic);
/*
** Get the parameters for ft_egroup.
** Call ft_egroup and verify the outcome.
*/
egr_parm_in( & return_event_name, & egr_input_dcb, & egr_inout_dcb );
res = ft_egroup(conn_id[i], return_event_name, egr_input_dcb,
               & egr_inout_dcb);
if (res != SUCCESS)
    error_handler(egr_inout_dcb-result, diag);
/*
** Wait on the asynchronous events in the group.
*/
for (j = 1; j %<= 3; ++j) {
    wait_parm_in( & time, & result );
    res = em_wait(time, & return_event_name, & result);
    if (res != SUCCESS)
        error_handler(result, diag);
    switch (return_event_name) {
        case 1:
            if (bgr_inout_dcb-result.return_code != SUCCESS)
                error_handler(bgr_inout_dcb-result, diag);
            break;
        case 2:
            if (clo_inout_dcb-result.return_code != SUCCESS)
                error_handler(clo_inout_dcb-result,
                             clo_inout_dcb-diagnostic);
            break;
        case 3:
            if (des_inout_dcb-result.return_code != SUCCESS)
                error_handler(des_inout_dcb-result,
                             des_inout_dcb-diagnostic);
            break;
        default:
            break;
    }
}
/*
** Free memory.
*/
res = ft_dfdbc((Octet *)bgr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)clo_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)des_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
res = ft_dfdbc((Octet *)egr_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
}
```

---

```
/*
** Release the connections between ftam_init and the
** responders for the source file and the destination file.
**
** Get the parameters for ft_rrequest.
** Call ft_rrequest and verify the outcome.
*/
(void)printf("Releasing connections...\n");
for (i = 0; i %< TWO_CONNECTIONS; ++i) {
    rre_parm_in( & return_event_name, & rre_input_dcb, & rre_inout_dcb );
    res = ft_rrequest(conn_id[i], return_event_name, rre_input_dcb,
                    & rre_inout_dcb);
    if (res != SUCCESS)
        error_handler(rre_inout_dcb-result, diag);
    /*
    ** Free memory.
    */
    res = ft_dfddb((Octet *)rre_inout_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
}
/*
** Deactivate ftam_init
**
** Get the parameters for ft_aedeactivation.
** Call ft_aedeactivation and verify the outcome.
*/
(void)printf("Deactivating ftam_init...\n");
aed_parm_in( & return_event_name, & aed_inout_dcb );
res = ft_aedeactivation(ae_label, return_event_name, & aed_inout_dcb);
if (res != SUCCESS)
    error_handler(aed_inout_dcb-result, diag);
/*
** Free memory.
*/
res = ft_dfddb((Octet *)aed_inout_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(void)printf("ftm_llcopy: finished\n");
return(SUCCESS);
}
```

## Example Programs

### Common Code Example

```
#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include "ftm_globs.h"
#include %<stdio.h>
#include %<malloc.h>
/*
**
**          aea_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_aeactivation
**   program.
**
** PARAMETERS:
**   OUTPUT:
**     my_ae_name :           the ftam_init name of the user
**     return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :           contains authentication field
**     inout_dcb :           contains return_code field
**     ae_label :            AE invocation identifier
**
**
**
*/
void
aea_parm_in(my_ae_name, return_event_name, input_dcb, inout_dcb,
            ae_label)
Ae_dir_name           *my_ae_name;
Local_event_name      *return_event_name;
struct Ft_aeactivate_in_dcb **input_dcb;
struct Ft_output      **inout_dcb;
Ae_label              *ae_label;
{
    /*
    **   Initialize my_ae_name identifying the local ftam_init.
    **
    */
    convert_ddn(my_ae_name, LOCAL_INIT);
    *return_event_name = SYNCHRONOUS;
    /*
    **   Initialize input_dcb.
    **
    */
    *input_dcb = (struct Ft_aeactivate_in_dcb *)malloc
        (sizeof(struct Ft_aeactivate_in_dcb));
    (*input_dcb)-authentication.pointer = (Octet *)NULL;
    (*input_dcb)-authentication.length = 0;
    (*input_dcb)-my_ae_title_option = No_value_option;
    (*input_dcb)-my_ae_title = NULL;
}
```



## Example Programs

### Common Code Example

---

```
Return_code          res;
Api_rc              outcome;
struct Ft_diagnostic *diag = NULL;
/*
** Initialize the parameters with valid values.
*/
*return_event_name = SYNCHRONOUS;
/*
** Initialize the called_ae_name as the DDN identifying the
** responder.
*/
convert_ddn(called_ae_name, LOCAL_RESP);
/*
** Initialize the input_dcb.
*/
res = ft_didcb(FTiConnect, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
** Set up the presentation address.
**
** Note: Only the called_ae_dirname OR the called_presentation_address
**       is needed. Both are set up for example purposes only.
*/
/* convert_paddr( & ((*input_dcb)-called_presentation_address),
                 "0102.0102.0102.490003080009011E20FE00" ); */
/*
** Set up the called_ae_title.
*/
/* 6-5-91 MCK:
*       Changed to supply AP-title.
*/
(*input_dcb)-called_ae_title_option = User_object_id_option;
(*input_dcb)-called_ae_title = (struct Object_id *)malloc
    (sizeof(struct Object_id));
(*input_dcb)-called_ae_title-ae_object_id.length = 5;
(*input_dcb)-called_ae_title-ae_object_id.element = (Sint32 *)malloc
    (5*sizeof(Sint32));
(*input_dcb)-called_ae_title-ae_object_id.element[0] = 1;
(*input_dcb)-called_ae_title-ae_object_id.element[1] = 3;
(*input_dcb)-called_ae_title-ae_object_id.element[2] = 9999;
(*input_dcb)-called_ae_title-ae_object_id.element[3] = 1;
(*input_dcb)-called_ae_title-ae_object_id.element[4] = 7;
/*
** Set up the invoke ids.
*/
(*input_dcb)-called_ae_invoke_id = NO_VALUE;
(*input_dcb)-called_ap_invoke_id = NO_VALUE;
/*
** Set up the retry counters and timer.
*/
(*input_dcb)-number_of_retry = 0;
(*input_dcb)-delay_between_retry = 0;
(*input_dcb)-connection_resource_wait_timer = 0;
```



---

```
/*
** Set up the context_name for FTAM.
*/
(*input_dcb)-context_name.element = (Sint32 *)malloc(5*sizeof(Sint32));
(*input_dcb)-context_name.length = 5;
(*input_dcb)-context_name.element[0] = 1;
(*input_dcb)-context_name.element[1] = 0;
(*input_dcb)-context_name.element[2] = 8571;
(*input_dcb)-context_name.element[3] = 1;
(*input_dcb)-context_name.element[4] = 1;
/*
** Set up the service_class, functional_units, and attribute_groups.
*/
(*input_dcb)-connect_in_info.service_class = SERVICE_CLASS;
(*input_dcb)-connect_in_info.functional_units = FUNCTIONAL_UNITS;
(*input_dcb)-connect_in_info.attribute_groups = ATTRIBUTE_GROUPS;
/*
** Set up the quality of service.
*/
(*input_dcb)-connect_in_info.quality_of_service = FT_NO_RECOVERY;
/*
** Set up the contents_type_list for FTAM-3 document type.
*/
(*input_dcb)-connect_in_info.contents_type_list =
    (struct Ft_contents_type_element *)malloc
    (sizeof(struct Ft_contents_type_element));
(*input_dcb)-connect_in_info.contents_type_list-next_element = NULL;
ftam_3( & ((*input_dcb)-connect_in_info.
    contents_type_list-contents_type) );
/*
** Set up the initiator_id, account, and file_store_pw.
*/
(*input_dcb)-connect_in_info.initiator_identity = DEST_ID;
(*input_dcb)-connect_in_info.file_store_pw.length = DEST_PW_LEN;
(*input_dcb)-connect_in_info.file_store_pw.pointer = (Octet *)DEST_PW;
(*input_dcb)-connect_in_info.account = DEST_ACCOUNT;
/*
** Initialize "inout_dcb".
*/
*inout_dcb = NULL;
*connection_id = 0;
}
```

## Example Programs

### Common Code Example

---

```
void
con_parm_in(return_event_name, called_ae_name, input_dcb, inout_dcb,
            connection_id)
Local_event_name      *return_event_name;
Ae_dir_name           *called_ae_name;
struct Ft_connect_in_dcb **input_dcb;
struct Ft_connect_out_dcb **inout_dcb;
Connection_id         *connection_id;
{
Return_code           res;
Api_rc                outcome;
struct Ft_diagnostic  *diag = NULL;
/*
**   Initialize the parameters with valid values.
*/
*return_event_name = SYNCHRONOUS;
/*
**   Initialize the called_ae_name as the DDN identifying the
**   responder.
*/
convert_ddn(called_ae_name, REMOTE_RESP);
/*
**   Initialize the input_dcb.
*/
res = ft_didcb(FTiConnect, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
/*
**   Set up the presentation address.
**
**   Note: Only the called_ae_dirname OR the called_presentation_address
**         is needed. Both are set up for example purposes only.
*/
/*
convert_paddr( & ((*input_dcb)-called_presentation_address),
              "0102.0102.0102.490003080009011E20FE00" ); */
/*
**   Set up the called_ae_title.
*/
/* 6-5-91 MCK:
*       Changed to supply AP-title.
*/
(*input_dcb)-called_ae_title_option = User_object_id_option;
(*input_dcb)-called_ae_title = (struct Object_id *)malloc
    (sizeof(struct Object_id));
(*input_dcb)-called_ae_title-ae_object_id.length = 5;
(*input_dcb)-called_ae_title-ae_object_id.element = (Sint32 *)malloc
    (5*sizeof(Sint32));
(*input_dcb)-called_ae_title-ae_object_id.element[0] = 1;
(*input_dcb)-called_ae_title-ae_object_id.element[1] = 3;
(*input_dcb)-called_ae_title-ae_object_id.element[2] = 9999;
(*input_dcb)-called_ae_title-ae_object_id.element[3] = 1;
(*input_dcb)-called_ae_title-ae_object_id.element[4] = 7;
```

---

```

/*
**  Set up the invoke ids.
*/
(*input_dcb)-called_ae_invoke_id = NO_VALUE;
(*input_dcb)-called_ap_invoke_id = NO_VALUE;
/*
**  Set up the retry counters and timer.
*/
(*input_dcb)-number_of_retry = 0;
(*input_dcb)-delay_between_retry = 0;
(*input_dcb)-connection_resource_wait_timer = 0;
/*
**  Set up the context_name for FTAM.
*/
(*input_dcb)-context_name.element = (Sint32 *)malloc(5*sizeof(Sint32));
(*input_dcb)-context_name.length = 5;
(*input_dcb)-context_name.element[0] = 1;
(*input_dcb)-context_name.element[1] = 0;
(*input_dcb)-context_name.element[2] = 8571;
(*input_dcb)-context_name.element[3] = 1;
(*input_dcb)-context_name.element[4] = 1;
/*
**  Set up the service_class, functional_units, and attribute_groups.
*/
(*input_dcb)-connect_in_info.service_class = FT_SC_UNCONSTRAINED
      | FT_SC_MANAGEMENT | FT_SC_TRANSFER
      | FT_SC_TRANSFER_AND_MGMT | FT_SC_ACCESS;
(*input_dcb)-connect_in_info.functional_units = FT_FU_READ | FT_FU_WRITE
      | FT_FU_FILE_ACCESS | FT_FU_LTD_MGMT
      | FT_FU_ENH_MGMT | FT_FU_GROUPING;
/* (*input_dcb)-connect_in_info.attribute_groups = FT_AG_STORAGE
      | FT_AG_SECURITY | FT_AG_PRIVATE; */
(*input_dcb)-connect_in_info.attribute_groups = FT_AG_STORAGE
      | FT_AG_PRIVATE;
/*
**  Set up the quality of service.
*/
(*input_dcb)-connect_in_info.quality_of_service = FT_NO_RECOVERY;
/*
**  Set up the contents_type_list for FTAM-3 document type.
*/
(*input_dcb)-connect_in_info.contents_type_list =
    (struct Ft_contents_type_element *)malloc
    (sizeof(struct Ft_contents_type_element));
(*input_dcb)-connect_in_info.contents_type_list-next_element = NULL;
ftam_3( & ((*input_dcb)-connect_in_info.
    contents_type_list-contents_type) );
/*
**  Set up the initiator_id, account, and file_store_pw.
*/
(*input_dcb)-connect_in_info.initiator_identity = DEST_ID;
(*input_dcb)-connect_in_info.file_store_pw.length = DEST_PW_LEN;
(*input_dcb)-connect_in_info.file_store_pw.pointer = (Octet *)DEST_PW;
(*input_dcb)-connect_in_info.account = DEST_ACCOUNT;

```

## Example Programs

### Common Code Example

---

