

**Tivoli Data Exchange  
Version 1.2.5**

**Technical Reference**

Copyright 2001, 2002 Tivoli, Inc. All rights reserved.

5/20/02

Tivoli Data Exchange, Tivoli, and the Tivoli logo are registered trademarks or trademarks of Tivoli, Inc. AS/400, AIX, MQSeries, and OS/390 are registered trademarks of IBM Corp. InstallShield is a registered trademark of InstallShield Software Corporation. Win 32 is a registered trademark of Microsoft Corp. UNIX is a registered trademark of The Open Group. Solaris and Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other trademarks are property of their respective owners.

# Contents

<b>chapter 1</b>	<b>About This Guide . . . . .</b>	<b>9</b>
	About Tivoli Data Exchange . . . . .	9
	Who Should Read This Guide . . . . .	10
	How This Guide Is Organized . . . . .	11
	Contacting Customer Support . . . . .	12
	Where to Look for More Information . . . . .	12
	Conventions . . . . .	12
<b>chapter 2</b>	<b>Tivoli Data Exchange Concepts . . . . .</b>	<b>15</b>
	Introduction . . . . .	15
	Why FTF? . . . . .	15
	Architectural Overview . . . . .	17
	FTF Interfaces . . . . .	18
	FTF Manager . . . . .	19
	FTF Sender . . . . .	19
	FTF Receiver . . . . .	20
	FTF Status . . . . .	20
<b>chapter 3</b>	<b>Tivoli Data Exchange Interface . . . . .</b>	
	<b>Commands . . . . .</b>	<b>23</b>
	Overview . . . . .	24
	FTF . . . . .	25
	Transferring One File to Multiple Destinations . . . . .	29
	FTFCNCL . . . . .	50
	FTFPING . . . . .	53
	Syntax . . . . .	53
	FTFSTAGE . . . . .	62
	Using Staging Queues . . . . .	67
	FTFSTAT . . . . .	72
	Using the FTF Publish/Subscribe Function . . . . .	85
	FTFBRK . . . . .	86
	Supported Operating Systems . . . . .	86
	Description . . . . .	86
	Syntax . . . . .	86

Queue Manager Arguments	87
Process Arguments	87
Environment Arguments	88
Help Arguments	89
FTFPUB	90
Supported Operating Systems	90
Description	90
Syntax	90
Queue Manager Node Arguments	92
Process Arguments	92
Environment Arguments	95
User Exit Arguments	96
Data Specification Arguments	97
Help Arguments	99
FTFSUB	100
Supported Operating Systems	100
Description	100
Syntax	100
Queue Manager Node Arguments	101
Environment Arguments	102
Subscription Criteria Arguments	103
Process Arguments	103
User Exit Arguments	104
Data Specification Arguments	106
AS/400 Arguments	108
MVS/ESA Arguments	110
Help Arguments	112

## **chapter 4      Component Configuration Commands . . . . . 115**

Overview	116
FTFCFG	117
Syntax	119
FTFEND	121
FTFLOG	127
FTFMGR	134
FTFRCV	141
FTFSDR	148
FTFSTART	155

## **chapter 5      FTF COBOL API . . . . . 161**

The FTFCOBOL Copybook	162
FTFH-HEADER	166

Request Fields .....	167
FTFPing Fields .....	173
FTFStatus Summary List Fields .....	174
FTF STAT Filter Fields .....	175
Issuing FTF Transfer Requests .....	176
Issuing FTF Ping Requests .....	179
Issuing FTF Status Requests .....	179

## **chapter 6      C API Reference ..... 181**

FTFCancel .....	183
FTFPing .....	185
FTFReq .....	187
FTFReq API .....	190
FTFShutdown .....	192
FTFStage .....	194
FTFStatusDelete .....	196
FTFStatusFreeDetailList .....	198
FTFStatusFreeSummaryList .....	200
FTFStatusGetDetailList .....	202
FTFStatusGetSummaryList .....	204
FTFSubmitStatusMsg .....	207

## **chapter 7      C Data Structures ..... 209**

Component Status Messages .....	210
FTF Manager Status Messages .....	211
FTF Sender Status Messages .....	211
FTF Receiver Status Messages .....	212
FTFAS400FileInfo .....	213
FTFCA .....	215
FTFCancelInfo .....	217
FTFExitAuthInfo .....	219
FTFExitFileInfo .....	221
FTFExitInfo .....	225
FTFExitJobInfo .....	229
FTFExitQMgrsInfo .....	232
FTFExitRequestInfo .....	234
FTFExitSourceFileInfo .....	236
FTFExitStatusOffload .....	239
FTFExitTargetFileInfo .....	241
FTFExitUserInfo .....	245
FTFIdentifiersInfo .....	247
FTFJobInfo .....	248

FTFPingInfo	251
FTFQMgrsInfo	254
FTFRequestMsgInfo	256
FTFShutdownInfo	259
FTFShutdownReply	261
FTFSourceFileInfo	263
FTFStagedListMsgInfo	266
FTFStagedTransactionMsgInfo	267
FTFStageMsgInfo	269
FTFStatDetail	271
FTFStatDetailList	281
FTFStatSummary	282
FTFStatSummaryList	286
FTFStatusDeleteInfo	288
FTFStatusGetDetailListInfo	290
FTFStatusGetSummaryListInfo	292
FTFStatusSummaryFilter	294
FTFSubscribeMsgInfo	298
FTFSubscriptionListMsgInfo	301
FTFSubscriptionMsgInfo	302
FTFTargetFileInfo	304
FTFUserInfo	308

## chapter 8      **Tivoli Data Exchange and MQSeries . . . . .**

### **Security . . . . . 311**

Assumptions	311
Introduction	312
Modes of Security	312
MQSeries Security Enabling Interface	313
General MQSeries Security Concepts	313
MQSeries Alternate User Authority	314
Tivoli Data Exchange External Security Manager	314
Enabling Security for Tivoli Data Exchange	315
Enabling Alternate User Authority	315
Enabling File Resource Security	315

## chapter 9      **XML Integration. . . . . 319**

Assumptions	319
Operating Requirements	319
Overview	320
Implementation Details	321

Input .....	321
Output .....	322
Return Objects .....	324
XML Element Names .....	325
Examples .....	337
Sample DTD .....	339

## **chapter 10      Multi-File Connector . . . . . 343**

Assumptions .....	343
Introduction .....	343
Multi-File Connector Installation .....	344
Connector Entry Points .....	345
Overview .....	345
Invoking the Multi-File Connector .....	346
Sample Transfer from Win 32 to OS/390 .....	346
Warnings: .....	347
Output .....	347
VSAM Data .....	348
Directory / PDS Transfers .....	351
XML Configuration File .....	352
Error Conditions .....	354

## **chapter 11      The File-to-Message Connector . . . . . 395**

Assumptions .....	395
Introduction .....	396
DataExtract Processing .....	396
DataLoad Processing .....	396
File-to-Message Connector Installation .....	396
Arguments .....	397
File-to-Message Modifiers .....	399
DataExtract Modifiers .....	399
DataLoad Modifiers .....	401
Delimiter Processing .....	403
Record Processing .....	405
Logging and Status .....	406
Examples .....	407
Sending a Single Message to a File .....	407
Sending Multiple Messages to a File .....	408
Sending a Specific Message to a File .....	408
Sending a Message to a File Using the -DELSRC Option .....	409
Waiting for a Single Message and Transferring to a File .....	410
Sending a File to a Single Message .....	410

	Sending a File Using and Keeping Delimiters . . . . .	411
	Sending a File Using and Removing Delimiters . . . . .	412
	Error Messages . . . . .	413
<b>chapter 12</b>	<b>FTFBPI Wrapper . . . . .</b>	<b>415</b>
	Assumptions . . . . .	415
	FTFBPI Wrapper . . . . .	416
<b>appendix a</b>	<b>Security Authorization Exit JCL . . . . .</b>	<b>425</b>
	FTFSTART Relink JCL . . . . .	426
	FTFSTART Execution JCL . . . . .	427
<b>appendix b</b>	<b>Tivoli Data Exchange GUI Environment . . . . .</b>	
	<b>Variables . . . . .</b>	<b>429</b>
	Command-File Environment Variables . . . . .	429
	Variable Settings Outside the Command File . . . . .	430
	<b>Index . . . . .</b>	<b>431</b>



---

---

# About This Guide

The *Tivoli Data Exchange Technical Reference* describes the commands, APIs, and data structures used for data transfers. It is intended as a reference for specific details about these items. For information about how to use Tivoli Data Exchange (TDE) to transfer data, see the *Tivoli Data Exchange User's Guide*.

This chapter contains guidelines about the information in this manual and the conventions used to present the information. It contains the following sections:

Section	Page
About Tivoli Data Exchange	9
Who Should Read This Guide	10
How This Guide Is Organized	11
Contacting Customer Support	12
Where to Look for More Information	12
Conventions	12

## About Tivoli Data Exchange

Tivoli Data Exchange enables the collection and distribution of business-critical data, regardless of content type or size, across major platforms and network protocols. The benefits include reduced transfer times, saved network bandwidth, and easy scalability to meet growing e-business needs. Tivoli Data Exchange leverages and exploits the inherent strengths of MQSeries to provide secure, efficient, and reliable transfer of data across all platforms in an enterprise through compression and parallel delivery paths. It uses an IBM MQSeries engine for any-to-any connection, assured delivery, and data integrity checking. Tivoli Data Exchange caters to heterogeneous networks by automatically handling various protocols including SNA, TCP/IP, and X.25 and supports over 35 platforms. The result is a scalable solution that seamlessly converts from SNA to IP with no product adjustments, as well as seamless integration with existing in-house applications via its exposed API.

## About This Guide

### *Who Should Read This Guide*

Tivoli Data Exchange provides the following services:

- Moves and accepts files among all supported platforms
- Provides data compression, if you require it
- Performs binary and ASCII transfers
- Transfers files regardless of their size, format, or destination
- Allows individual status tracking for any phase of the file transfer at any node across the enterprise

Tivoli Data Exchange provides integration capabilities on top of the software, including business process and workflow integration. Exits can be customized by users and can be employed in a plug-and-play manner. They are called at strategic points during a Tivoli Data Exchange transaction.