```
    /*
    **   Initialize "inout_dcb".
    */
    *inout_dcb = NULL;
    *connection_id = 0;
}
/*
**
**           rre_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_rrequest parameters.
**
** PARAMETERS:
**   OUTPUT:
**     return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :         input_dcb is NULL
**     inout_dcb :        contains return_code field
**
**
*/
void
rre_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
struct Ft_relreq_in_dcb  **input_dcb;
struct Ft_relreq_out_dcb **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    */
    *return_event_name = SYNCHRONOUS;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**           aed_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_aedeactivation parameters.
**
** PARAMETERS:
**   OUTPUT:
**     return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**     inout_dcb :         contains return_code field
**
**
*/
```

---

```

void
aed_parm_in(return_event_name, inout_dcb)
Local_event_name      *return_event_name;
struct Ft_output      **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    */
    *return_event_name = SYNCHRONOUS;
    *inout_dcb = NULL;
}
/*
**
**           fco_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_fcopy parameters.
**
**
** PARAMETERS:
**   Outputs:
**     source_dirname :           name of the source AE directory
**     source_filename :         name of the source file
**     destination_dirname :     name of the destination AE directory
**     destination_filename :    name of the destination file
**     ae_label :               label of AE invocation
**     return_event_name :      gets signalled when the event completes
**                               default: SYNCHRONOUS
**     input_dcb :              contains ft_fcopy input information
**     inout_dcb :              contains return_code field
**
*/
void
fco_parm_in(source_dirname, source_filename, destination_dirname,
            destination_filename, ae_label, return_event_name, input_dcb,
            inout_dcb)
Ae_dir_name           *source_dirname;
Ft_filename           *source_filename;
Ae_dir_name           *destination_dirname;
Ft_filename           *destination_filename;
Ae_label              *ae_label;
Local_event_name      *return_event_name;
struct Ft_fcopy_in_dcb **input_dcb;
struct Ft_fcopy_out_dcb **inout_dcb;
{
    Return_code        res;
    Api_rc             outcome;
    struct Ft_diagnostic *diag = NULL;
}

```

## Example Programs

### Common Code Example

---

```
/*
** Set the directory distinguished names to identify the
** responders on the source and destination nodes.
*/
convert_ddn(source_dirname, LOCAL_RESP);
convert_ddn(destination_dirname, REMOTE_RESP);
/*
** Initialize the filenames.
*/
*source_filename = SRC_FNAME;
*destination_filename = DEST_FNAME;
*ae_label = NULL;
*return_event_name = SYNCHRONOUS;
/*
** Initialize the input_dcb.
*/
res = ft_didcb(FTiFCopy, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(*input_dcb)-source_init_id = SRC_ID;
(*input_dcb)-source_filestore_pw.pointer = (Octet *)SRC_PW;
(*input_dcb)-source_filestore_pw.length = SRC_PW_LEN;
(*input_dcb)-source_account = SRC_ACCOUNT;
passwords_in( & ((*input_dcb)-source_file_passwords) );
(*input_dcb)-dest_init_id = DEST_ID;
(*input_dcb)-dest_filestore_pw.pointer = (Octet *)DEST_PW;
(*input_dcb)-dest_filestore_pw.length = DEST_PW_LEN;
(*input_dcb)-dest_account = DEST_ACCOUNT;
passwords_in( & ((*input_dcb)-dest_file_passwords) );
(*input_dcb)-create_file_pw.pointer = (Octet *)CREAT_PW;
(*input_dcb)-create_file_pw.length = CREAT_PW_LEN;
(*input_dcb)-overwrite = FT_RECREATE_FILE;
/*
** Initialize the inout_dcb.
*/
*inout_dcb = NULL;
}
```

---

```
/*
**
**          fca_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_fcattributes parameters.
**
**
** PARAMETERS:
**   Outputs:
**     dirname :          name of the AE directory
**     filename :        name of the file
**     ae_label :        label of AE invocation
**     return_event_name : gets signalled when the event completes
**                         default: SYNCHRONOUS
**     input_dcb :        contains ft_fcattributes input information
**     inout_dcb :        contains return_code field
**
**
*/
void
fca_parm_in(dirname, filename, ae_label, return_event_name, input_dcb,
            inout_dcb)
Ae_dir_name          *dirname;
Ft_filename           *filename;
Uint32               *ae_label;
Local_event_name      *return_event_name;
struct Ft_fcattributes_in_dcb **input_dcb;
struct Ft_fcattributes_out_dcb **inout_dcb;
{
Return_code           res;
Api_rc               outcome;
struct Ft_diagnostic  *diag = NULL;
/*
**   Set the directory distinguished name to identify the responder
**   on the desired node.
**
*/
convert_ddn(dirname, REMOTE_RESP);
*filename = DEST_FNAME;
*ae_label = 0;
*return_event_name = SYNCHRONOUS;
/*
**   Initialize the input_dcb.
**
*/
res = ft_didcb(FTiFCattributes, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(*input_dcb)-init_id = DEST_ID;
(*input_dcb)-filestore_pw.pointer = (Octet *)DEST_PW;
(*input_dcb)-filestore_pw.length = DEST_PW_LEN;
passwords_in( & ((*input_dcb)-file_passwords) );
```

## Example Programs

### Common Code Example

---

```
/*
** Set up the changed attributes.
*/
(*input_dcb)-attribute_names |= FT_AN_STORAGE_ACCT
                             | FT_AN_FILE_AVAILABILITY | FT_AN_FUTURE_FILESIZE
                             | FT_AN_LEGAL_QUAL
                             | FT_AN_PRIVATE_USE;
cat_attribute_in( & ((*input_dcb)-attributes) );
(*input_dcb)-account = DEST_ACCOUNT;
/*
** Initialize the inout_dcb.
*/
*inout_dcb = NULL;
}
/*
**
**          fde_parm_in
**
** DESCRIPTION:
** This routine assigns valid values to the ft_fdelete parameters.
**
** PARAMETERS:
** Outputs:
**   dirname :           name of the AE directory
**   filename :         name of the file
**   ae_label :         label of AE invocation
**   return_event_name : gets signalled when the event completes
**                       default: SYNCHRONOUS
**   input_dcb :        contains ft_fdelete input information
**   inout_dcb :        contains return_code field
**
*/
void
fde_parm_in(dirname, filename, ae_label, return_event_name, input_dcb,
            inout_dcb)
Ae_dir_name           *dirname;
Ft_filename           *filename;
Uint32                *ae_label;
Local_event_name      *return_event_name;
struct Ft_fdelete_in_dcb **input_dcb;
struct Ft_fdelete_out_dcb **inout_dcb;
{
Return_code           res;
Api_rc                outcome;
struct Ft_diagnostic  *diag = NULL;
/*
** Set the directory distinguished name to identify the desired
** responder.
*/
convert_ddn(dirname, LOCAL_RESP);
*filename = SRC_FNAME;
*ae_label = 0;
*return_event_name = SYNCHRONOUS;
```



---

```

    /*
    ** Initialize the input_dcb.
    */
    res = ft_didcb(FTiFDelete, 0, (Octet **)input_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
    (*input_dcb)-init_id = SRC_ID;
    (*input_dcb)-filestore_pw.pointer = (Octet *)SRC_PW;
    (*input_dcb)-filestore_pw.length = SRC_PW_LEN;
    (*input_dcb)-delete_file_pw.pointer = (Octet *)DEL_PW;
    (*input_dcb)-delete_file_pw.length = DEL_PW_LEN;
    (*input_dcb)-account = DEST_ACCOUNT;
    /*
    ** Initialize the inout_dcb.
    */
    *inout_dcb = NULL;
}
/*
**
**                               sel_parm_in
**
** DESCRIPTION:
** This routine assigns valid values to the ft_select parameters.
**
** PARAMETERS:
** Outputs:
** filename :           name of the file to select
** requested_access :   requested file action
** return_event_name :  gets signalled when the event completes
**                               default: SYNCHRONOUS
** input_dcb :         contains ft_select input information
** inout_dcb :         contains return_code field
**
**
void
sel_parm_in(filename, requested_access, return_event_name, input_dcb,
            inout_dcb)
Ft_filename           *filename;
Ft_file_actions       *requested_access;
Local_event_name      *return_event_name;
struct Ft_select_in_dcb **input_dcb;
struct Ft_select_out_dcb **inout_dcb;
{

```

## Example Programs

### Common Code Example

```

Api_rc                               outcome;
Return_code                           res;
struct Ft_diagnostic                   *diag = NULL;
/*
**   Initialize the parameters with valid values.
*/
*filename = SRC_FNAME;
*requested_access = FT_FA_READ | FT_FA_REPLACE
                  | FT_FA_EXTEND | FT_FA_ERASE | FT_FA_READ_ATTRIBUTE
                  | FT_FA_CHANGE_ATTRIBUTE | FT_FA_DELETE_FILE;
*return_event_name = 2;
/*
**   Initialize the input_dcb.
*/
res = ft_didcb(FTiSelect, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
attribute_in( & ((*input_dcb)-attributes) );
passwords_in( & ((*input_dcb)-file_passwords) );
(*input_dcb)-account = DEST_ACCOUNT;
(*input_dcb)-concurrency_control = (struct Ft_concurrency_control *)
    malloc(sizeof(struct Ft_concurrency_control));
concur_cntl_in( & ((*input_dcb)-concurrency_control) );
/*
**   Initialize the inout_dcb.
*/
*inout_dcb = NULL;
}

/*
**
**
**
**
**   DESCRIPTION:
**   This routine assigns valid values to the ft_deselect parameters.
**
**
**   PARAMETERS:
**   Outputs:
**   return_event_name :   gets signalled when the event completes
**                         default: SYNCHRONOUS
**   input_dcb :          input_dcb is NULL
**   inout_dcb :          contains return_code field
**
**
*/
void
des_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
char                  **input_dcb;
struct Ft_deselect_out_dcb **inout_dcb;
{

```

---

```

    /*
    ** Initialize the parameters with valid values.
    */
    *return_event_name = 3;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**          rat_parm_in
**
** DESCRIPTION:
** This routine assigns valid values to the ft_rattributes parameters.
**
** PARAMETERS:
** Outputs:
**   return_event_name : gets signalled when the event completes
**                       default: SYNCHRONOUS
**   input_dcb :         contains Ft_attributes_name field
**   inout_dcb :        contains return_code field
**
*/
void
rat_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name *return_event_name;
struct Ft_rattributes_in_dcb **input_dcb;
struct Ft_rattributes_out_dcb **inout_dcb;
{
    Api_rc outcome;
    Return_code res;
    struct Ft_diagnostic *diag = NULL;
    /*
    ** Initialize the parameters with valid values.
    */
    *return_event_name = 3;
    res = ft_didcb(FTiReadattributes, 0, (Octet **)input_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
    (*input_dcb)-attribute_names = FT_AN_PERMITTED_ACTIONS
    | FT_AN_CONTENT_TYPE
    | FT_AN_ID_OF_CREATOR
    | FT_AN_ACCESS_CONTROL;
    *inout_dcb = NULL;
}

```

## Example Programs

### Common Code Example

---