Two connectors, Multi-File and File-to-Message, allow you to pass data to the Tivoli Data Exchange transfer engine with a simple, consistent application programming interface (API). The data is protected through syncpoint control. Transactional status is distributed across the enterprise. High-performance multiplexing capabilities are employed transparently.

No complex changes, design requirements, or re-engineering efforts are required for existing applications to take advantage of these features.

## Who Should Read This Guide

This guide is intended for the following groups:

- Developers using Tivoli Data Exchange to design file-transfer solutions
- Users performing Tivoli Data Exchange solutions
- Tivoli Data Exchange administrators
- MQSeries administrators
- System administrators for the machine on which Tivoli Data Exchange is running

## How This Guide Is Organized

The following table lists and describes the parts and chapters in this manual.

Chapter	Title	Purpose
1	About This Guide	Provides a general introduction to this manual.
2	Tivoli Data Exchange Concepts	Provides an overview of the major components of TDE, its architecture, and how its components work.
3	Tivoli Data Exchange Interface Commands	Describes the commands used to submit and monitor data-transfer requests from a command-line interface.
4	Component Configuration Commands	Describes the commands used to start up TDE components.
5	FTF COBOL API	Describes the TDE API, which allows you to submit and monitor data-transfer requests using API calls from within business applications.
6	C API Reference	Describes the TDE API, which allows you to submit and monitor data-transfer requests using API calls from within business applications.
7	C Data Structures	Describes the data structures used with the API functions.
8	Tivoli Data Exchange and MQSeries Security	Describes the security used by TDE and MQSeries.
9	XML Integration	Describes the characteristics of XML files that can be submitted to the TDE Manager to initiate TDE functions.
10	Multi-File Connector	Describes how the Multi-File Connector can be used to transfer files.
11	The File-to-Message Connector	Describes how the File to Message connector can be used to transfer files to queues and queue messages to file.
12	FTFBPI Wrapper	Describes TDE interfacing with enableNet's Business Process Integrator (BPI) process.
Appendix A	Security Authorization Exit JCL	Contains two sets of sample JCL to be used with security authorization exits.
Appendix B	Tivoli Data Exchange GUI Environment Variables	Lists and describes the environment variables used in the TDE GUI command file.

## Contacting Customer Support

We are very interested in hearing from you about your experience with Tivoli products and documentation. We welcome your suggestions for improvements. If you have comments or suggestions about this documentation, please send e-mail to [usib2hpd@vnet.ibm.com](mailto:usib2hpd@vnet.ibm.com).

If you encounter difficulties with Tivoli products, contact Tivoli Customer Support. In the United States, the Tivoli number is 1-800-TIVOLI8 and the IBM number is 1-800-237-5511 (press or say **8** after you reach this number). Both of these numbers direct your call to the Tivoli Customer Support call center. In addition, you can enter <http://www.tivoli.com/support> to view the Tivoli Customer Support home page.

After you link to and submit the customer registration form, you will be able to access many customer support services on the Web. For support services outside the United States and Puerto Rico, contact your local IBM representative or your authorized IBM supplier.

## Where to Look for More Information

Information about MQSeries issues is available in the MQSeries documentation. The IBM corporate website includes a web page that lists the MQSeries manuals and allows you access to an online version of each manual. At publication time, this page's URL was:

*<http://www.software.ibm.com/ts/mqseries/library/manuals/>*

## Conventions

The following elements are used in this guide to make it easier to use:

---

**Note:**

Notes provide additional information about the current subject.

---

### **Warning:**

Warnings alert you to situations that can cause problems, such as the loss of data, if you do not carefully follow instructions.

### **Sidebar**

Sidebar contains information that does not fit specifically with the flow of the current topic, but is important to the topic. Sidebars are usually a short topic.

All syntax, operating system terms, and literal examples are presented in this typeface.

*Italics in a command string signify variables.*

Text enclosed in angle brackets (<>) denotes variable information. Replace the variable information with the actual value.



# **Tivoli Data Exchange Concepts**

This chapter provides an overview of the Tivoli Data Exchange (TDE) product and its architecture. It includes the following sections:

<b>Section</b>	<b>Page</b>
Introduction	15
Why FTF?	15
Architectural Overview	17

## **Introduction**

TDE (FTF) provides the means to exchange information between dissimilar networks, operating systems, databases, and applications. For developers who want to leverage the strengths of MQSeries, FTF is a high-performance solution for distributing mission-critical information throughout the enterprise. Surpassing simple data movement tools, FTF capabilities extend from basic data movement to the comprehensive management of all data-transfer activities. This results in greater productivity and enables users to integrate FTF into existing business processes with minimal impact and/or change.

## **Why FTF?**

Data-transfer mechanisms available to date have used existing, underlying network transport protocols. These products typically provide a connection-oriented, synchronous data-transfer facility. Because of the inherent nature of the transport protocols and the architecture of the products, every data-transfer request requires a session between the computers involved. Data transfers are usually limited to a point-to-point transfer of a single file or a series

of files between one source and one destination. In some cases, if a transfer fails at any point, the entire transfer session must be reestablished and started again. In addition, no facility for intelligent recovery, checkpoint restarting, load-balancing, or assured data delivery is provided in the event of network or other system failures.

Most traditional data-transfer utilities and third party products do not provide incremental and scalable solutions. For organizations with diverse business application integration requirements and network infrastructures, the traditional data-transfer utilities fall short in delivering a robust solution to solve their unique data-transfer needs today, and providing a migration path to real-time transaction-based highly distributed applications in the future.

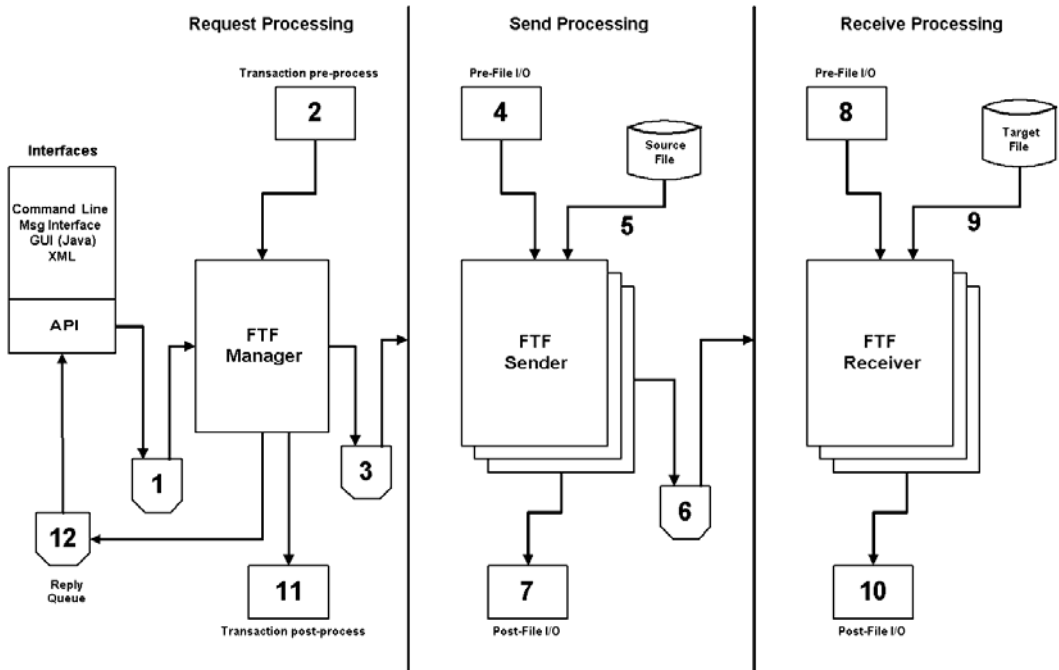
Most of the available data-transfer solutions are modeled after the UNIX style File Transfer Protocol (FTP) utility and implement a synchronous data-transfer model between a client and server. Key characteristics of these types of data-transfer implementations are:

- Connection oriented – A network session or conversation is established between the source and the destination. The underlying transport mechanism may be reliable or unreliable. Sessions are created, used, and terminated for each data-transfer request.
- Non-network transparent – By having to interact directly with the network layer protocols, most data-transfer utilities cannot isolate the source and the destination processes from the idiosyncrasies of the network layer.
- Non-modular – In most data-transfer utilities, the source and the destination processes interact directly with the file systems and are directly responsible for manipulating the data on both the source and destination.
- Limited restart and recovery – Most data-transfer utilities provide limited restart and recovery logic to recover from system and network failures. In most cases, the entire data-transfer request must be restarted from scratch in case of system and network failures.
- Limited workflow and application integration – Tools such as FTP provide limited or no work-flow and business-application integration hooks.
- Lack of centralized monitoring and administration – Providing a consistent management and administration framework for the data-transfer utility and the underlying transport mechanism is limited to few data-transfer utilities.



## Architectural Overview

It is important to understand the architecture of FTF in order to grasp the references made throughout this document to the various FTF components. The following diagram illustrates the FTF architecture.



The following major components compose FTF:

- FTF Interfaces
- FTF Manager
- FTF Sender
- FTF Receiver
- FTF Status

The flow of a data transfer occurs as follows:

1. A request is submitted to FTF by one of the supported interfaces. The request is passed to the FTF Manager's input queue.

2. After the FTF Manager accepts the request, but before processing it, the FTF Manager transaction preprocess exit can be called.
3. The FTF Manager submits the request to the FTF Sender via the FTF Sender's input queue.
4. After the FTF Sender accepts the request, but before processing the data being transferred, the FTF Sender can call the sender pre-process exit to perform application-specific processing.
5. The FTF Sender reads and transforms the data into MQSeries messages.
6. The messages that make up the data are submitted to the FTF Receiver via the FTF Receiver's input queue and data queues.
7. After processing the data, the FTF Sender can call the post-process exit to perform application-specific processing.
8. After the FTF Receiver accepts the data, but before processing it, the FTF Receiver can call the receiver pre-process exit to perform application-specific processing.
9. The FTF Receiver retrieves the data messages and processes the data accordingly.
10. After the FTF Receiver processes the data, it can call the FTF Receiver post-process exit to perform application-specific processing.
11. The FTF Manager receives all responses and ends the logical unit of work (LUW). Before ending the LUW, the FTF Manager can call the manager post-processing exit.
12. An optional response is delivered to the appropriate end-user interface indicating that the data transfer has completed.