```
/*
**
**          ope_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_open parameters.
**
**
** PARAMETERS:
**   Outputs:
**     processing_mode :   type of permission
**     contents_type   :   document type
**     return_event_name : gets signalled when the event completes
**                         default: SYNCHRONOUS
**     input_dcb       :   contains ft_open input information
**     inout_dcb       :   contains return_code field
**
**
*/
void
ope_parm_in(processing_mode, contents_type, return_event_name,
            input_dcb, inout_dcb)
Ft_processing_mode      *processing_mode;
struct Ft_contents_type *contents_type;
Local_event_name        *return_event_name;
struct Ft_open_in_dcb   **input_dcb;
struct Ft_open_out_dcb  **inout_dcb;
{
  Api_rc                 outcome;
  Return_code            res;
  struct Ft_diagnostic   *diag = NULL;

  /*
  **   Initialize the parameters with valid values.
  **
  */
  *processing_mode = FT_PM_READ | FT_PM_REPLACE
                   | FT_PM_EXTEND | FT_PM_ERASE;
  ftam_3(contents_type);
  *return_event_name = SYNCHRONOUS;
  /*
  **   Initialize the input_dcb.
  **
  */
  res = ft_didcb(FTiOpen, 0, (Octet **)input_dcb, & outcome);
  if (res != SUCCESS)
    error_handler(outcome, diag);
  (*input_dcb)-concurrency_control = (struct Ft_concurrency_control *)
    malloc(sizeof(struct Ft_concurrency_control));
  concur_cntl_in( & ((*input_dcb)-concurrency_control) );
  /*
  **   Initialize the inout_dcb.
  **
  */
  *inout_dcb = NULL;
}
}
```

---

```

/*
**
**                                clo_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_close parameters.
**
** PARAMETERS:
**   Outputs:
**     return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :         input_dcb is NULL
**     inout_dcb :        contains return_code field
**
*/
void
clo_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
char                  **input_dcb;
struct Ft_close_out_dcb *inout_dcb;
{
    /*
    **   Initialize the parameters with all valid values.
    */
    *return_event_name = 2;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**                                cre_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_create parameters.
**
** PARAMETERS:
**   Outputs:
**     filename :          name of file to create
**     contents_type :    type of document
**     requested_access : type of file access
**     file_status :      state of the file
**     return_event_name : gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :        contains ft_create input information
**     inout_dcb :        contains return_code field
**
*/
void
cre_parm_in(filename, contents_type, requested_access, file_status,
             return_event_name, input_dcb, inout_dcb)
Ft_filename          *filename;
struct Ft_contents_type *contents_type;
Ft_file_actions      *requested_access;
enum Ft_file_status  *file_status;
Local_event_name     *return_event_name;
struct Ft_create_in_dcb **input_dcb;
struct Ft_create_out_dcb *inout_dcb;
{

```

## Example Programs

### Common Code Example

```

Api_rc                               outcome;
Return_code                           res;
struct Ft_diagnostic                   *diag = NULL;
/*
** Initialize the parameters with valid values.
*/
*filename = DEST_FNAME;
ftam_3(contents_type);
*requested_access = FT_FA_READ | FT_FA_REPLACE
                  | FT_FA_EXTEND | FT_FA_ERASE | FT_FA_READ_ATTRIBUTE
                  | FT_FA_CHANGE_ATTRIBUTE | FT_FA_DELETE_FILE;
*file_status = FT_NEW;
*return_event_name = 2;
/*
** Initialize the input_dcb.
*/
res = ft_didcb(FTiCreate, 0, (Octet **)input_dcb, & outcome);
if (res != SUCCESS)
    error_handler(outcome, diag);
(*input_dcb)-concurrency_control = (struct Ft_concurrency_control *)
    malloc(sizeof(struct Ft_concurrency_control));
concur_cntl_in( & ((*input_dcb)-concurrency_control) );
(*input_dcb)-create_file_pw.pointer = (Octet *)CREAT_PW;
(*input_dcb)-create_file_pw.length = CREAT_PW_LEN;
passwords_in( & ((*input_dcb)-file_passwords) );
(*input_dcb)-account = DEST_ACCOUNT;
/*
** Initialize the inout_dcb.
*/
*inout_dcb = NULL;
}
/*
**                               bgr_parm_in
** DESCRIPTION:
** This routine assigns valid values to the ft_bgroup parameters.
**
** PARAMETERS:
**   Outputs:
**     threshold :           number of calls within the group
**     return_event_name :  gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :          contains ft_bgroup input information
**     inout_dcb :         contains return_code field
**
*/
void
bgr_parm_in(threshold, return_event_name, input_dcb, inout_dcb)
Uint16
    *threshold;
Local_event_name
    *return_event_name;
char
    **input_dcb;
struct Ft_bgroup_out_dcb
    **inout_dcb;
{

```

---

```
    /*
    **   Initialize the parameters with all valid values.
    */
    *return_event_name = 1;
    *threshold = 2;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**           egr_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_egroup parameters.
**
** PARAMETERS:
**   Outputs:
**     return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**     input_dcb :         input_dcb is NULL
**     inout_dcb :        contains return_code field
**
*/
void
egr_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
char                  **input_dcb;
struct Ft_egroup_out_dcb **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    */
    *return_event_name = SYNCHRONOUS;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
```

## Example Programs

### Common Code Example

---

```
/*
**
**          rea_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_read parameters.
**
** PARAMETERS:
**   Outputs:
**     fadu_identity :      FADU identifier
**     access_context :    file structure
**     return_event_name : gets signalled when the event completes
**                          default: SYNCHRONOUS
**     input_dcb :         input_dcb is NULL
**     inout_dcb :        contains return_code field
**
**
*/
void
rea_parm_in(fadu_identity, access_context, return_event_name, input_dcb,
            inout_dcb)
struct Ft_fadu_identity      *fadu_identity;
enum Ft_access_context      *access_context;
Local_event_name            *return_event_name;
char                        **input_dcb;
struct Ft_read_out_dcb      **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    **
    */
    fadu_ident_in(fadu_identity);
    *access_context = FT_UNSTRUCTURED_ALL_DATA_UNITS;
    *return_event_name = SYNCHRONOUS;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**          wri_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_write parameters.
**
** PARAMETERS:
**   Outputs:
**     fadu_identity :      FADU identifier
**     fadu_operation :     type of write to perform
**     return_event_name :  gets signalled when the event completes
**                          default: SYNCHRONOUS
**     input_dcb :         input_dcb is NULL
**     inout_dcb :        contains return_code field
**
**
*/
void
wri_parm_in(fadu_identity, fadu_operation, return_event_name, input_dcb,
            inout_dcb)
```



---

```
struct Ft_fadu_identity      *fadu_identity;
enum Ft_fadu_operation      *fadu_operation;
Local_event_name           *return_event_name;
char                        **input_dcb;
struct Ft_write_out_dcb    **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    */
    fadu_ident_in(fadu_identity);
    *fadu_operation = FT_REPLACE;
    *return_event_name = SYNCHRONOUS;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
/*
**
**           sda_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_sdata parameters.
**
** PARAMETERS:
**   Outputs:
**   return_event_name :   gets signalled when the event completes
**                           default: SYNCHRONOUS
**   data_unit :         Actual data to send
**   inout_dcb :         contains return_code field
**
**
*/
void
sda_parm_in(return_event_name, inout_dcb)
Local_event_name      *return_event_name;
struct Ft_sdata_out_dcb **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    */
    *return_event_name = SYNCHRONOUS;
    *inout_dcb = NULL;
}
```

## Example Programs

### Common Code Example

---

```
/*
**
**          rda_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_rdata parameters.
**
**
** PARAMETERS:
**   Outputs:
**     des_requested :      number of FADUs requested to receive
**     return_event_name : gets signalled when the event completes
**                         default: SYNCHRONOUS
**     inout_dcb :        contains return_code field
**
**
*/
void
rda_parm_in(des_requested, return_event_name, inout_dcb)
Local_event_name      *return_event_name;
Uuint16               *des_requested;
struct Ft_rdata_out_dcb **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    **
    */
    *des_requested = 10;
    *return_event_name = SYNCHRONOUS;
    *inout_dcb = NULL;
}
/*
**
**          eda_parm_in
**
** DESCRIPTION:
**   This routine assigns valid values to the ft_edata parameters.
**
**
** PARAMETERS:
**   Outputs:
**     return_event_name : gets signalled when the event completes
**                         default: SYNCHRONOUS
**     input_dcb :        contains ft_edata input information
**     inout_dcb :        contains return_code field
**
**
*/
void
eda_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
struct Ft_edata_in_dcb **input_dcb;
struct Ft_edata_out_dcb **inout_dcb;
{
    Api_rc              outcome;
    Return_code         res;
    struct Ft_diagnostic *diag = NULL;
}
```

---

```
    /*
    ** Initialize the parameters with all valid values.
    */
    *return_event_name = SYNCHRONOUS;
    res = ft_didcb(FTiDataEnd, 0, (Octet **)input_dcb, & outcome);
    if (res != SUCCESS)
        error_handler(outcome, diag);
    (*input_dcb)-action_result = FT_AR_SUCCESS;
    (*input_dcb)-diagnostic = NULL;
    *inout_dcb = NULL;
}
/*
**
**          etr_parm_in
**
** DESCRIPTION:
** This routine assigns valid values to the ft_ettransfer parameters.
**
** PARAMETERS:
** Outputs:
**   return_event_name :   gets signalled when the event completes
**                         default: SYNCHRONOUS
**   input_dcb :         contains ft_ettransfer input information
**   inout_dcb :        contains return_code field
**
*/
void
etr_parm_in(return_event_name, input_dcb, inout_dcb)
Local_event_name      *return_event_name;
char                  **input_dcb;
struct Ft_ettransfer_out_dcb **inout_dcb;
{
    /*
    ** Initialize the parameters with all valid values.
    */
    *return_event_name = SYNCHRONOUS;
    *input_dcb = NULL;
    *inout_dcb = NULL;
}
```

## Example Programs

### Common Code Example

---

```
/*
**
**          wait_parm_in
**
** DESCRIPTION:
** This routine sets up the parameter values and the expected
** outcome values for em_wait calls. This routine allows
** the test programs to specify which set of parameter values
** you want to set.
**
**
** PARAMETERS:
** OUTPUT:
**   time :          timeout value
**   result :       contains return_code field for em_wait
**
*/
void
wait_parm_in(time, result)
Em_t          *time;
Api_rc       *result;
{
    /*
    ** Initialize "time" to infinite.
    */
    *time = -1;
    /*
    ** Initialize "result".
    */
    result-return_code = 9999;
    result-vendor_code = 9999;
}
/*
**
**          passwords_in
**
** DESCRIPTION:
** This routine assigns valid values to the file_passwords structure.
**
**
** PARAMETERS:
** Outputs:
**   file_passwords :       pointer to the Ft_file_passwords structure
**
*/
void
passwords_in(file_passwords)
struct Ft_file_passwords *file_passwords;
{
```



## Example Programs

### Common Code Example

---

```
/*
**  Change the legal_qualification.
*/
attributes-values.legal_qualification = "I'm_bad";
/*
**  Change the private_use.
*/
attributes-values.private_use.pointer = (Octet *)"private";
attributes-values.private_use.length = 7;
}
/*
**
**          ftam_3
**
** DESCRIPTION:
**  This routine assigns a ftam_3 document type to the Ft_contents_type
**  structure.
**
** PARAMETERS:
**  Outputs:
**    cont_type :          pointer to the Ft_contents_type structure
**
*/
void
ftam_3(cont_type)
struct Ft_contents_type          *cont_type;
{
struct Ft_dt_ftam_3             *ftm_3;
/*
**  Initialize all fields of the Ft_contents_type structure.
*/
cont_type->contents_form = FT_DOCUMENT_TYPE;
ftm_3 = (struct Ft_dt_ftam_3 *)malloc(sizeof(struct Ft_dt_ftam_3));
cont_type->contents_info.document.name.element =
(Sint32 *)malloc(5*sizeof(Sint32));
ftm_3->string_length = 512;
ftm_3->significance = FT_SS_NO_SIGNIFICANCE;
cont_type->contents_info.document.parameters = (char *)ftm_3;
cont_type->contents_info.document.name.length = 5;
cont_type->contents_info.document.name.element[0]=1;
cont_type->contents_info.document.name.element[1]=0;
cont_type->contents_info.document.name.element[2]=8571;
cont_type->contents_info.document.name.element[3]=5;
cont_type->contents_info.document.name.element[4]=3;
}
```

---