## **FTF Interfaces**

The FTF interfaces communicate with the FTF subsystem. FTF currently supports the following interfaces:

- command-line interface
- C API
- COBOL API on OS/390
- Java graphical user interface (GUI)
- 5250 user interface on AS/400
- XML scripting

With the exception of XML scripting, the API is the lowest layer for communicating with FTF. The API places the input messages on the FTF input queue and receives replies from a predefined reply queue. The input queue is defined in the FTF configuration file.

## **FTF Manager**

The FTF Manager reads its input queue, creates log entries, submits status messages to the status queues, and manages the state of all transfers. The FTF Manager starts the transfer unit of work and ends the transfer unit of work. The processing FTF Manager is always the originating queue manager (oqm). FTF requests are stored at the FTF Manager for processing. The node on which the FTF Manager executes is considered the originating node. The request is forwarded to the appropriate FTF Sender node as defined by the required input. The message is submitted to the FTF Sender's input queue, which is defined in the FTF configuration file. If the FTF Sender is on a remote queue manager, the message destination is resolved to a transmission queue and transmitted appropriately based on the predefined MQSeries configuration.

The FTF Manager correlates all operational replies and reports the final status of the FTF transactions. These replies should not be confused with the status messages that are part of the integrated status subsystem. Status messages are independent and not required for internal FTF processing. There are four completion codes that the FTF Manager reports:

- Request completed successfully
- Request failed (accompanied with the appropriate failure information)
- Request expired
- Request canceled

---

### **Note:**

For more information about completion codes, see *Tivoli Data Exchange Messages and Codes*.

---

## **FTF Sender**

The FTF Sender reads its input queue, creates log entries, submits status messages, and transforms the data into MQSeries messages. The processing FTF Sender is always where the source data resides and is identified as the source FTF Sender or source queue manager (sqm). The FTF Sender has the capability to read the data and place it in a staging area or transmit it directly to the target

FTF Receiver. The request, and all of its associated data, is forwarded to the target FTF Receiver as defined by the required input. If the FTF Receiver is on a remote queue manager, the message destination is resolved to a transmission queue or set of transmission queues and appropriately transmitted via the MQSeries message channel agent. Immediately after completing the processing of its portion of the FTF transaction, the FTF Sender replies to the originating FTF Manager.

## **FTF Receiver**

The FTF Receiver reads its input queue, creates log entries, submits status messages, and receives incoming data from MQSeries to create the target data. The FTF Receiver is always the destination for the data and is identified as the destination FTF Receiver or destination queue manager (dqm). The FTF Receiver accepts a data-transfer request and processes the inbound data from its data queues. Immediately after processing its portion of the FTF transaction, it reconstructs the target data from the MQSeries messages and submits an operational reply to the originating FTF Manager.

## **FTF Status**

The FTF status queues are defined in the FTF configuration file. FTF Status messages, which are submitted by each of the FTF components that make up the FTF subsystem, are not required for internal FTF processing. Rather, they provide a status reporting subsystem that can be exercised by a number of different mechanisms to report on the current and past status of data-transfer requests. Status messages are MQSeries messages destined for the queue or list of queues defined in the FTF configuration file. These queues can be defined as local or remote, and can be processed and viewed with any of the supporting interfaces or user-written programs. These messages may be disabled, as FTF can operate without them.

---

**Note:**

To disable status messages, you need to change the FTF configuration file, as follows:

- Stop the FTF components.
  - Go to the FTF configuration file (ftfconfig.ini) and find the “Status Defaults” section.
  - Change status to off (Status=OFF). The default status is NOTPERSIST.
  - Restart the FTF components for the change to take effect.
-



---

---

# Tivoli Data Exchange Interface Commands

This chapter describes the commands that compose the Tivoli Data Exchange (TDE) command-line interface. It contains the following sections:

Section	Page
Overview	24
FTF	25
FTFCNCL	50
FTFPING	53
FTFSTAGE	62
Using Staging Queues	67
FTFSTAT	72
Using the FTF Publish/Subscribe Function	85
FTFBRK	86
FTFPUB	90
FTFSUB	100

## Assumptions

This chapter makes the following assumptions:

- You have a working knowledge of TDE data transfers.
- You know how to access and use a command line in the appropriate operating system.

## Overview

The commands in this chapter allow you to perform data transfers from a command line or command script. This chapter contains the following information for each command:

- Supported operating systems
- Description
- Syntax
- Argument summaries, including examples

---

### **Note:**

In many cases, these commands allow you to use a substantial number of command-line arguments. Because of the volume of the arguments, they are presented in functional groups. Unless otherwise noted, any command-line argument can be used with any other argument in the same functional group or in another functional group.

---



## **FTF**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS
- IBM 4690 OS

### **Description**

The FTF command starts a TDE data-transfer request and specifies the conditions under which it runs.

---

#### **Note:**

When a data-transfer request fails and the file mode for the TDE Receiver is set to “Create” or “Replace,” the Receiver component deletes the file. If the transfer request fails before the TDE Receiver starts processing to the target file, cleanup is not required. However, if the mode is set to “Create” or “Replace” and the TDE Receiver has started writing data to the target file when the data-transfer request fails, the TDE Receiver deletes the file.

---

#### **Note:**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

## Syntax

FTF <Queue Manager arguments> <Source and Target File arguments>  
<Environment arguments><Process arguments> <User Exit arguments>  
<Data Specification arguments><OS/400 arguments><OS/390  
arguments><Help arguments>

The following sections describe each type of command-line argument used with the FTF command:

- Queue Manager Arguments (see page 26)
- Source and Target File Arguments (see page 27)
- Standard Environment Arguments (see page 31)
- Authorization Arguments (see page 32)
- Process Arguments (see page 33)
- User Exit Arguments (see page 38)
- Data Specification Arguments (see page 39)
- OS/400 Option Arguments (see page 44)
- OS/390 Option Arguments (see page 46)
- Help Option Arguments (see page 49)

## Queue Manager Arguments

This section describes the FTF arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTF command with the queue manager arguments takes the following form:

```
FTF -lqm localQueueMgr -oqm origQueueMgr -sqm sourceQueueMgr  
-dqm destQueueMgr
```

Where:

- **-lqm *localQueueMgr*** – Determines the queue manager from which the FTF command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTF command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-oqm *origQueueMgr*** – Determines the queue manager where the TDE Manager operates. If this value is not specified, the oqm is given the same value as the lqm.

- **-sqm *sourceQueueMgr*** – Determines the queue manager where the TDE Sender operates. If this value is not specified, the sqm is given the same value as the lqm.
- **-dqm *destQueueMgr*** – Determines the queue manager where the TDE Receiver operates.

---

### **Note:**

To designate a queue manager name when using the default client queue naming convention, use the @ symbol to join the client and queue manager names, e.g.

-dqm CLIENTA@QMGR.

---

### **Example**

The following FTF command is run with an lqm and dqm of PROD16A. The oqm and sqm default to the value of lqm.

```
FTF -lqm PROD16A -dqm PROD16A
```

## **Source and Target File Arguments**

This section describes the FTF arguments used to specify source and target files. It includes a syntax model and examples for the command-line options.

The FTF command with source and target file arguments takes the following form:

FTF -spath *sourcePath* -dpath *destPath*

Where:

- **-spath *sourcePath*** – Determines the fully qualified path and filename of the source file. This argument is required unless the file is being sent from the staging queue [using the *-fromstage* option] and an FTFID is specified

- **-dpath** *destPath* – Determines the fully qualified path and filename of the destination file. This argument is required unless the transaction sends the file to the staging queues.

## **File-Naming Conventions and OS/390 Transfers**

When you transfer files to or from an OS/390 system, you must use specific data set naming conventions. A data set name that is contained within parentheses and is not accompanied by the symbols “+” or “-” is treated as a PDS member. A data set name that is contained within parentheses and is accompanied by the symbols “+” or “-” is recognized as a Generation Data Group. All file mode options (create, append, and noreplace) and transfer types (binary and text) are supported for PDS members. If the PDS specified for the destination filename does not exist, TDE automatically allocates the PDS and creates the member.

### **Example**

In the following example, the source is FTFMQ.SOURCE.RPT1016 and the destination path is FTFMQ.TARGET.RPT1016.

```
FTF -sqm PROD16A -spath FTFMQ.SOURCE.RPT1016 -dqm  
PROD16A -dpath FTFMQ.TARGET.RPT1016
```

## **Transferring One File to Multiple Destinations**

In order to provide the ability to send one file to multiple destinations with a single command, an AWK script has been designed that reads the source file location and multiple target destinations, then constructs and issues multiple FTF requests automatically. Any error that occurred during processing is saved in a return code and the error message displays after the script processes the entire input file. The script can be edited by the user to add more error checking if desired. A sample file illustrates this script.

### **Using the AWK Script**

An AWK Interpreter must be installed on your system. This is available on all UNIX platforms and UNIX toolkits for Win 32/DOS.

The script processes the source record line and saves its sqm, spath, cfile, lqm, and oqm information. For each target record line, the script retrieves its dqm, dpath, and any extra options such as type, mode, and MVS options. The script then constructs an FTF command line and issues it through a system call.

## Tivoli Data Exchange Interface Commands

### FTF

```
#-----
# Sample input file for the One-to-Many AWK Script (One2Many.AWK)
#
# This file contains 2 sections, Source Definition and Target Definition.
#
# +-----+
# | S O U R C E   D E F I N I T I O N |
# +-----+
# Only one line specifies mandatory SQM, SPATH, CFILE, optional LQM, OQM
#
# +-----+
# | T A R G E T   D E F I N I T I O N |
# +-----+
# Each line should contain at least 2 columns (DQM & DPATH), for example
#
#   TGTQMGR1    F:\D1.TXT
#   TGTQMGR2    /tmp/t.dat
#
# Optionally, any target platform-specific options can be appended at the end
# of each line in e-Adapter's FTF Command line arguments format, for example
#
#   TGTQMGR1    DATASET.PDS(MBR1) -ORG PDS -UNIT SYSDA
#   TGTQMGR2    /tmp/t.dat        -TYPE BINARY
#
# +-----+
# | N o t e |
# +-----+
#   (1) Use '#' to mark a comment line
#   (2) Use '\\' to represent '\' in the CFILE path on NT.
#   (3) This sample use Space as Field Separator. You are free to choose any
#       FS like '|' or ';'. Make sure you update One2Many.AWK on variable FS
#       initialization in the BEGIN block accordingly.
#-----

#-----
# Source Definition
#-----
# SQM      SPATH          CFILE                                LQM      OQM
#-----
SQMGR  F:\SPATH.TXT      D:\E-ADAPTER\BIN\FTFCONFIG.INI  LQMGR  OQMGR

#-----
# Target Definition
#-----
# DQM      DPATH          EXTRA_OPTIONS
#-----
TGTQMGR1  F:\D1.TXT      -TYPE TEXT
TGTQMGR2  F:\D2.TXT      -MODE NOREPLACE
TGTQMGR3  F:\D3.TXT      -MSGSIZE 1 -TYPE TEXT
#TGTQMGR4  F:\D4.TXT      -DBCS -MODE APPEND
#TGTQMGR5  F:\D5.TXT      -DTYPE DATAMAP
```

To start the one-to-many transfer using the sample input file, issue the following command:

```
awk -f One2Many.awk One2Many.in
```

The source file F:\SPATH.TXT on SQMGR will be transferred to three destinations: F:\D1.TXT on TGTQMGR1, F:\D2.TXT on TGTQMGR2, and F:\D3.TXT on TGTQMGR3.

## Standard Environment Arguments

This section describes the standard environment arguments used to direct how the FTF command operates. It includes a syntax model and examples for the command-line options.

The FTF command with standard TDE environment arguments takes the following form:

FTF -cfile *configFile* -cq *configQueueName* -ofile *optionsFile* -version

Where:

- **-cfile** *configFile* – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.
- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a cfile and a cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.

### Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, TDE checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, TDE checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

## Tivoli Data Exchange Interface Commands

### *FTF*

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTF command. In the options file, you can set any of the command-line arguments that can be set for the FTF command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current TDE version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

### Example

The three following FTF command examples use values from the configuration file, configuration queue, or the options file. These arguments are used with other arguments in a command.

```
FTF -cfile C:\PRODCFG.FLE
```

.

```
FTF -cq PRODCONFIGQ
```

.

```
FTF -ofile PROD16A
```

The following example causes the TDE version to display:

```
FTF -version
```

## Authorization Arguments

**\*\*ADD oid opw sid spw did dpw\*\***



## Process Arguments

This section describes the process arguments used to direct how the FTF command operates. It includes a syntax model and examples for the command-line options.

The FTF command with process arguments takes the following form:

```
FTF -cancelmode cancelMode -delsrc -expiry expDuration -id1
identifierText1 -id2 identifierText2 -id3 identifierText3 -immed -label
labelVal -msgsize sizeValue -priority priorityValue -replyQ replyQueue
-replyqmgr replyQueueMgr -stage -stageonly -stagepersist -fromstage -ftfid
idValue -tranpersist -trusted -wait waitSecs
```

Where:

- **-cancelmode** *cancelMode* – Provides a command-line override to the preemptive cancel flag in the TDE configuration file.

**Valid values:** ON, OFF

If you specify -cancelmode ON, TDE looks for a cancel message between the processing of each data block. FTF cancels the transaction during processing. This type of cancellation is known as *preemptive cancellation*. If you specify -cancelmode OFF, TDE does not look for a cancel message between the processing of each data block. By choosing -cancelmode OFF, you increase the efficiency of the data transfer but you cannot cancel the transaction after processing begins.

If you do not supply the -cancelmode option, the default is the value set in the corresponding preemptive cancel flag that is specified in the TDE configuration file (ReceiverCancel and SenderCancel).

However, the data transfer can still be cancelled if the cancel message is received before the processing of the transaction begins and the -cancelmode is set to OFF or if the preemptive cancel flags in the TDE configuration file are set to NO.

- **-delsrc** – Indicates that the source data is to be deleted once the data-transfer request is completed.
- **-expiry** *expDuration* – Determines the time period, measured in minutes, after which the data-transfer request expires. If the expiration duration is exceeded, the request is terminated and the FTFRCI\_REQUEST\_EXPIRED message is returned. If the expiration occurs partway through a request, the request is marked as expired.

- **-id1 *identifierText1*** – Designates the first user-defined field that is associated with a data transfer. The *identifierText1* field can contain any value. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **-id2 *identifierText2*** – Designates the second user-defined field that is associated with a data transfer. The *identifierText2* field can contain any value. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **-id3 *identifierText3*** – Designates the third user-defined field that is associated with a data transfer. The *identifierText3* field can contain any value. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **-immed** – Gives you the ability to issue a TDE data-transfer request that is processed synchronously between the Sender and Receiver rather than the normal asynchronous mode. Using the immediate option, the TDE Receiver begins to process the data transfer request as soon as it gets the first data message instead of waiting for all of the data to arrive. Since the receiver processes the data messages as they arrive, the queue storage required to transfer a large amount of data is significantly reduced. Normal TDE data transfers require that the Receiver have all related data messages before writing the data to the disk. For example, using TDE in normal mode processing very large data transfers such as 1 gigabyte, the receiving node needs to have 1 gigabyte for queue storage and 1 gigabyte to store the data. With FTF immediate, the storage space on the queue storage is not required.

### Immediate Works with Connectors

The connector library accepts the immediate option. Immediate file transfers and immediate connector transfers both operate in the same manner.

## Considerations with Immediate

An immediate transfer is a synchronous transaction. If either the Sender or the Receiver goes down or if the connection between the Sender and Receiver is lost during processing, the transaction will timeout and fail. For example, if the Sender is processing an immediate transfer and the Receiver is down, the Sender times out and fails the transaction. Likewise, if the Sender is processing an immediate transfer and goes down, then the Receiver times out and fails the transaction.

If the receiver starts processing an immediate transfer and then goes down, it will not be able to recover the request. The request fails. The Sender, on the other hand, can recover if it goes down during processing of an immediate transfer as long as it is restarted before the Receiver times out and fails the transfer.

### Note:

When either the Sender or the Receiver times out and fails the transaction, a cancel message is sent to notify the other side to stop processing and to clean up any messages. This is when the failed transaction appears to be cancelled. You should look at the detailed status for the transaction to determine the reason for failure. The status logs reflect two transactions:

1. A failed transaction message when the Sender recognized that the Receiver was not responding.
2. A cancelled transaction message when the Receiver comes back up and recognizes the cancellation message from the Sender.

- **-label** *labelVal* – Specifies the user-defined label. This value allows you to assign arbitrary labels to data transfers to allow for status queries. Each label can be up to 20 bytes in length.
- **-msgsize** *sizeValue* – Allows you to set a message size value to override the MQSeries message size value. **Valid values:** 1-3906 KB (3.9 MB)  
**Default value:** 512

- **-priority** *priorityValue* – Determines the priority applied to the data-transfer request. **Valid values:** 1 (highest) – 5 (lowest) **Default value:** 5
- **-stage** – Enables the data messages that make up the data being transferred to be staged in queues and remain there after the data-transfer transaction has ended. Staging is appropriate if the data being transferred does not permanently exist on the sender. It allows the transaction to resend the message from the staging area. Since data messages in the staging area are not automatically removed, they can be deleted by using the FTFSTAGE command (see page 62).

To support remote staging, the stage queues, FTFSDR.STAGE and FTFSDR.STAGE.CONTROL, are defined as remote queues that resolve to a local queue on another queue manager. The source file then stages at the remote queue manager. To send the files in the remote stage queues, define the sqm on the request as the queue manager where the file is actually staged. The local TDE Sender, however, cannot purge or query the remotely staged files. These functions must be performed by the TDE Sender on the remote queue manager. If purge or query is attempted by the local TDE Sender, the Sender will report a failure attempting to open the remote queues.

- **-stageonly** – Places the data message on the staging queue, but does not send it. If you specify this value, you cannot specify a dqm or dpath.
- **-stagepersist** – Specifies that the messages in the staging area are persistent. If you specify this argument, the messages still exist after a system reboot or TDE shutdown. If you select this argument, you increase the recovery ability of TDE, but reduce performance.
- **-fromstage** – Indicates that the source file specified in the *-spath* or *-ftfid* argument should be retrieved from the staging queue rather than from the actual source file.
- **-ftfid** *idValue* – Indicates the FTFID of the transaction that should be sent from the staging queue. You should not use this option unless you are sending a file from the staging queue using the *-fromstage* option, and you have not specified a source filename.
- **-tranpersist** – Specifies that the data being transferred is persistent. If you specify this argument, the data still exists after a system reboot or TDE shutdown. If you select this argument, you increase the recovery ability of TDE, but reduce performance.

- **-trusted** – Sacrifices the ability to recover in order to allow for greater performance. In a trusted transaction, no file recovery is possible. Specifying this argument invokes the TDE trusted option, not the MQSeries trusted option.
- **-wait** *waitSecs* – Contains the amount of time, in seconds, to wait for a reply. This argument indicates that the FTF command waits for a response from the TDE Manager to indicate whether a request has succeeded or failed. If the TDE Manager does not respond within the specified time period the command times out.

## Example

In this example, the label of *data update query* and the wait value of 600 seconds (10 minutes) are assigned. The file sent from the staging area has an FTFID of de679301-8eaa-11d2-af00-ac2f955ce9c8 and the preemptive cancellation flag is activated. Required command-line options not listed are specified in the options file.

```
FTF -label "data update query" -wait 600 -fromstage  
-ofile FTFMQ.FTF.OPT  
-ftfid de679301-8eaa-11d2-af00-ac2f955ce9c8  
-cancelmode ON
```

The following FTF command is used to send a file to the staging queue, but not to a TDE Receiver. Required command-line options not listed are specified in the options file.

```
FTF -ofile FTFMQ.FTF.OPT -stageonly
```

The identifier arguments can be specified at the command line along with the data transfer request. If the identifier text value contains spaces, the value needs to be in quotes. The following example illustrates using the identifier arguments:

```
FTF -spath E:\finance\status.rpt -id1 "October 1999  
report" -id2 monthlystatus
```

The value “October 1999 report” is stored in the id1 field and the value “monthlystatus” is stored in id2. Both fields are carried with the data transfer request. The FTFSTAT command shows both values as part of its display.