```
/*
**
**          attribute_in
**
** DESCRIPTION:
**   This routine assigns valid values to the Ft_attribute structure.
**
**
** PARAMETERS:
**   Outputs:
**     attributes :          pointer to the attributes structure
**
**
*/
void
attribute_in(attributes)
struct Ft_attributes          *attributes;
{
    /*
    **   Initialize all fields of Ft_attributes structure.
    **
    */
    attributes-mask = FT_AN_FILENAME | FT_AN_PERMITTED_ACTIONS
                    | FT_AN_CONTENT_TYPE | FT_AN_STORAGE_ACCT
                    | FT_AN_ID_OF_CREATOR | FT_AN_ID_OF_MODIFIER
                    | FT_AN_ID_OF_READER | FT_AN_ID_OF_ATT_MOD
                    | FT_AN_FILE_AVAILABILITY | FT_AN_FUTURE_FILESIZE
                    | FT_AN_LEGAL_QUAL | FT_AN_PRIVATE_USE;
    attributes-values.filename = SRC_FNAME;
    attributes-values.permitted_actions = FT_PA_READ | FT_PA_REPLACE
                    | FT_PA_EXTEND | FT_PA_ERASE | FT_PA_READ_ATTRIBUTE
                    | FT_PA_CHANGE_ATTRIBUTE | FT_PA_DELETE_FILE;
    ftam_3( & (attributes-values.contents_type) );
    attributes-values.storage_account = STORAGE_ACCOUNT;
    attributes-values.identity_of_creator = CREAT_ID;
    attributes-values.access_control.delete_ace = NULL;
    attributes-values.access_control.insert_ace = NULL;
    attributes-values.file_availability = FT_DEFERRED_AVAIL;
    attributes-values.future_filesize = 1000;
    attributes-values.legal_qualification = "legal_qual";
    attributes-values.private_use.pointer = (Octet *)"private";
    attributes-values.private_use.length = 7;
}
```

## Example Programs

### Common Code Example

---

```
/*
**
**          concur_cntl_in
**
** DESCRIPTION:
**   This routine assigns valid values to the Ft_concurrency_control
**   structure.
**
**
** PARAMETERS:
**   Outputs:
**     concurrency_control :      pointer to the Ft_concurrency_control
**                               structure
**
**
*/
void
concur_cntl_in(concurrency_control)
struct Ft_concurrency_control      **concurrency_control;
{
    /*
    **   Initialize the Ft_concurrency_control structure.
    **
    */
    *concurrency_control = NULL;
}
/*
**
**          fadu_ident_in
**
** DESCRIPTION:
**   This routine assigns valid values to the Ft_fadu_identity structure.
**
**
** PARAMETERS:
**   Outputs:
**     fadu_identity :          pointer to the Ft_fadu_identity structure
**
**
*/
void
fadu_ident_in(fadu_identity)
struct Ft_fadu_identity      *fadu_identity;
{
    /*
    **   Initialize all fields of Ft_fadu_identity structure.
    **
    */
    fadu_identity-fadu_form = FT_FADU_LOCATION;
    fadu_identity-fadu_info.fadu_location = FT_FIRST;
}
}
```



```
/*
**
**          nwc_parm_in
**
** DESCRIPTION:
**   This routine assigns values to the ft_nwcleared parameters.
**
**
** PARAMETERS:
**   OUTPUT:
**     return_event_name :   gets signalled when the event completes
**                          default: SYNCHRONOUS
**     inout_dcb :         contains return_code field
**
**
**/
void
nwc_parm_in(return_event_name, inout_dcb)
Local_event_name      *return_event_name;
struct Ft_nwcleared_out_dcb  **inout_dcb;
{
    /*
    **   Initialize the parameters with valid values.
    **
    **/
    *return_event_name = SYNCHRONOUS;
    *inout_dcb = NULL;
}
```

## Example Programs

### Common Code Example

```

#include %</opt/ftam/include/map.h>
#include %</opt/ftam/include/mapftam.h>
#include %<stdio.h>
void exit();
/*
**
**          error_handler
**
** DESCRIPTION:
** This routine is an example of handling failures experienced
** during the FTAM functions. This routine will display the
** error messages.
**
** PARAMETERS:
** INPUT:
** result :          the MAP and HP error information
** diag  :          the ISO error information
**
**
*/
void
error_handler(result, diag)
  Api_rc      result;
  struct Ft_diagnostic *diag;
{
  Return_code res;
  Api_rc      outcome;
  Octet      *return_string;
  Octet      *vendor_string;
  struct Ft_diagnostic *ft_diag = NULL;
  /*
  ** Initialize variables.
  */
  return_string = NULL;
  vendor_string = NULL;
  /*
  ** Call ft_gperror to get the printable strings for
  ** the return_code and vendor_code. Print the
  ** strings returned from ft_gperror.
  */
  res = ft_gperror( & result, & return_string, & vendor_string, & outcome
);
  if (res == SUCCESS) {
    if (return_string != NULL) {
      (void)printf("FTAM failure: return_code = %s\n", return_string);
      res = ft_fdmemory(return_string, & outcome);
      if (res != SUCCESS)
        error_handler(outcome, ft_diag);
    }
    /*
    ** Print the vendor_code if it exists.
    */
    if (vendor_string != NULL) {
      (void)printf("FTAM failure: vendor_code = %s\n", vendor_string);
      res = ft_fdmemory(vendor_string, & outcome);
      if (res != SUCCESS)
        error_handler(outcome, ft_diag);
    }
  }
}

```