## User Exit Arguments

This section describes the FTF arguments used to specify user exit arguments. It contains syntax information and an example. For more information about user exits see “Tivoli Data Exchange User Exits” in the *Tivoli Data Exchange User’s Guide*.

The FTF command with user exit arguments takes the following form:

```
FTF -exit exitNo -exitdll dllName -exitentry entryPoint [-exitdata  
dataValue]
```

Where:

**-exit** *exitNo* – Determines the exit number to be invoked. **Valid values:** 3-8, 9-10 (connectors)

---

- **Note:**

- If you are running connectors on a Solaris 2.5.1 operating system, you must install Solaris Patch 103627.
- 

**-exitdll** *dllName* – Determines the DLL, shared object, or load module used to invoke the exit module.

**-exitentry** *entryPoint* – Contains the name of the function in the DLL that contains the exit module.

**-exitdata** *dataValue* – Contains the command-line argument to execute when you invoke a user exit that requires input parameters.

---

**Note:**

You must keep the exit arguments together and in order on the command line. If you specify an argument that is not associated with the current exit, the command generates an error when you run it.

If you specify an -exitdata argument that is not adjacent to the exit’s other arguments specified, the exit data is ignored.

---

## Example

The following FTF command invokes the sample user exit for exit 7 to run Notepad.

```
FTF -exit 7 -exitdll FTFEX78 -exitentry LoadParms
    -exitdata "c:\winnt\system32\notepad.exe"
```

## Data Specification Arguments

This section describes the FTF arguments used to establish data specification arguments. It contains syntax information and an example.

The FTF command with data specification arguments takes the following form:

```
FTF -compress -DBCS -ddata destData -dtype destType -mkdirs -mode
modeType -notifystat notifyStatus -notifytype notifyType -notifydata
notifyData -padchar padCharacter -pool poolName -recpad recordPad
-recwrap wrapMode -replyq replyQueue -replyqmgr replyQueueMgr -sdata
sourceData -stype sourceType -type fileType
```

Where:

- **-compress** – Specifies that the data being sent is compressed using the internal TDE compression algorithm.
- 
- Note:**
- This compression utility is not supported on the 4690 platform.
  - Compression for TDE V1.2 is not compatible with previous versions.
- 
- **-DBCS** – Specifies the option to scan the source file and split each datablock at the point where double-byte validity is maintained. Using this option ensures that there is no truncation of a double-byte character when the specified message size does not span the entire character.

### Note:

If the DBCS option is used with the wrap option in a data-transfer request, the results can be unpredictable.

- **-ddata** *destData* – Determines the data output for a transfer that receives data that is not stored in a file.

## **Tape Input and Output**

In an OS/390 environment, magnetic tape can be used in your data transfers. You can specify that a data transfer be read from or written to magnetic tape. For output, use the unit argument and specify the name of the tape device defined in the system configuration. For input, use the name of the file on tape and the system finds the file and brings it into TDE.

- **-dtype** *destType* – Determines the data type for the destination data. You should only use this argument to handle destination data that is not stored in a file. This value must match a data type specified in the TDE configuration file and it is case sensitive.

You can also reference a datamap name in the **-dtype** argument of the FTF command. Specify which datamap you want to use when issuing the data transfer request as in the following example:

```
FTF -spath fullfile.txt -dtype MyMap
```

See the “TDE Configuration” chapter of the TDE *Installation Guide* for information describing how to formulate a conversion map.

- **-makedirs** – Creates the directories required to support the **destPath** value. If this argument is not specified and the specified directory does not exist, the data-transfer request fails with a FILE OPEN ERROR. Applies to Windows-based and UNIX systems only.
- **-mode** *modeType* – Determines what occurs when the data is written to the target. Although the default value for this argument is *create*, the only values you can specify from the command line are *append* and *noreplace*. Do not use this argument unless you want to override the default *create* setting. **Valid values:** APPEND, NOREPLACE. **Default value:** CREATE.



- **-notifyStatus** – Defines when a notification message will be sent to the NotifyQueue (see *Tivoli Data Exchange Installation Guide*, Chapter 9, “Tivoli Data Exchange Configuration,” Notification Message Property, for more information). The notification is sent if the transaction’s status matches the status specified in this argument. If you specify this argument, you must also specify *-notifyData* and *-notifyType* arguments. **Valid values:** Success, Failure, Nonsuccess (includes failed, cancelled, expired)

---

**Note:**

TDE functions only deliver the notification messages; any further processing is the responsibility of the user.

---

- **-notifyType** – Specifies the user-defined method, such as EMAIL, PAGER, FAX, or WTO, that will be used to deliver a notification message based on a transaction’s status. If you specify this argument, you must also specify *-notifyData* and *-notifyStatus* arguments.
- **-notifyData** – Specifies user-defined data to aid in notification. The data supplied with this argument is placed in the notification message. If you specify this argument, you must also specify *-notifyStatus* and *-notifyType* arguments.
- **-padchar** *padCharacter* – If padding of target file records is elected by entering the argument of *recpad=pad*, the *padCharacter* indicates what hexadecimal character is to be used for padding. If left out of the command and padding is elected, blank is assumed to be the padding character.
- **-pool** *poolName* – Is the name of the data pool used for transferring data from the TDE Sender to the TDE Receiver. If this value is not specified, the default pool specified in the TDE configuration file is used. For this option to function, the specified pool must be defined in the TDE configuration file.

- **-recpad** *recordPad* – Enables or disables the padding facility. TDE provides the following options:
  - **nopad** – Specifies that blanks are not to be inserted in each record to fill it out to the length of the other records in the file.
  - **pad** – Specifies that blanks are to be inserted in each record to fill it out to the length of the other records in the file.

### **How Padding Works**

TDE padding functionality is governed at the source node (the TDE Sender) and by the characteristics of the data being transferred. No padding decisions are made at the target node (the TDE Receiver). If you request a data transfer in which the source and target nodes both run under operating systems that support record-formatted I/O (such as OS/390), using padding may cause undesired results if you specify different record lengths for your source and target files.

For example, if you send an OS/390 source file with a record length of 80 bytes to an OS/390 target file with a record length of 100 bytes, and if you enable padding, the target file's records are padded up to 80 bytes and contain blanks thereafter.

In addition, if you use the wrap option and the padding option, the padded characters will wrap if the record length of the target file is less than the source file's record length. You can specify padding in the FTF command, the TDE configuration file, and the ISPF panels.

- **-recwrap** *wrapMode* – Indicates how records will be processed when they reach the target file and the records are longer than the target record length. TDE provides the following options:
  - **WRAP** - wraps records that are of greater length than the target file record length.
  - **TRUNC** - truncates records up to the record length of the target file.
  - **FAIL** - fails the data-transfer request when the record length exceeds the maximum allowed for the target file.

- **-replyq** *replyQueue* – Names the queue to which the reply message is to be routed.
- **replyqmgr** *replyQueueMgr* – Names the queue manager to which the reply message is to be routed.
- **-sdata** *sourceData* – Determines the data input for a transfer that sends data that is not stored in a file.
- **-stype** *sourceType* – Determines the data type for the source data. You should only use this argument to handle source data that is not stored in a file. This value must match a data type specified in the TDE configuration file, and it is case sensitive.
- **-type** *fileType* – Determines whether the file is text or binary. **Valid values:** text, binary **Default value:** binary.

## Example

In this example, the values listed in the following table must be set in the FTF command.

Value	Setting
Trusted	Yes
Compressed	Yes
Staged	Yes
Persistent Transfer Files	Yes
Persistent Staged Files	Yes
Message Size	3906
Priority	1
Expiry	10 minutes
File Type	Binary

The following FTF command requests a data transfer with the specified settings. Required command-line options not listed are specified in the options file.

```
FTF -ofile FTFMQ.FTF.OPT -trusted -compress -stage
-tranpersist -stagepersist -msgsize 3906 -priority
1 -expiry 10 -type binary
```

## Tivoli Data Exchange Interface Commands

### FTF

In this example, data is sent from an OS/400 physical file to another OS/400 physical file.

```
FTF -ofile c:\ftf\FTF.OPT -stype AS400PF
      -dtype AS400PF
      -sdata "TBL=PROD/WKLYSALE;RCDFORMAT=RFMT1;DLM=,"
      -ddata "TBL=PROD/WKLYSALE;RCDFORMAT=RFMT1;DLM=,"
```

## OS/400 Arguments

This section describes the FTF arguments used to specify OS/400 arguments. It contains syntax information and examples. The default value for each argument is the value set in the AS400 DEFAULTS portion of the TDE configuration file. You can use multiple queue managers. Each instance of TDE must be installed against a single queue manager, but you can install multiple instances of TDE against different queue managers.

The FTF command with OS/400 arguments takes the following form:

```
FTF -as400ft fileType -ccsid codedCharSetId -crtlib createLib -fileasp
fileAuxStoragePool -filetxt fileText -libasp libAuxStoragePool -libtxt libText
      -rcdlen recordLength
```

Where:

- **-as400ft** *fileType* – The value entered specifies the type of file. **Valid values:** \*DFLT (defaults to the value entered in the configuration table) \*SAVE (specifies an OS/400 Save file) \*SRCPF (specifies an OS/400 source physical file).
- **-crtlib** *createLib* – Specifies that TDE is to create the specified library if it does not exist. **Valid Values:** YES, NO.
- **-ccsid** *codedCharSetId*– The CCSID is used as the identifier for the data-transfer request. If CCSID is not specified, TDE uses the CCSID of the job. **Valid values:** 1-65535
- **-fileasp** *fileAuxStoragePool* – Specifies the File Auxiliary Pool for a file that TDE creates for a data-transfer request. **Valid values:** 1-16
- **-filetxt** *fileText* – Specifies the file description for a library that TDE creates for a data-transfer request.

- **-libasp** *libAuxStoragePool* – Specifies the Library Auxiliary Pool for a library that TDE creates for a data-transfer request. **Valid values:** 1-16
- **-libtxt** *libText* – Specifies the library description for a library that TDE creates for a data-transfer request.
- **-rcdlen** *recordLength* – Determines the record length for the target file on OS/400. **Valid values:** 13-32766

---

**Note:**

On the OS/400, if the record length of the data being sent is over 80 bytes then the RCDLEN parameter should always be 12 greater than the actual record length. If you send a file with lrecl=1024 then you need to set rcdlen=1036.