```
/*  
**  Traverse the diagnostic list and print the further_details strings.  
*/  
while(diag != NULL) {  
    (void)printf("FTAM failure: diagnostic number = %d\n",  
                diag-error_id);  
    if(diag-further_details != NULL)  
        (void)printf("FTAM failure: diagnostic = %s\n",  
                    diag-further_details);  
    diag = diag-next;  
}  
exit(-1);  
}
```

Example Programs  
**Common Code Example**



## Document Types

Document types define the semantics for classes of files. A document type defines the type of file contents by specifying a constraint set and the possible data types in the file. Example file types include text files, binary files, sequential files, and directories.

The following table shows information about the parameters in the `cont_type.contents_info.document->parameters` field for FTAM-1, FTAM-2, FTAM-3, and INTAP-1 document types. In all cases, the maximum string length is 6144 octets.

Document Type	String Type	String Significance
FTAM-1	General String (Default type)	Not Significant
	Visible String	Variable or Fixed
	Graphic String	Variable or Fixed
	Ia5 String	Not Significant
FTAM-2	Visible String	Not Significant
	Graphic String (Default type)	Not Significant
FTAM-3	Octet String	Not Significant
INTAP-1	Octet String	Variable or Fixed

The NBS-9 document type parameter is used only on the `ft_open()` call. The attribute names you optionally supply list the attributes you will receive for each file in the directory. The default for HP-UX responders is all attributes. Other responders may have different defaults.

---

**NOTE**

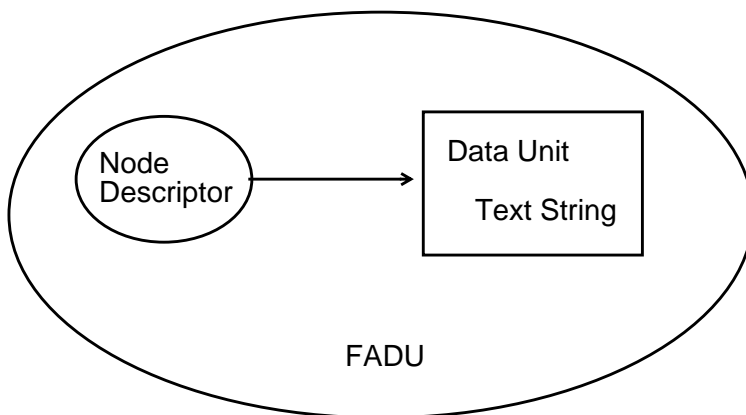
Refer to the “FTAM Data Structures” chapter for detailed information on setting document types using the `Ft_document_type` structure.

---

## FTAM-1 Document Type

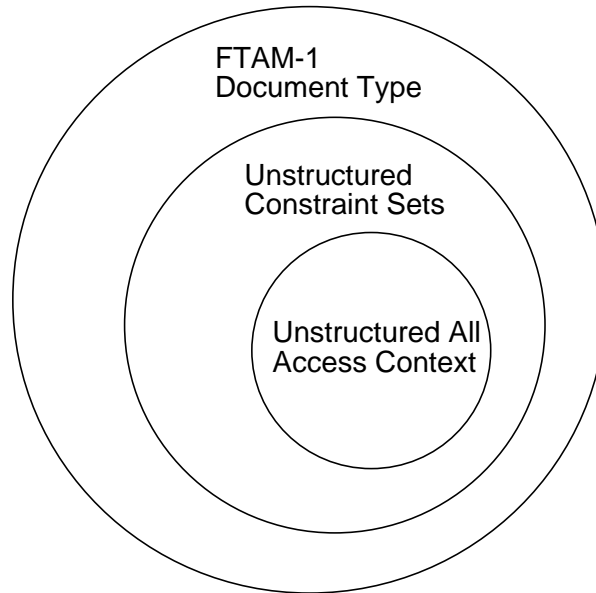
FTAM-1 has only one FADU that contains zero or more text strings. An FTAM-1 file contains only one string type, chosen from FT\_CL\_GENERAL\_STRING, FT\_CL\_IA5\_STRING, FT\_CL\_GRAPHIC\_STRING, or FT\_CL\_VISIBLE\_STRING (Figure 11-1)

**Figure 11-1** FTAM-1 Document Type Structure



You can only read from or write to the whole data unit (not parts of it). FTAM-1 uses the Unstructured constraint set to restrict the file model, and therefore uses the Unstructured All access context (Figure 11-2). Refer to the “Constraint Sets” and “Access Contexts” sections for detailed information.

**Figure 11-2 FTAM-1 Document Type Restrictions**



---

**NOTE** Refer to the “FTAM Data Structures” chapter for detailed information on setting the FTAM-1 document type using the `Ft_dt_ftam_1` structure.

---

### **FTAM-1 Document Semantics**

The following semantics are mandatory for FTAM-1 document types.

- FTAM-1 uses restrictions specified by the Unstructured constraint set.
- FTAM-1 does not specify the semantics of the character strings.
- The characters are from `FT_CL_GENERAL_STRING`, `FT_CL_IA5_STRING`, `FT_CL_GRAPHIC_STRING`, or `FT_CL_VISIBLE_STRING`.



If the class parameter is not present, the default is FT\_CL\_GENERAL\_STRING. The character strings may contain characters from any of the characters sets registered for use as C0, C1, G0, G1, G2, or G3 sets, including SPACE.<sup>1</sup> Refer to the “Character Sets” chapter for this list of characters.

- The string\_length parameter determines the length of maximum number of octets in each data element. HP-UX responders accept an unlimited string\_length. If string\_length is not present, the character strings have a maximum length of 6144.
- The string\_significance parameter determines the exact significance of the character strings. Three values are supported:
  - FT\_SS\_NO\_SIGNIFICANCE String (data element) boundaries are not preserved across transfers. For example, if you create a file using eight strings, you may or may not receive eight strings back when you read the file. Different FTAM implementations may break the file into strings in different ways.
  - FT\_SS\_FIXED String (data element) boundaries are preserved across transfers, and each string is of the size specified in the maximum string length parameter.
  - FT\_SS\_VARIABLE. String (data element) boundaries are preserved across transfers, and the length of each string is less than—or equal to—the size specified in the maximum string length parameter.

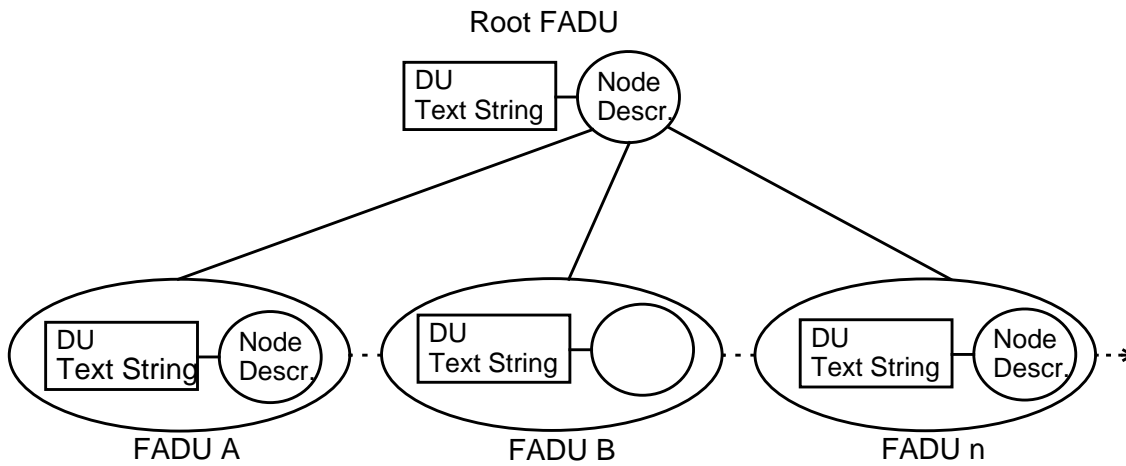
## FTAM-2 Document Type

FTAM-2 contains zero or more FADUs, each of which contains zero or more text strings. For FTAM-2, you can have any number of FADUs and any amount of data (Figure 11-3). The data units are distinct. For example, if you send five data units and ask for them back, you will receive five distinct data units in return.

---

1. Registered in the International Register of Coded Characters Sets for use with Escape Sequences

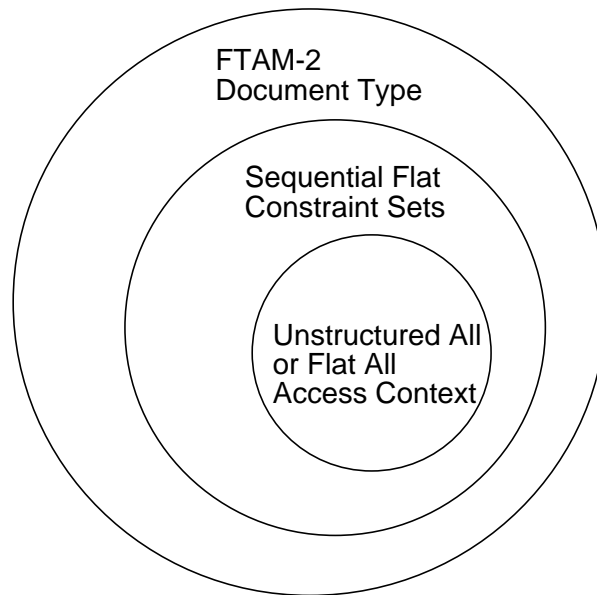
**Figure 11-3 FTAM-2 Document Type Structure**



FTAM-2 uses the Sequential Flat constraint set to restrict the file model, and therefore uses either the Unstructured All or Flat All access context (Figure 11-4).

Refer to the “Constraint Sets” and “Access Contexts” sections for detailed information.

**Figure 11-4 FTAM-2 Document Type Restrictions**



---

**NOTE**

Refer to the “FTAM Data Structures” chapter for detailed information on setting the FTAM-2 document type using the `Ft_dt_ftam_2` structure.

**FTAM-2 Document Semantics**

The following semantics are mandatory for FTAM-2 document types.

- FTAM-2 uses the restrictions specified by the Sequential Flat constraint set.
- FTAM-2 does not specify the semantics of the character strings.

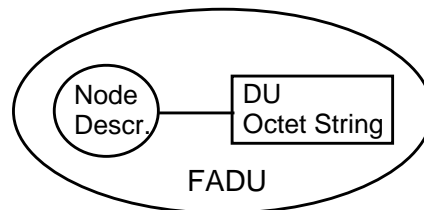
### Document Types

- If the class parameter is not present, the default is FT\_CL\_GRAPHIC\_STRING. The character strings may contain characters from any of the characters sets registered for use as G0, G1, G2, or G3 sets, including SPACE. <sup>1</sup> Refer to the “Character Sets” chapter for this list of characters.
- The string\_length parameter determines the maximum number of octets in each data element. HP-UX responders accept an unlimited string\_length. If the string\_length parameter is not present, the character strings are unbounded.
- The string\_significance parameter determines the exact significance of the character strings. Only the FT\_SS\_NO\_SIGNIFICANCE value is valid. The character string boundaries are not preserved when you store the file and do not contribute to the document semantics.

## FTAM-3 Document Type

FTAM-3 has only one FADU that contains zero or more binary octet strings (Figure 11-5).

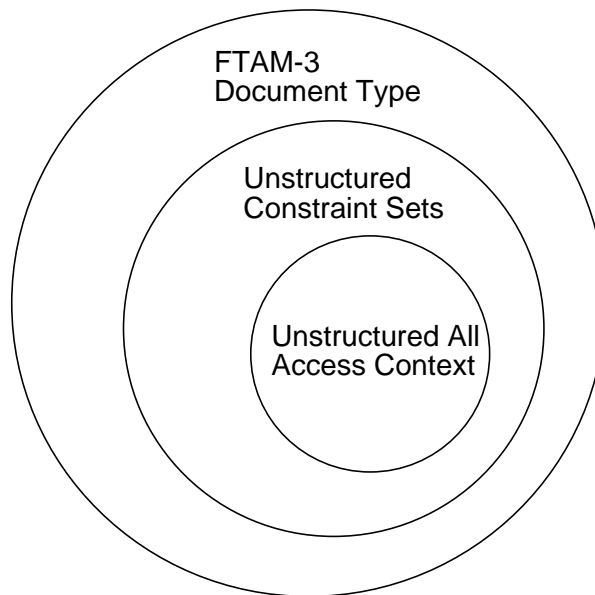
**Figure 11-5 FTAM-3 Document Type Structure**



FTAM-3 uses the Unstructured constraint set to restrict the file model, and therefore uses the Unstructured All access context (Figure 11-6). Refer to the “Constraint Sets” and “Access Contexts” sections for detailed information.

1. Registered in the International Register of Coded Characters Sets for use with Escape Sequences

**Figure 11-6 FTAM-3 Document Type Restrictions**



---

**NOTE**

---

Refer to the “FTAM Data Structures” chapter for detailed information on setting the FTAM-3 document type using the `Ft_dt_ftam_3` structure.

### **FTAM-3 Document Semantics**

The following semantics are mandatory for FTAM-3 document types.

- FTAM-3 uses the constraints specified by the Unstructured constraint set.
- FTAM-3 does not specify the semantics of the binary strings.
- Each binary string contains octets of any value from 0 to 255. FTAM-3 does not constrain the meanings of these values.
- The default class parameter is `FT_CL_OCTET_STRING`.
- The `string_length` parameter determines the length of each binary string. HP-UX responders accept an unlimited `string_length`. If the `string_length` parameter is not present, the binary strings are unbounded.

**Document Types**

- The `string_significance` parameter determines the exact significance of the character strings. Three values are supported:
  - `FT_SS_NO_SIGNIFICANCE` - the string or data element boundaries are not preserved across file transfers. For example, if you create a file using eight strings, you may or may not receive eight strings back when you read the file. Different FTAM implementations may break the file into strings in different ways.
  - `FT_SS_FIXED` - the string boundaries are preserved across transfers and each string is of the size specified in the maximum string length parameter.
  - `FT_SS_VARIABLE` - the string boundaries are preserved across transfers and the length of each string is less than or equal to the size specified in the maximum string length parameter.

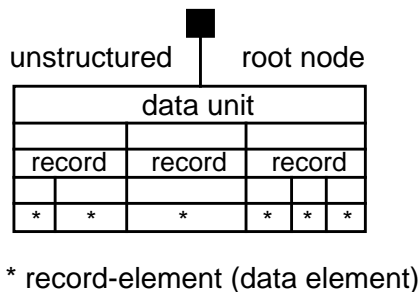
**INTAP-1 Document Type**

INTAP-1 has only one FADU that contains zero, one, or more records. Each record consists of octets of any value from 0 to 255. Figure 11-7 shows the INTAP-1 document type structure.

INTAP-1 uses the Unstructured constraint set to restrict the file model, and therefore uses the Unstructured All (UA) access context. Refer to the “Constraint Sets” and “Access Contexts” sections for detailed information.

Refer to the “FTAM Data Structures” chapter for detailed information on setting the INTAP-1 document type using the `Ft_dt_intap_1` structure.

**Figure 11-7 INTAP-1 Document Type Structure**



## INTAP-1 Document Semantics

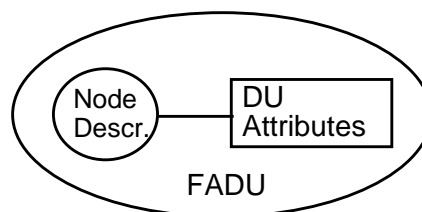
The following semantics are mandatory for INTAP-1 document types.

- An INTAP-1 document consists of one file access data unit, which consists only of zero, one, or more records. The order of these records is significant.
- Each record consists of octets of any value from 0 to 255. The meaning attached to these values is not constrained by the document type.
- There are no size or length limitations imposed by this definition, except for those specified here. Each record is of a length determined by the number of octets given by the maximum-record-length parameter. If this parameter is not present, the default is that the length of the records is unbounded. If the value of the record-significance parameter is variable, or if the parameter is not present, the length of each record is less than or equal to the length given in the maximum-record-length parameter. If the value is fixed, the length of each record is exactly equal to the length given.

## NBS-9 Document Type

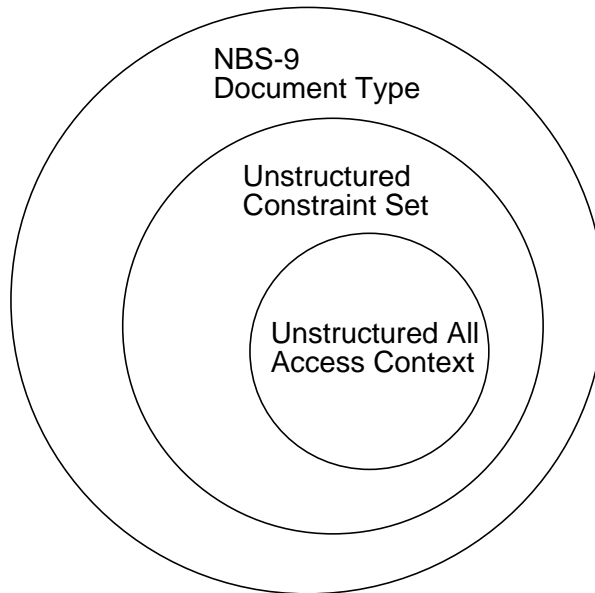
The NBS-9 document type models directories. Each NBS-9 document (directory) has only one FADU that contains zero or more data elements.(Figure 11-8).

**Figure 11-8** NBS-9 Document Type Structure



NBS-9 uses the Unstructured constraint set to restrict the file model, and therefore uses the Unstructured All access context (Figure 11-9). Refer to the “Constraint Sets” and “Access Contexts” sections for detailed information.