---

**Example**

In this example, the values listed in the following table must be set in the FTF command.

Value	Setting
AS/400 File Type	*DFLT
Create Library	YES
Library Aux Storage Pool	1
File Aux Storage Pool	1
Library Text	TARGET LIBRARY
File Text	TARGET FILE
CCSID	129
Record Length	255

The following FTF command requests a data transfer with the specified settings. Required command-line options not listed are specified in the options file.

```
FTF -ofile FTFMQ.FTF.OPT -as400ft *DFLT -crtlib YES -libasp 1
-fileasp 1-libtxt TARGET LIBRARY -filetxt TARGET FILE -ccsid 129
-rcdlen 255
```

## OS/390 Arguments

This section describes the FTF arguments used to specify OS/390 arguments. It contains syntax information and examples. The default value for each argument is the value set in the MVS DEFAULTS portion of the TDE configuration file.

### Specifying an Esoteric Unit Name

To specify an esoteric name for the OS/390 UNIT value, follow these steps:

- Do not set a value in the MVSVOLUME stanza in the TDE configuration file.
- Specify the unit value in the **-unit** argument or in the MVSUNITNAME stanza in the TDE configuration file on either the TDE Sender or the TDE Receiver.

The FTF command with OS/390 arguments takes the following form:

```
FTF -alcunit allocUnit -blksize blockSize -bufno numberBuffers -dirblks  
numBlks -lrecl logRecLength -model GDGName -org fileOrg -primary  
primAlloc -recfmt recordFormat -secondary secAlloc -unit unitName -volser  
serialNumber
```

Where:

- **-alcunit** *allocUnit* – Determines the allocation unit used for the target on OS/390. **Valid values:** CYL (cylinder), BLK (block), and TRK (track).
- **-blksize** *blockSize* – Determines the block size for the target file on OS/390. Specifying a block size of 0 enables the system to choose the optimum block size for the dataset during allocation. If the record format is Fixed Block (FB), the block size in the blksize argument must be a multiple of the logical record length, the lrecl parameter. When the record format is Variable Block (VB), the blksize value must be at least four bytes greater than the lrecl value. **Valid values:** 0 - 32760
- **-bufno** *numberBuffers* – Allows you to specify the number of internal buffers that are to be used when processing data transfers. The throughput of a TDE data transfer is governed by a combination of the block size of the data being transferred and the number of buffers that are allocated for transfer in the bufno argument. **Valid Values:** 1 - 255

- **-dirblks** *numBlks* – Sets up the number directory blocks that is used to allocate the target PDS if it does not exist. If this argument is not specified, the value in the configuration file is used. If neither is specified, the PDS allocation will fail. **Valid values:** 1-32760
- **-lrecl** *logRecLength* – Determines the logical record length for the target file on OS/390. If the value for recfmt is V or VB then the value for lrecl should be 4 bytes greater than the longest data record. **Valid values:** 1-32760.

---

### Note:

You need to be aware of the relationship of lrecl and blksize values depending on the record format, recfmt argument. The following table lists how the values need to relate depending on the record format.

---

Recfmt Value	Lrecl and Blksize Relationship
F	Lrecl must be equal to blksize
FB	If blksize is not equal 0 then BLKSIZE must be a multiple of LRECL
V	Blksize must be equal to lrecl+4. This will allow for the block descriptor word.
VB	If blksize not equal 0, then Lrecl must be no more than blksize-4. This will allow for the block descriptor word

- **-model** *GDGName* – Indicates a model dataset for Generation Data Group (GDG) allocation. Consult your OS/390 Systems Administrator for the available model datasets.
- **-org** *fileOrg* – Determines the file organization of the target file on OS/390. This argument is not required for a pre-allocated data set. **Valid values:** Physical Sequential (PS), Partitioned Data Set (PDS).

---

### Note:

TDE now requires the data set organization value in a transfer (PS, PDS) to match the dsorg value for the target pre-allocated data set. If these values do not match, the transfer fails with the following error code:  
FTFRCE\_INVALID\_FILEORG.

---

## Tivoli Data Exchange Interface Commands

### FTF

- **-primary** *primAlloc* – Determines the number of primary allocation units required on OS/390.
- **-recfmt** *recordFormat* – Determines the record format for the target file on OS/390. **Valid values:** F (fixed), V (variable), FB (fixed block), and VB (variable block).
- **-secondary** *secAlloc* – Determines the number of secondary allocation units required on OS/390.
- **-unit** *unitName* – Determines the unit name for the target file on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.
- **-volser** *serialNumber* – Determines the volume serial number for the target on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.

### Example

In this example, the values listed in the following table must be set in the FTF command.

Value	Setting
Data Set Organization	Physical Sequential
Record Format	Fixed Block
Logical Record Length	255
Block Size	25,500
Unit Name	SYSALLDA
Volume Serial Number	TECH01
Allocation Unit	TRK
Primary Allocation	15
Secondary Allocation	30
Directory Blocks	0



---

**Note:**

If a data path has a member name, the file organization setting is physical sequential, and the directory block has a value other than 0, TDE overrides the physical sequential setting and sets the file organization as a partitioned data set.

---

The following FTF command requests a data transfer with the specified settings. Required command-line options not listed are specified in the options file.

```
FTF -ofile FTFMQ.FTF.OPT -dpath FTFMQ.TARGET.RPT1016  
-org PS -recfmt FB -lrecl 255 -blksize 25500 -unit  
SYSALLDA -volser TECH01 -alcunit TRK -primary 15  
-secondary 30 -dirblks 0
```

## Help Arguments

This section describes the help TDE arguments used with the FTF command. It contains syntax information and examples.

The FTF command with standard TDE arguments takes the following form:

```
FTF [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTF command-line arguments. No other commands are executed if this argument is used with other arguments.

## Examples

The following FTF command displays the FTF command-line arguments.

```
FTF -h
```

## **FTFCNCL**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS

### **Description**

The FTFCNCL command cancels the specified data-transfer request. Each TDE component can receive and cancel a data-transfer request at any time before or during the data-transfer transaction. When the appropriate TDE component receives the cancellation request, the transaction is immediately stopped and purged. If this command is issued for a transfer that has already finished, it is ignored.

### **Syntax**

FTFCNCL -cfile *configFile* -cq *configQueueName* -ftfid *idValue* -lqm *localQueueMgr* -ofile *optionsFile* -oqm *origQueueMgr* -version [-help | -h | -?]

Where:

- **-cfile** *configFile* – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.

- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a cfile and a cq argument in the same command.

## **Configuration File and Queue Order of Precedence**

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, TDE checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, TDE checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ftfid** *idValue* – Must contain the unique alphanumeric identifier of the data transfer being cancelled. You can find this value by looking at the output from the corresponding FTF command, by looking in the log files, or by using the FTFSTAT command.
- **-lqm** *localQueueMgr* – Determines the queue manager from which the FTFCNCL command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFCNCL command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFCNCL command. In the options file, you can set any of the command-line arguments that can be set for the FTFCNCL command. Any values specified on the command line override the values in the options file.
- **-oqm** *origQueueMgr* – Determines the queue manager where the TDE Manager operates. If this value is not specified, the oqm is given the same value as the lqm.
- **-version** – Returns the current TDE version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

- **-help, -h, or -?** – Displays a list and description of the FTF command-line arguments.

### Example

The following FTFCNCL command cancels a data-transfer request, as identified by its FTFID. The local queue manager and originating queue manager are PRD1MQM.

```
FTFCNCL -lqm PRD1MQM -oqm PRD1MQM -cfile  
FTFMQ.FTFCONFIG.INI -ofile FTFMQ.FTF.OPT  
-ftfid de679301-8eaa-11d2-af00-ac2f955ce9c8
```

## Additional Information

You can submit a cancel request against any TDE transaction by providing a TDE transaction identifier which correlates to the transaction that is to be cancelled. If the transaction is not found, the TDE Manager generates a warning that the transaction in question is not found and the cancel operation is complete.

If the transaction is found and pending, the TDE Manager can communicate with the source TDE Sender or with the target TDEReceiver, depending on the state of the data transfer. If preemptive cancellation is specified in the configuration file, the component that is currently processing the data transfer (either the TDE Sender or the TDE Receiver) stops the transfer's execution. The TDE Manager immediately flags the TDE transaction as cancelled, but waits for a reply from both the TDE Sender and/or TDE Receiver before the cancel operation is complete.

If a preemptive cancellation is not specified and the data transfer is in progress, the data transfer is terminated after the TDE component is done reading or writing the data. Preemptive cancellation allows for quicker cancellations, but slows the data transfer's execution. You should consider using preemptive cancellation when you are transferring large amounts of data with the potential for error.

## **FTFPING**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- IBM 4690 OS
- OpenVMS

### **Description**

The FTFPING command allows you to ping specified TDE components. The ping message starts at the lqm, the machine from which you are currently working. It is sent to the TDE Manager, as identified by the oqm. The TDE Manager sends it to the TDE Sender, as identified by the sqm. The TDE Sender sends it to the TDE Receiver, as identified by the dqm. The TDE Receiver then sends an acknowledgment to the TDE Manager. On OS/390, the acknowledgment appears in the ISPF result set when the ping operation is complete.

By using the FTFPING command, you can ensure that all TDE components involved in a data transfer are available before you start the transfer. You can also use FTFPING with the message size setting to test various message size values until you find the optimal message size setting.

---

#### **Note:**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### **Syntax**

FTFPING <queue manager node arguments> <process-control arguments>  
<standard arguments><help options>

## Queue Manager Node Arguments

Queue manager node arguments allow you to specify the queue and node to be pinged.

The FTFPING command with queue manager node arguments takes the following form:

```
FTFPING -lqm localQueueMgr -oqm origQueueMgr -sqm  
sourceQueueMgr -dqm destQueueMgr
```

Where:

- **-lqm *localQueueMgr*** – Determines the queue manager from which the FTFPING command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFPING command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (*lqm*) specified.
- **-oqm *origQueueMgr*** – Determines the queue manager where the TDE Manager operates. If this value is not specified, the oqm is given the same value as the *lqm*.
- **-sqm *sourceQueueMgr*** – Determines the queue manager where the TDE Sender operates. If this value is not specified, the sqm is given the same value as the *lqm*.
- **-dqm *destQueueMgr*** – Determines the queue manager where the TDE Receiver operates.

### Example

The following FTFPING command is issued from the queue manager called ADMIN1. The TDE Manager running on PROD11A is pinged. If it is running, it sends the ping to the TDE Sender running on PROD22. If it is running, it sends the ping to the TDE Receiver on PROD09B. If it is running, it sends an FTFPING response back to ADMIN1.