**Figure 11-9**      **NBS-9 Document Type Restrictions**



---

**NOTE**      Refer to the “FTAM Data Structures” chapter for detailed information on setting the NBS-9 document type using the Ft\_dt\_nbs\_9 structure.

---

### **NBS-9 Document Semantics**

The following semantics are mandatory semantics for NBS-9 document types.

- NBS-9 uses the Unstructured constraint set.
- Each data element contains the attributes of one file.
- The attribute\_names parameter is only used on the open call. It denotes the set of attributes to be returned for each file in the directory. If this parameter is not present on the open call, the default is to return all attributes.
- According to the NIST Phase III agreements, responders only need to return the filename attribute. However, HP-UX responders return all attributes as specified in the attribute\_names parameter supplied on the open call.



- The following table describes the legal operations on NBS-9 documents:

**Table 11-1**

**Legal Operations for the NBS-9 Document Type**

<b>Standard (specified by the NIST agreements)</b>	<b>HP-UX FTAM Supports these Additional Value-Added Operations</b>
select open read close deselect	create delete read attributes change attributes

## Constraint Sets

Every application needs a specific set of file structures, and any real system is likely to support only a limited range of file types, with restrictions on the ways you can modify files. Constraint sets express these limitations by restricting the following items.

- Range of allowable file structures
- Ways in which basic access actions can modify the structure

Constraint sets restrict (reduce) the file model. These sets help you efficiently model the type of file you are accessing.

The constraints sets supported by HP-UX FTAM are Unstructured and Sequential Flat.

Unstructured	Single, un-named data unit
Sequential Flat	Series of un-named data units

Each constraint set consists of the following items.

- A statement of the intended field of application
- A constraint set descriptor for reference in documentation
- Constraint set identifier for reference in the FTAM protocol
- Constraints on the node names
- Set of valid file access actions
- Qualified actions when writing to the file
- Set of file access contexts in which the file can be read
- State of the file when created
- FADU located immediately after the file is opened
- Definitions of “beginning of file” and “end of file”
- Manner in which you read from or write to the whole file
- General constraints on how you can modify the structure

- Special semantics of specific file access actions
- Allowable FADU identity forms for each of the file access actions

## Access Contexts

Each constraint set contains a list of available access contexts which allow you to further restrict the file structure view during read operations. The Access Contexts are Unstructured All Data Units (UA) and Flat All Data Units (FA).

For FTAM-1, FTAM-3, and INTAP-1, only the UA access context is available. For FTAM-2, both UA and FA are available. If you use the UA access context when transferring FTAM-2 files, the file simplifies to an FTAM-1 file; the node descriptor information does not transfer.

---

**NOTE**

---

Refer to the “FTAM Data Structures” chapter for detailed information on setting access contexts using the `Ft_access_context` structure.

Unstructured All (UA)	FT_UNSTRUCTURED_ALL_DATA_UNITS Only the data elements in the data unit (from the addressed FADU) transfer.
Flat All (FA)	FT_FLAT_ALL_DATA_UNITS Node descriptor information and data elements from the data unit (from the addressed FADU) transfer in the order that they are stored.

## Unstructured Constraint Set

The unstructured constraint set applies to files that you can transfer or access in whole, or extend; partial access is not available.

Refer to the following list for applicable basic constraints.

Constraint Set Name	Unstructured
Constraint Set Identifier	ISO Standard 8571 constraint-set (4) unstructured (1) (ASN.1 syntax for an identifier)
Node Names	None
Available File Access Actions	FT_FA_ERASE FT_FA_EXTEND FT_FA_READ FT_FA_REPLACE
Qualified Actions	None
Available Access Contexts	FT_UNSTRUCTURED_ALL_DATA_UNITS (UA)
Creation State	Root node with an empty data unit
Location After Open	FT_FIRST
Beginning of File	Not applicable
End of File	Not applicable
Read Whole File	Read in access context UA with a FADU identity of FT_FIRST
Write Whole File	Transfer a single data unit (without a node descriptor) with a FADU identity of FT_FIRST and an FT_FA_REPLACE file access action

Additional constraints on the Unstructured constraint set are as follows.

Structural Constraints	All actions in the unstructured constraint set result in a structure with a root node that has a data unit (although this data unit may be empty).
Action Constraints	The FT_FA_ERASE action yields a root node with an empty data unit.
Identity Constraints	The FADU identity is always FT_FIRST. Use this form of FADU identity with all permitted file actions.

## Sequential Flat Constraint Set

The Sequential Flat constraint set applies to files that are structured into a sequence of individual FADUs. You can access specific FADUs in the files by specifying their individual position in the sequence. Refer to the following list for applicable basic constraints.

Constraint Set Name	Sequential Flat
Constraint Set Identifier	ISO Standard 8571 constraint-set (4) sequential-flat (2) (ASN.1 syntax for an identifier)
Node Names	None
File Access Actions	FT_FA_ERASE FT_FA_LOCATE FT_FA_INSERT FT_FA_READ
Qualified Actions	None
Available Access Contexts	FT_FLAT_ALL_DATA_UNITS (FA) FT_UNSTRUCTURED_ALL_DATA_UNITS (UA)
Creation State	Root node without an associated data unit
Location After Open	Root node
Beginning of File	Root node

Document Types and Constraint Sets  
**Constraint Sets**

---

End of File	No node selected. FT_PREVIOUS gives last node in sequence; FT_CURRENT and FT_NEXT give an error
Read Whole File	Read in access context FA or UA with FADU identity of FT_BEGIN
Write Whole File	Transfer the series of leaf FADUs which would be generated by reading the whole file in access context FA; perform the transfer with a FADU identity of FT_END and an FT_FA_INSERT file access action

Additional constraints on the Sequential Flat constraint set are as follows:

Structural Constraints	The root node does not have an associated data unit. All children of the root node are leaf nodes and have an associated data unit. All arcs from the root node are of length one.
Action Constraints	You can use FT_FA_INSERT only at the end of a file, with FADU identity set to FT_END. The location following an FT_FA_INSERT action is FT_END. You can erase only at the root node to empty the file, with FADU identity set to FT_BEGIN. The result is a solitary root node without an associated data unit.
Identity Constraints	The FADU identity associated with the file action is FT_BEGIN, FT_END, FT_FIRST, or FT_NEXT. The allowable actions are as follows.

FT_FA_LOCATE	Actions are valid for each FADU identity on the whole file and on all leaf nodes.
FT_FA_READ	Actions are valid for the whole file at FT_BEGIN. Actions are valid for leaf nodes at FT_FIRST and FT_NEXT. Actions are not valid at FT_END.
FT_FA_INSERT	Actions are valid on the leaf node with the FT_END FADU identity.
FT_FA_ERASE	Actions are valid on the whole file only with the FT_BEGIN FADU identity.



## Former Intro ... Change

This chapter defines the available character sets for the HP FTAM/9000 product.

All numbers are in decimal notation.

IA5String	The characters available for the IA5String are all characters with ordinal values between 0 and 127, inclusive. The exceptions are the characters with ordinal values 14 and 15.
GeneralString	The characters available for the GeneralString are all characters with ordinal values between 0 and 255, inclusive.
GraphicString	The characters available for the GraphicString are all characters with ordinal values <ul style="list-style-type: none"><li>• between 32 and 126, inclusive, and</li><li>• between 160 and 255, inclusive.</li></ul> Additionally, the characters 27, 142, and 143 used within escape sequences are available.
VisibleString	The characters available for the VisibleString are all characters with ordinal values between 32 and 126, inclusive.



---

## Glossary

**Access Context** A set of restrictions that dictates how the information in a file is accessed; you can specify access contexts only when reading a file.

**Activity Attributes** Attributes that define the properties that exist only while a specific FTAM connection is being used; these attributes are not part of the file.

**Application Entity** A uniquely identified component of your application program that associates it with OSI; may be an FTAM initiator or FTAM responder.

**Application Interface** Your means of accessing the FTAM services (e.g., `ft_select()`).

**Application Service Element (ASE)** Any of the services provided to you in the Application layer of the OSI model (e.g., FTAM and ACSE).

**Association** An ISO term, synonymous with "connection."

**Association Control Service Element (ACSE)** An OSI layer seven protocol that provides association establishment, abort, and association termination.

**Attribute** Information that states a property of a file and takes a set of defined values, with each value having a defined meaning.

**connection\_id** A unique identifier of a connection for the duration of that connection.

**Constraint Set** A set of restrictions that specify the allowable file structures and the ways in which access actions can modify the structures.

**Context Free (CF) Function** A function that does not depend on the use of other FTAM functions.

**Context Sensitive (CS) Function** A function that depends on the use of other FTAM functions; you must call context sensitive functions in a specific order.

### **Data Control Block (DCB)**

Data structures used for passing information between user application programs and FTAM applications entities. The DCBs vary, depending on the function call.

---

## Glossary

**Data Element** The smallest piece of data whose identity is preserved when transferred by the Presentation Service. A data element may convey file contents information or protocol control information.

**Data Unit (DU)** The file unit that contains the actual file contents; a data unit may contain file structuring information or data elements.

**Document Type** The specification that defines the type of file contents by specifying the allowable constraint sets.

**Empty File** An FTAM file that contains only a root node with no associated data unit or node name.

**Event** The synchronous or asynchronous invocation of an FTAM function.

**Event Name** A value used to uniquely define an event; returned on asynchronous calls.

**FADU Identifier** A means of identifying the FADU; location is the only valid value.

**File Access** The inspection, modification, replacement, or erasure of a file's contents or a portion of it.

**File Access Data Unit (FADU)**

The smallest unit of the file access structure on which you can transfer, delete, extend, replace, or insert; contains zero or more data unit; each FTAM file consists of one or more FADUs.