```
FTFPING -lqm ADMIN1 -oqm PROD11A -sqm PROD22 -dqm  
PROD09B
```

## Process-Control Arguments

Process-control arguments govern the conditions under which the FTFPING command runs.

The FTFPING command with process-control arguments takes the following form:

FTFPING -msgsize *sizeValue* -priority *priorityValue* -timeout *expireVal*

Where:

- **-msgsize *sizeValue*** – Allows you to set the size of the ping message. FTFPING computes the size of the message sent by multiplying the number specified by 1024 and adding 310 bytes. As an example, if 10 is entered. The message size sent would be (10 X 1024) + 310 bytes or a value of 10550. **Valid values:** 1-1023 **Default value:** 310 Bytes
- **-priority *priorityValue*** – Determines the priority applied to the ping request. **Valid values:** 1 (highest) – 5 (lowest) **Default value:** 5
- **-timeout *expireVal*** – Determines the amount of time until the FTFPING command times out. If the time limit is exceeded, the ping operation is terminated and an error message is generated. This value represented is measured in seconds. **Valid values:** 1-32767

### Example

The following FTFPING command pings the same components as in the previous example. The ping operation times out after 60 seconds. Its message size is 20 KB and its priority value is 1 (the highest priority).

```
FTFPING -lqm ADMIN1 -oqm PROD11A -sqm PROD22 -dqm  
PROD09B -timeout 60 -msgsize 20 -priority 1
```

## Standard Environment Arguments

This section describes the standard environment arguments used with the FTFPING command. It contains syntax information and an example.

The FTFPING command with standard TDE environment arguments takes the following form:

FTFPING -cfile *configFile* -cq *configQueueName* -ofile *optionsFile*  
-version

Where:

- **-cfile** *configFile* – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.
- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a cfile and a cq argument in the same command. Use the FTFCGF command to populate the queue with the configuration information.

### Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, TDE checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, TDE checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFPING command. In the options file, you can set any of the command-line arguments that can be set for the FTFPING command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current TDE version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.



## Examples

The following FTFPING command pings the components specified in the previous example. The FTFMQ.FTFCONFIG.INI configuration file and the FTFMQ.FTF.OPT options file are used.

```
FTFPING -lqm ADMIN1 -oqm PROD11A -sqm PROD22 -dqm  
PROD09B -cfile FTFMQ.FTFCONFIG.INI -ofile  
FTFMQ.FTF.OPT
```

The following FTFPING command displays the version, release, and patch number for the current TDE software.

```
FTFPING -version
```

## Help Arguments

This section describes the standard TDE arguments used with the FTFPING command. It contains syntax information and an example.

The FTFPING command with standard TDE arguments takes the following form:

```
FTFPING [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFPING command-line arguments.

## Examples

The following FTFPING command displays the FTFPING command-line arguments.

```
FTFPING -h
```

## **Additional Information**

FTFPING determines if all the components required to complete a data transfer are operational and awaiting requests. In addition, FTFPING reports round-trip times of the ping message. If a ping request is unsuccessful, the interface (command line, API, etc) times out. Continued FTFPING failures may indicate one of the following conditions:

- The FTFPING request might have been submitted with a time-out value that is too short.
- A problem might exist with the TDE or MQSeries configuration.

FTFPING submits a virtual data transfer request that exercises all components required to perform a data transfer. To ensure all components on a local node are operating, submit FTFPING with no parameters or the *lqm* and *cfile* arguments which are required on OS/390. The TDE Manager, TDE Sender, and TDE Receiver services are exercised on the local node. FTFPING reports round trip times for the ping message. Issuing the FTFPING request with no arguments, or just the *-lqm* argument, is the equivalent of issuing the following:

```
ftfping -lqm queueMgrName -oqm queueMgrName -sqm  
queueMgrName -dqm queueMgrName
```

Where *queueMgrName* is the name of the local queue manager.

Each FTFPING request exercises a specific data-transfer path. To verify a data transfer's route, use the FTFPING command with the *-sqm* and *-dqm* parameters. For example, to ensure that data can be sent from node A to node B, issue the following command from node A:

```
ftfping -dqm B
```

The results of a successful FTFPING include the ping message's path, the number of bytes transmitted, and the time to process the ping message.

For example:

```
LQM->OQM->SQM->DQM bytes=310 time<# secs
```

Actual output is as follows:

```
ftfping -lqm MQM1 -sqm CSQ1 -dqm MQS1
```

```
MQM1->MQM1->CSQ1->MQS1 bytes=310 time<1  
MQM1->MQM1->CSQ1->MQS1 bytes=310 time<0  
MQM1->MQM1->CSQ1->MQS1 bytes=310 time<0  
MQM1->MQM1->CSQ1->MQS1 bytes=310 time<1  
MQM1->MQM1->CSQ1->MQS1 bytes=310 time<0
```

If the FTFPING was unsuccessful, the following message may display:

```
FTF Ping timed out
```

If the FTFPING fails, please note the following items:

1. FTFPING connects to the local queue manager (either the queue manager specified or the default queue manager).
2. A request is submitted to the originating TDE Manager (where this request will be managed and tracked). The originating queue manager is the -oqm parameter. If -oqm is not specified, it is assumed that the originating manager is operating on the lqm. Therefore, the -oqm parameter is optional.
3. The TDE Manager processes the ping by submitting a send request to the sending node. This is specified with the -sqm parameter. This is where the source file would reside. When using the FTFPING command, this parameter is optional and if not specified, it is assumed that the sending node to be exercised is on the local queue manager (the -lqm parameter).
4. The TDE Sender accepts and processes the ping by submitting a receive request to the receiving node. This is specified with the -dqm parameter. When using the FTFPING command, the -dqm parameter is optional and if not specified, it is assumed that the receiving node to be exercised is on the local queue manager (the -lqm parameter).

## **Troubleshooting FTFPING**

If FTFPING does not complete successfully, any data-transfer request that uses the nodes specified for FTFPING fails. Troubleshooting FTFPING ensures that all data-transfer requests that use the same path complete successfully. In the event of an FTFPING timeout, check the time-out value passed to FTFPING. The default time-out value is three seconds, but you can increase it. After adjusting the time-out value, if FTFPING continues to timeout, perform the following steps on all nodes used in the FTFPING command:

1. Verify that the queue managers are running on each node specified by the FTFPING command. To verify, use the **runmqsc** command on UNIX and Win 32 or use the **CSQUTIL** command on OS/390.

For example, the response to the **runmqsc qMgrName** should be, “Starting MQSeries commands”. If the response is, “MQSeries Queue Manager not available”, start the queue manager.

The response to **strmqm qMgrName** should be, “MQSeries Queue Manager started”.

2. Verify the queue manager’s configurations on each node. The queue managers must have all of the queue definitions required for TDE and the channel definitions required for communication between the nodes.

The FTF.MQS file creates the queues needed by the TDE subsystem. Check that the FTF.MQS file was applied to each queue manager by running the display queue (FTF\*) command within runmqsc and verifying that queues begin with the FTF notation.

Check that the channels have been correctly defined between the queue managers and that the transmission queues have been correctly defined on each queue manager.

3. Verify that all of the required MQSeries objects are running on all the nodes.

Ensure that the MQSeries listener process is running. On UNIX, use the **ps** command. On Win 32, use the Task Manager. Start the listener if it is not running.

For example, the response to **runmqslr -t tcp -p port** should be “Channel program started”.

4. Check that the MQSeries channel initiator is running. For example, on UNIX and Win 32, if the response to the **start chinit** command within runmqsc is, “Program cannot open queue manager object”, the channel initiator is currently running. If the response is “MQSeries channel initiator started”, the **start chinit** command has now started it.
5. Ensure that the MQSeries channels are available. For example, on UNIX and Win 32, the response to the ping channel **qMgrNameA.qMgrNameB** command within runmqsc should be, “channel **qMgrNameA.qMgrNameB** is in use”. If the response is, “not available retry later”, start the channel using the **start channel qMgrNameA.qMgrNameB** command within runmqsc. The response should be “Start MQSeries channel accepted”.

6. Check that the MQSeries channels are running. For example, use the **display channel status** command within runmqsc on UNIX and Win 32. The status should be “RUNNING”. If the status is “RETRYING”, the previous steps must be checked on the remote queue manager.
7. Verify that all TDE components are running. Check that the correct TDE components are running on their respective nodes. The Receiver must be running on the node specified by the dqm option and the Sender must be running on the node specified by the sqm option.

For example, on the TDE Sender queue manager node, the response to **ftfsdr -lqm qMgrName** should be “TDE Sender initialization complete”.

On the TDE Receiver node, the response to **ftfrcv -lqm qMgrName** should be “TDE Receiver initialization complete”.

On the TDE Manager node, the response to **ftfmgr -lqm qMgrName** should be “TDE Manager initialization complete”.

8. Retry the FTFPING command. The response to **ftfping -lqm qMgrName** should be as follows.

```
QMGRNAME->QMGRNAME->QMGRNAME->QMGRNAME> bytes=310  
time<0 secs
```

After completing the Troubleshooting FTFPING on all nodes, the response to **ftfping -lqm qMgrNameA -dqm qMgrNameB** should be as follows.

```
QMGRNAMEA->QMGRNAMEA->QMGRNAMEA->QMGRNAMEB  
bytes=310 time<0 secs
```

## **FTFSTAGE**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS

### **Description**

The FTFSTAGE command allows you to query the staging queue for a list of its contents or to purge specified contents.

### **Syntax**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

`ftfstage <queue manager node arguments> <process-control arguments>  
<standard environment arguments> <help arguments>`

### **Queue Manager Node Arguments**

The FTFSTAGE command with queue manager node arguments takes the following form:

`FTFSTAGE -lqm localQMgr -oqm origQMgr -sqm sourceQMgr`

Where:

- **-lqm *localQMgr*** – Determines the queue manager from which the FTFSTAGE command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFSTAGE command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (*lqm*) specified.