**File Attribute** The name and other identifiable properties that describe a file, excluding file contents.

**File Contents** A file's data units, node names, and structuring information that you can manipulate during the File Open regime.

**File Management** The creation and deletion of files; the inspection or manipulation of file attributes.

**File Model** A view of a file's structure.

**File Protocol Data Unit (FPDU)** The blocks of information transferred between FTAM initiators and FTAM responders.

---

## Glossary

**File Service User** The portion of the application entity that invokes the FTAM service.

**File Transfer** A function that moves a part or the whole of a file's contents between open systems.

**File Transfer, Access and Management** An OSI layer seven protocol that allows peer-to-peer file transfer, record access, and file management.

**FTAM** See File Transfer, Access and Management

**ftam\_init** The executable name for an HP-UX FTAM initiator; sends your request to an FTAM responder.

**FTAM Protocol Machine (FPM)** The protocol followed by FTAM to accomplish file transfer, access, and management.

**ftam\_resp** The executable name for an HP-UX responder; daemon process that provides the interface for the HP-UX file system (real filestore) to the FTAM VFS.

**Grouping** A collection of FTAM functions that are transferred as a single FPDU.

**High Level Service (HLS)** A function that performs a sequence of low level functions, thereby eliminating the need to call the low level functions individually.

**Indication** Notification that a request arrived at its destination; notification that information is available. For example, a connect indication occurs when an FTAM responder receives your connect request (`ft_connect()`).

**Initial Configuration Store (ICS)** The database that stores the initial OSI network configuration.

**Initiator** The entity that submits requests on your behalf; when you call a function, the initiator sends your request to an FTAM responder.

**International Organization for Standards (ISO)** The United Nations body that conducts the balloting and ratification procedures for international communications standards.

**Logging** A diagnostic tool used to record significant network events; logging is always enabled while the network is operational.

---

## Glossary

**Low Level Service (LLS)** A function that provides the lowest level of service for the MAP 3.0 application interface; a LLS does not contain a sequence of other calls.

**Manufacturing Automation Protocol (MAP)** A seven layer communications specification; allows multivendor communication among factory automation equipment without custom links.

**Network Service Access Point (NSAP)** An addressable point in the Network layer that provides services to the Transport layer; identified by a unique value that allows network layers to identify the connection to its peer.

**Node** The elementary component from which a FADU tree is built.

**Node Name** A name used to locate a node that is part of the structuring information contained in a FADU.

**Note (an event)** To indicate that a request was serviced and its outcome is available. For example, issuing an em\_wait call is asking for the event that was noted for the longest period of time.

**Open Systems Interconnection (OSI)** A set of standards that define the seven layers of communication protocols for a network; ISO defines these standards.

**Packet** A unit of transmitted data, with a fixed maximum size.

**Presentation Address** An address used to communicate with other systems and processes; is unique within the network.

**Presentation Service Data Unit (PSDU)** The actual Presentation layer data transmitted to the peer Presentation layer.

**Protocol** A formalized set of rules for network communication.

**Real File** The actual file, including its name and attributes, that is mapped to the VFS.

**Real Filestore** A collection of actual files, including their names and attributes, that is mapped to the VFS.

**Regime** The period during which you can call a specific set of FTAM functions; the regime determines the activities that can occur at a given time and therefore, creates

---

## Glossary

the rules (protocols) for the FTAM state machine. (For example, the Data Transfer regime allows reading or writing of the file.) Regimes are nested.

**Responder** The entity that receives and services an initiator's request; if applicable, the responder returns a response; the responder interacts directly with the virtual filestore.

**Sequential Flat Constraint Set**

A constraint set that generates an access structure that

- consists of two levels (levels zero and one),
- may have data units at only the leaf nodes, and
- has no data unit at the root node.

**Service Provider Process (SPP)** See "Application Entity."

**State** See "Regime."

**Threshold** The number of functions (within a set of grouped functions) that must change regimes for the entire group to be successful.

**Tracing** A diagnostic tool that records specific events on individual HP OSI Transport

Services subsystems. Tracing degrades system performance and thus, is not always enabled.

**Unstructured Constraint Set** A constraint set that contains a single root node with a data unit; no file structure information is available.

**Virtual Filestore (VFS)** An abstract model for describing files and their attributes, and the possible actions on them; provides a uniform interface to all real file systems.

---

## **Glossary**

---

## Index

---

### A

Abort  
  ft\_abort(), 266  
  ft\_aeraset(), 264  
  ft\_ireceive(), 267  
  Indication, 267  
Abort Connections, 266  
Access Contexts, 40, 90, 443  
Access Control, 92  
  Defined Constants, 94  
Accessing FTAM Files, 272  
action\_result, 341  
Activate AE, 254  
Activity Attributes, 39  
AE, 37  
Ae\_dir\_name, 75  
Ae\_dir\_name Example, 81  
Ae\_label, 83  
Ae\_title, 84  
Ae\_title\_option, 85  
API Tracing, 349, 350, 351, 352  
Api\_rc, 87  
Application Entities  
  Activate, 254  
  Deactivate, 256  
Application Entities (AE), 37, 254  
Asynchronous Operations, 38  
  Errors, 338, 340  
  Function Return Values, 338  
  MAP 3.0 FTAM Errors, 340  
  result.return\_code, 340  
Asynchronous operations  
  and em\_hp\_select(), 176  
  and em\_hp\_sigio(), 176  
  and em\_wait(), 175  
Attribute Groups, 98  
Attributes, 103  
  Changing (HLCF), 230, 238  
  Changing (LLCS), 296  
  ft\_cattributes(), 296  
  ft\_fattributes(), 230  
  ft\_fattributes\_aet(), 238

  ft\_fattributes(), 226  
  ft\_fattributes\_aet(), 234  
  ft\_rattributes(), 294  
  Groups, 98  
  Names, 101, 105  
  Reading (HLCF), 226, 234  
  Reading (LLCS), 294  
  Values, 106

### B

Begin Grouping Functions, 305  
Bits, 69

### C

Cancel Data Transfer, 328  
Cancel Indication, 330  
Change Attributes (HLCF), 230, 238  
Change Attributes (LLCS), 296  
Character Sets, 448  
Close Files, 283  
Communication, 35  
Compiling Programs, 46  
Concurrency Control, 110  
  Default Settings, 112  
Connection Process, 252  
Connection\_id, 89  
Connections  
  Abort, 264  
  Establish, 257  
  Constants, Defined, 69  
  Constraint Sets, 40, 442  
    Sequential Flat, 42, 445  
    Unstructured, 41, 444  
  Contents Type, 113  
  Context Free Function, 52  
  Context Sensitive Function, 52  
  Converting Lines (HP-UX and FTAM), 64  
  Copy Files (HLCF), 206, 216  
  Create Files, 141, 274

### D

Data Elements, 122  
Data Structures, 66, 72  
  Ae\_dir\_name, 75  
  Ae\_label, 83  
  Ae\_title, 84  
  Ae\_title\_option, 85  
  Api\_rc, 87  
  Connection\_id, 89  
  Dir\_ava, 79  
  Dir\_dn, 76  
  Dir\_rdn, 77  
  Ft\_access\_context, 90  
  Ft\_access\_control, 92  
  Ft\_access\_control\_element, 94  
  Ft\_account, 97  
  Ft\_attribute\_groups, 98  
  Ft\_attribute\_names, 101, 105  
  Ft\_attribute\_values, 106  
  Ft\_attributes, 103  
  Ft\_concurrency\_control, 110  
  Ft\_contents\_form, 114  
  Ft\_contents\_info, 115  
  Ft\_contents\_type, 114  
  Ft\_contents\_type\_element, 113  
  Ft\_data\_element, 122  
  Ft\_data\_unit, 119  
  Ft\_dcb\_type, 126  
  Ft\_delete\_action, 128  
  Ft\_delete\_overwrite, 129  
  Ft\_diag\_type, 132  
  Ft\_diagnostic, 130  
  Ft\_document\_type, 115  
  Ft\_dt\_ftam\_1, 117  
  Ft\_dt\_ftam\_2, 117  
  Ft\_dt\_ftam\_3, 117  
  Ft\_dt\_intap\_1, 118  
  Ft\_entity\_ref, 132  
  Ft\_fadu\_form, 134  
  Ft\_fadu\_identity, 134  
  Ft\_fadu\_info, 135  
  Ft\_fadu\_operation, 137

---

## Index

---

- Ft\_file\_actions, 138
- Ft\_file\_lock, 111
- Ft\_file\_passwords, 140
- Ft\_file\_status, 141
- Ft\_filename, 143
- Ft\_functional\_units, 144
- Ft\_initiator\_identity, 147
- Ft\_node\_descriptor, 123
- Ft\_output, 156
- Ft\_processing\_mode, 148
- Ft\_qos, 149
- Ft\_service\_class, 150
- Ft\_single\_file\_pw, 155
- Ft\_structure\_id, 121
- Ft\_xxx\_out\_dcb, 157
- Inout\_dcb, 156
- Input\_dcb, 160
- Local\_event\_name, 163
- P\_address, 164
- Data Transfer
  - Cancel, 328
  - End, 324, 326
  - ft\_edata(), 324
  - ft\_ettransfer(), 326
  - Receive Cancel Indication, 330
- Data Types, 70
- Data Units, 41, 119
- Deactivate AE, 256
- Defined Constants, 69
  - Access Control, 94
  - FT\_AG, 100
  - FT\_AN, 102
  - FT\_FA, 94, 138
  - FT\_FU, 144
  - FT\_PA, 106
  - FT\_PM, 148
  - FT\_SC, 150
- Delete Files, 128
- Delete Files (HLCF), 242
- Deselect Files, 128, 286
- Device id, 112
- Diagnostic, 130
- diagnostic, 341
- Dir\_ava, 79
- Dir\_dn, 76
- Dir\_rdn, 77
- Directory Services, 72
  - Ae\_dir\_name, 75
  - Dir\_ava, 79
  - Dir\_dn, 76
  - Dir\_rdn, 77
  - Organization of, 72
- Document Types, 39, 113, 115, 430
  - FTAM-1, 39, 117
  - FTAM-2, 39, 117, 433
  - FTAM-3, 39, 117, 436
  - INTAP-1, 118, 438
  - NBS-9, 39, 439
- E**
- em\_gperror(), 196
- em\_fdmemory(), 172
- em\_hp\_select(), 176, 180
- em\_hp\_sigio(), 176, 188
- em\_wait(), 175, 177
- em\_wait() Example, 179
- End Grouping Functions, 306
- Ending Data Transfer, 324
- Erase Files, 301
- Error
  - action\_result, 341
  - Asynchronous Calls, 338
  - diagnostic, 341
  - Failure Types, 338
  - FPM, 340
  - Function Return Values, 338
  - HP-specific, 340
  - MAP 3.0 FTAM, 339
  - Recovery, 149
  - result.return\_code, 339
  - result.vendor\_code, 340
  - state\_result, 341
  - Synchronous Calls, 338
  - Translating, 196
  - VFS, 340
- Error Checking Example, 344, 380
- Error Handling, 335
- Example
  - Ae\_dir\_name, 81
  - Common Code, 382
  - em\_hp\_select(), 181, 186
  - em\_hp\_sigio(), 188
  - em\_wait(), 179
  - Error Checking, 344, 380
  - ft\_nwcleared(), 201
  - Global Definitions, 382
  - Logging Errors, 347
  - Support Functions, 382
- Exclusive Access
  - Effect of NFS, 112
- F**
- FADU, 41, 134
  - Actions On, 136, 137
  - Locations, 135, 299
- Failure Types, 338
- File
  - Close, 283
  - Copy (HLCF), 206, 216
  - Create, 141, 274
  - Delete, 128
  - Delete (HLCF), 242
  - Deselect, 128, 286
  - Erase, 301
  - Move (HLCF), 211, 221
  - Open, 141
  - Open (HLCS), 288
  - Select, 141, 277
- File Access Data Unit, 41
- File Actions, 138
- File Attributes, 39
- File Locks, 111
- File Passwords, 140, 155
- File Structure Model, 41
- Filename, 143



---

## Index

- 
- FPM Errors, 341
  - ft\_gpperror(), 198
  - ft\_abort(), 266
    - Ft\_abort\_in\_dcb, 266
    - Ft\_output, 266
  - Ft\_access\_context, 90
  - Ft\_access\_control, 92
  - Ft\_access\_control\_element, 94
  - Ft\_account, 97
  - ft\_aeactivation(), 254
    - Ft\_aeactivation\_in\_dcb, 254
    - Ft\_output, 255
  - ft\_aedeactivation(), 256
    - Ft\_output, 256
  - ft\_aereset(), 264
    - Ft\_output, 264
  - FT\_AG Defined Constants, 100
  - FT\_AN Defined Constants, 102
  - Ft\_attribute\_groups, 98
  - Ft\_attribute\_names, 101, 105
  - Ft\_attribute\_values, 106
  - Ft\_attributes, 103
  - ft\_bgroup(), 305
    - Ft\_bgroup\_out\_dcb, 305
  - ft\_cancel(), 328
    - Ft\_cancel\_in\_dcb, 328
    - Ft\_cancel\_out\_dcb, 328
  - ft\_cattributes(), 296
    - Ft\_cattributes\_in\_dcb, 297
    - Ft\_cattributes\_out\_dcb, 297
  - ft\_close(), 282
    - Ft\_close\_out\_dcb, 283
  - Ft\_concurrency\_control, 110
  - ft\_connect(), 257
    - Ft\_connect\_in\_dcb, 258
    - Ft\_connect\_out\_dcb, 259
  - Ft\_contents\_form, 114
  - Ft\_contents\_info, 115
  - Ft\_contents\_type, 114
  - Ft\_contents\_type\_element, 113
  - ft\_copy(), 206
  - ft\_copy\_aet(), 216
  - ft\_create(), 274
    - Ft\_create\_in\_dcb, 275
    - Ft\_create\_out\_dcb, 275
  - Ft\_data\_element, 122
  - Ft\_data\_unit, 119
  - Ft\_dcb\_type, 126
  - ft\_delete(), 284
    - Ft\_delete\_out\_dcb, 284
  - Ft\_delete\_action, 128
  - Ft\_delete\_overwrite, 129
  - ft\_deselect(), 286
    - Ft\_deselect\_out\_dcb, 286
  - ft\_dfddb(), 171
  - Ft\_diag\_type, 132
  - Ft\_diagnostic, 130
  - ft\_didcb(), 170
  - Ft\_document\_type, 115
  - Ft\_dt\_ftam\_1, 117
  - Ft\_dt\_ftam\_2, 117
  - Ft\_dt\_ftam\_3, 117
  - Ft\_dt\_intap\_1, 118
  - ft\_edata(), 324
    - Ft\_edata\_in\_dcb, 324
    - Ft\_edata\_out\_dcb, 324
  - ft\_egroup(), 306
    - Ft\_egroup\_out\_dcb, 306
  - Ft\_entity\_ref, 132
  - ft\_erase(), 301
    - Ft\_erase\_out\_dcb, 301
  - ft\_ettransfer(), 326
    - Ft\_ettransfer\_out\_dcb, 326
  - FT\_FA Defined Constants, 94, 138
  - Ft\_fadu\_form, 134
  - Ft\_fadu\_identity, 134
  - Ft\_fadu\_info, 135
  - Ft\_fadu\_operation, 137
  - ft\_fcattributes(), 230
    - Ft\_fcattributes\_in\_dcb, 231
    - Ft\_fcattributes\_out\_dcb, 231
  - ft\_fcattributes\_aet(), 238
    - Ft\_fcattributes\_in\_dcb, 239
    - Ft\_fcattributes\_out\_dcb, 239
  - ft\_fcclose(), 292
    - Ft\_fcclose\_out\_dcb, 292
  - Ft\_fcopen(), 211
    - Ft\_fmmove\_in\_dcb, 212
    - Ft\_fmmove\_out\_dcb, 212
  - Ft\_fmmove\_aet(), 221
    - Ft\_fmmove\_in\_dcb, 222
    - Ft\_fmmove\_out\_dcb, 223
  - ft\_fopen(), 288
    - Ft\_fopen\_in\_dcb, 289
    - Ft\_fopen\_out\_dcb, 290
  - ft\_frattributes(), 226
    - Ft\_frattributes\_in\_dcb, 227
    - Ft\_frattributes\_out\_dcb, 227
  - ft\_frattributes\_aet(), 234
    - Ft\_frattributes\_in\_dcb, 235
    - Ft\_frattributes\_out\_dcb, 235
  - FT\_FU Defined Constants, 144
  - Ft\_functional\_units, 144
  - Ft\_initiator\_identity, 147
  - ft\_ireceive(), 267
    - Ft\_indication\_out\_dcb, 268
  - ft\_locate(), 299
    - Ft\_locate\_out\_dcb, 299
  - Ft\_node\_descriptor, 123
  - ft\_nwcleared(), 200

---

## Index

---

- Functions
    - ft\_open(), 280
    - ft\_open\_in\_dcb, 280
    - ft\_open\_out\_dcb, 281
    - ft\_output, 156, 256, 264, 266
    - FT\_PA Defined Constants, 106
    - FT\_PM Defined Constants, 148
    - ft\_processing\_mode, 148
    - Ft\_qos, 149
    - ft\_rattributes(), 294
      - Ft\_rattributes\_in\_dcb, 295
      - Ft\_rattributes\_out\_dcb, 295
    - ft\_rcancel(), 330
      - Ft\_rcancel\_out\_dcb, 330
    - ft\_rdata(), 315
      - Ft\_rdata\_out\_dcb, 315
    - ft\_rdataqos(), 317
      - Ft\_rdataqos\_out\_dcb, 318
    - ft\_read(), 313
      - Ft\_read\_out\_dcb, 314
    - ft\_rrequest(), 263
      - Ft\_relreq\_in\_dcb, 263
      - Ft\_relreq\_out\_dcb, 263
    - FT\_SC Defined Constants, 150
    - ft\_sdata(), 321
      - Ft\_sdata\_out\_dcb, 322
    - ft\_select(), 277
      - Ft\_select\_in\_dcb, 278
      - Ft\_select\_out\_dcb, 278
    - Ft\_service\_class, 150
    - Ft\_single\_file\_pw, 155
    - Ft\_structure\_id, 121
    - ft\_write(), 319
      - Ft\_write\_out\_dcb, 320
    - Ft\_xxx\_out\_dcb, 157
  - FTAM
    - Interaction with NFS, 112
    - FTAM Communication, 35
    - FTAM Libraries, 46
    - FTAM Overview, 26
    - ftam\_init, 36
    - ftam\_resp, 36
  - FTAM-1 Document Types, 39, 117
  - FTAM-2 Document Types, 39, 117, 433
  - FTAM-3 Document Types, 39, 117, 436
  - Functional Units, 144
  - Functions, 46
    - Connection Establishment, 250
      - em\_gperror(), 196
      - em\_fdmemory(), 172
      - em\_hp\_select(), 180
      - em\_hp\_sigio(), 188
      - em\_wait(), 177
    - ft\_gperror(), 198
    - ft\_abort(), 266
    - ft\_aeactivation(), 254
    - ft\_aedeactivation(), 256
    - ft\_aereset(), 264
    - ft\_bgroup(), 305
    - ft\_cancel(), 328
    - ft\_cattributes(), 296
    - ft\_close(), 282
    - ft\_connect(), 257
    - ft\_create(), 274
    - ft\_delete(), 284
    - ft\_deselect(), 286
    - ft\_dfdcb(), 171
    - ft\_didcb(), 170
    - ft\_edata(), 324
    - ft\_egroup(), 306
    - ft\_erase(), 301
    - ft\_ettransfer(), 326
    - ft\_fcattributes(), 230
    - ft\_fcattributes\_aet(), 238
    - ft\_fcclose(), 292
    - ft\_fcopy(), 206
    - ft\_fcopy\_aet(), 216
    - ft\_fdelete(), 242
    - ft\_fdelete\_aet(), 245
    - ft\_fdmemory(), 173
    - ft\_fmove(), 211, 221
    - ft\_fopen(), 288
    - ft\_frattributes(), 226
    - ft\_frattributes\_aet(), 234
    - ft\_ireceive(), 267
    - ft\_locate(), 299
    - ft\_nwcleared(), 200
    - ft\_rattributes(), 294
    - ft\_rcancel(), 330
    - ft\_rdata(), 315
    - ft\_rdataqos(), 317
    - ft\_read(), 313
    - ft\_rrequest(), 263
    - ft\_sdata(), 321
    - ft\_select(), 277
    - ft\_write(), 319
    - Grouping, 303
    - List of, 54
    - Support, 167
  - Functions Per Regime, 52
  - Functions Return Values, 338
  - Functions, HLCF, 204
- ### G
- Grouping Functions, 154, 303
    - Begin, 305
    - End, 306
    - ft\_bgroup(), 305
    - ft\_egroup(), 306
- ### H
- Header Files, 68
  - High Level Service (HLS), 51
  - High Level, Context Free
    - Functions, 204
  - HLS, 51
  - HP-specific Errors, 340
- ### I
- Initiator, 36
  - Initiator Identity, 147
  - Inode number, 112
  - inout\_dcb, 61, 156

---

## Index

- 
- input\_dcb, 160
  - INTAP-1
    - Document Types, 39
  - INTAP-1 Document Types, 39, 118, 438
  - International Standards Organization (ISO), 28
  
  - K**
  - Kernel Group, 99
  
  - L**
  - Libraries, 46
  - Linking FTAM Libraries, 46
  - lint(1) Library, 46
  - LLS, 51
  - Local\_event\_name, 163
  - Locate FADU, 299
  - Logging Errors, 346
  - Logging Errors Example, 347
  - Low Level Service (LLS), 51
  
  - M**
  - Managing FTAM Connections, 250
  - Managing FTAM Files, 272
  - Manufacturing Automation Protocol (MAP) 3.0, 29
  - MAP 3.0, 29
  - MAP 3.0 FTAM Errors, 339
  - map.h, 68
  - mapftam.h, 68
  - Memory Allocation, 170, 171, 172
    - Input, 161
    - Output, 62, 158
  - Move Files (HLCSF), 211, 221
  
  - N**
  - NBS Implementors Agreements, 29
  
  - NBS-9 Document Types, 39, 439
  - NFS
    - Interaction with FTAM, 112
    - no value available, 29
  - Node Descriptors, 41, 123
  - NTAP-1 Document Types, 118
  
  - O**
  - Object\_id, 70
  - Object\_string, 71
  - Open Files, 141
  - Open Files (HLCS), 288
  - Open Files (LLCS), 280
  - File
    - Open (LLCS), 280
  - Open Systems Interconnection (OSI), 28
  - OSI, 28
  - Output Errors, 339
  
  - P**
  - P\_address, 164
  - Parameters, 60
    - em\_gperror(), 197
    - em\_fdmemory(), 173
    - ft\_gperror(), 199
    - ft\_abort(), 267
    - ft\_aeactivation(), 255
    - ft\_aedeactivation(), 256
    - ft\_aereset(), 265
    - ft\_bgroup(), 305
    - ft\_cancel(), 329
    - ft\_cattributes(), 298
    - ft\_close(), 283
    - ft\_connect(), 259
    - ft\_create(), 276
    - ft\_delete(), 285
    - Ft\_deselect(), 286
    - ft\_dfdcb(), 172
    - ft\_didcb(), 171
    - ft\_edata(), 325
    - ft\_egrp(), 307
    - ft\_erase(), 302
    - ft\_ettransfer(), 327
    - ft\_fattributes(), 231, 240
    - ft\_fclose(), 293
    - ft\_fcopy(), 208, 218
    - ft\_fdelete(), 243, 246
    - ft\_fdmemory(), 174
    - ft\_fmove(), 213
    - ft\_fmove\_aet(), 223
    - ft\_fopen(), 290
    - ft\_frattributes(), 228, 235
    - ft\_ireceive(), 269
    - ft\_locate(), 300
    - ft\_nwcleared(), 201
    - ft\_open(), 281
    - ft\_rattributes(), 295
    - ft\_rcancel(), 331
    - ft\_rdata(), 316, 318
    - ft\_read(), 314
    - ft\_rrequest(), 264
    - ft\_sdata(), 323
    - ft\_select(), 278
    - ft\_write(), 321
    - Order of, 60
  - Passwords, 140, 155
  - Presentation Address, 164
  - Private Group, 99
  - Profiles Implemented, 30
  - Programming
    - Recommendations, 63
  
  - Q**
  - Quality of Service, 149
  
  - R**
  - Read Attributes (HLCSF), 226, 234
  - Read Attributes (LLCS), 294
  - Read Data Request, 313
  - Receive Cancel Indication, 330
  - Receive Data, 315, 317
  - Recommendations

---

# Index

---

Programming, 63  
Recovery, Error, 149  
Regimes, 34  
  Data Transfer, 50  
  File Open, 49  
  File Selection, 49  
  FTAM, 48  
  Functions Within, 52  
  Guidelines, 47  
Release Requests, 263  
Resources Available, 200  
Responder, 36  
result.return\_code, 339  
result.vendor\_code, 340  
return\_code, 87  
return\_event\_name, 178

## S

Security Group, 99  
Select Files, 141, 277  
Send Data, 321  
Sequential Flat Constraint Sets,  
  445  
Server provider process (SPP),  
  36  
Service Class, 150  
Setting Bits, 69  
SPP, 36  
state\_result, 341  
Storage Group, 99  
Support Functions, 167  
Synchronous Operations, 38  
  Errors, 338, 340  
  Function Return Values, 338  
  MAP 3.0 FTAM Errors, 340  
  result.return\_code, 340

## T

Tracing, API, 349, 350, 351, 352  
transferring FTAM Data, 310  
Translating Error Messages,  
  196

## U

Unstructured Constraint Set,  
  41, 444

## V

VFS, 31  
Virtual Filestore (VFS), 31

## W

Write Data Request, 319