- **-oqm** *origQMgr* – Determines the queue manager where the query or purge request will be accepted into the TDE subsystem by the processing TDE Manager. If this value is not specified, the oqm is given the same value as the lqm.
- **-sqm** *sourceQMgr* – Name of the queue manager where the staging queues that are to be queried or purged reside. Required for this command.

## Process-Control Arguments

The FTFSTAGE command with process-control arguments takes the following form:

```
FTFSTAGE -lqm localQMgr -oqm origQMgr -sqm sourceQMgr -file  
fileName -ftfid ftfid -purge -all -query -wait waitValue
```

Where:

- **-file** *fileName* – Determines the file that is purged from the staging queue. If you specify this argument, you cannot specify the *-query* or the *-ftfid* arguments.
- **-ftfid** *ftfid* – Determines the FTFID for the data that is purged from the staging queue. If you specify this argument, you cannot specify the *-query* or the *-file* arguments.
- **-purge** – Determines that the FTFSTAGE command will purge the data specified in the *-file* or *-ftfid* arguments. If you specify this argument, you cannot specify the *-query* argument.
- **-all** – Allows you to purge all data that is currently staged. If you use the *-all* argument, you must use the *-purge* argument within the same command. The *-all* argument cannot be used when the *-file* argument is specified.
- **-query** – Determines that the FTFSTAGE command will query the staging queue. If you specify this argument, you cannot specify the *-purge*, *-file*, or *-ftfid* arguments.
- **-wait** *waitValue* – Determines the time to wait for a reply to the query operation. This value does not apply to the purge operation. Specified in seconds. **Default value:** 300 seconds (5 minutes)

#### Example

The following FTFSTAGE command queries the staging queue for entries with lqm and oqm values of PROD9B, and an sqm value of PROD16A.

```
FTFSTAGE -lqm PROD9B -oqm PROD9B -sqm PROD16A -query
```

The following FTFSTAGE command queries the staging queue for entries with lqm and oqm values of PROD9B. The query times out after 10 minutes (600 seconds).

```
FTFSTAGE -lqm PROD9B -oqm PROD9B -query -wait 600
```

The following FTFSTAGE command purges all staging queue entries with a file value of PROD.YEAREND.REPT.

```
FTFSTAGE -lqm PROD9B -oqm PROD12C -purge -file  
PROD.YEAREND.REPT
```

## Standard Environment Arguments

The FTFSTAGE command with standard TDE environment arguments takes the following form:

```
FTFSTAGE -cfile configFile -cq configQueueName -ofile optionsFile  
-version [-help | -h | -?]
```

Where:

- **-cfile *configFile*** – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.



- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.

## **Configuration File and Queue Order of Precedence**

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, TDE checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, TDE checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFSTAGE command. In the options file, you can set any of the command-line arguments that can be set for the FTFSTAGE command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current TDE version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## **Example**

The following FTFSTAGE command queries the staging queue for entries with lqm and oqm values PROD9B. The query times out after 10 minutes (600 seconds).

```
FTFSTAGE -lqm PROD9B -oqm PROD9B -query -wait 600  
-cfile FTFMQ.FTFCONFIG.INI
```

## Tivoli Data Exchange Interface Commands

### *FTFSTAGE*

In this example, the FTFSTAGE command displays the version, release, and patch number for the current TDE software.

```
FTFSTAGE -version
```

## Help Arguments

The FTFSTAGE command with help arguments takes the following form:

```
FTFSTAGE -help | -h | -?
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFSTAGE command-line arguments.

## Example

The FTFSTAGE command displays the FTFSTAGE command-line arguments.

```
FTFSTAGE -h
```

## Using Staging Queues

Instead of transferring data from its source location, you can place it in a staging queue and send it from that queue. Sending data from staging queues allows you to quickly access data that you send frequently. It also allows you to send data from the staging queue when the original source data is no longer available.

To use the staging queue as a source for sending data, you must put the data in the staging queue. Then, at the appropriate time, you can send it from the staging queue.

TDE also allows you to query the staging queue and purge data you want to remove.

## Placing Data in the Staging Queue

To place data in the staging queue, use the FTF command with the *-stageonly* argument. If you use this option, instead of sending the data to a TDE Receiver, you send it to the stage control queue specified in the TDE configuration file. If you send data to the staging queue, you cannot specify the destination path (-dpath) or the destination queue manager (-dqm) in the FTF command.

### Specifying the Staging Queue

The staging queue is specified by the *SenderStageQueue* property in the TDE configuration file. The default value for this queue is FTFS DR.STAGE. To override this default, change the property value to the appropriate queue name and restart TDE. For more information about the TDE configuration file, see “Tivoli Data Exchange Configuration” in the *Tivoli Data Exchange Installation Guide*.

The following command sends the data located in c:\data\reptww01.txt to the stage queue. This example features the *-stageonly* and *-spath* arguments and the options file specification. The rest of the required arguments are specified in the options file.

```
ftf -ofile c:\ftf\options.txt -stageonly -spath  
c:\data\reptww01.txt
```

## **Sending Data from the Staging Queue**

To send data from the staging queue, you can use one of two methods:

- Use the *-fromstage* option with the FTF command.
- Set the *SenderAlwaysCheckStage* property in the FTF configuration file to ON.

### **Using the *-fromstage* option**

If you use the *-fromstage* option with the FTF command, you can specify a filename or an FTFID that identifies the data to be sent. If you specify one of these values, you cannot specify the other.

The following example uses the FTF command with a source filename. All of the required arguments that are not displayed are listed in the specified options file.

```
ftf -ofile c:\ftf\options.txt -fromstage -spath  
c:\data\reptw01.txt
```

---

### **Note:**

If you specify a filename for the transfer from the staging queue and multiple entries have the same filename, the file related to the first matching entry found is sent. If you need to specify a specific instance of a file in the staging queue, use that file's FTFID.

---

The following example uses the FTF command with an FTFID designation. Using the FTFID is useful when more than one file in the staging queue has the same name.

```
ftf -ofile c:\ftf\options.txt -fromstage -ftfid  
0165as8d-5ed3-11d2-8139-f4e7k3256k12
```

To obtain FTFID values from the staging queue, use the FTFSTAGE command with the *-query* option and make note of the desired FTFID. (For more information about the *-query* option, see “Querying the Staging Queue” on page 69.

## Using the *SenderAlwaysCheckStage* Property

The TDE configuration file's *SenderAlwaysCheckStage* property, if enabled, directs TDE to always check the staging queue for the specified data before starting the data transfer. If the specified data is not found in the staging queue, TDE looks for it at a specified location.

To enable the *SenderAlwaysCheckStage* property, open the TDE configuration file, set the property value to ON, then save, close the file, and restart TDE.

After you have enabled *SenderAlwaysCheckStage*, you can use the FTF command as you normally would to specify the data transfer. You cannot use the *-ftfid* argument without the *-fromstage* argument, even when the *SenderAlwaysCheckStage* property is enabled.

### **Warning:**

If you have the *SenderAlwaysCheckStage* property enabled, you could send data from the staging queue when you mean to send data from its source.

## Querying the Staging Queue

Before you send or purge data from the staging queue, you might need to query it for a list of its contents. To query the staging queue for its contents, use the FTFSTAGE command with the *-query* argument. This command and argument return the contents of the staging queue specified in the TDE configuration file.

The following example uses the FTFSTAGE command to return information about the contents of the staging queue. The required arguments that are not displayed are listed in the specified options file.

```
ftfstage -ofile c:\ftf\options.txt -query -wait 300
```

The *-wait* option tells TDE how long to wait for query output. It is specified in seconds. Default value is 300 seconds (5 minutes).

## Tivoli Data Exchange Interface Commands

### *Using Staging Queues*

The following figure displays output from the query argument used with FTFSTAGE. It lists information about each of the files in the staging queue.

Staged Files Found: 4 -----

FTF Id : de679301-8eaa-11d2-af00-ac2f955ce9c8  
filename : c:\images\bigfile.avi  
Date/Time Staged : 12/09/1998 09:32:52  
QMgr : QMGRA  
Message Size : 524298  
Number of messages : 6  
Compressed : Yes

FTF Id : e0da19a1-8eaa-11d2-86e6-c3ecf04f68dc  
filename : c:\temp\assets.txt  
Date/Time Staged : 12/09/1998 09:32:54  
QMgr : QMGRA  
Message Size : 524298  
Number of messages : 2  
Compressed : No

FTF Id : e3574ea1-8eaa-11d2-b371-8e1c4cc0f27f  
filename : d:\forwarding\invoice.dat  
Date/Time Staged : 12/09/1998 09:32:56  
QMgr : QMGRA  
Message Size : 524298  
Number of messages : 1  
Compressed : Yes

FTF Id : e6c60f91-8eaa-11d2-9d82-9dd888d7cdb5  
filename : d:\images\cover.jpg  
Date/Time Staged : 12/09/1998 09:33:02  
QMgr : QMGRA  
Message Size : 524298  
Number of messages : 1  
Compressed : Yes

## Purging the Staging Queue

Purge the staging queue when some of its contents are no longer required to reside there. To purge contents from the staging queue, use the FTFSTAGE command with the *-purge* argument. You can use the *-file* or *-ftfid* arguments to specify data to be purged.

---

### Note:

If you specify a filename to be purged from the staging queue and multiple entries have the same filename, the file related to the first matching entry found is purged. If you need to specify a specific instance of a file in the staging queue, use that file's FTFID.

---

The following example uses the FTFSTAGE command with the *-file* argument to purge the first entry with the filename of c:\data\reptww01.txt. The required arguments that are not displayed are listed in the specified options file.

```
ftfstage -ofile c:\ftf\options.txt -purge -file  
c:\data\reptww01.txt
```

The following example uses the FTFSTAGE command with an FTFID designation. To obtain FTFID values from the staging queue, use the FTFSTAGE command with the *-query* option and make note of the desired FTFID.

```
ftfstage -ofile c:\ftf\options.txt -purge -ftfid  
0165as8d-5ed3-11d2-8139-f4e7k3256k12
```

## **FTFSTAT**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS

---

#### **Note:**

For command-line (FTFSTATD, FTFSTADB) information on the status offload daemon component, see “Tivoli Data Exchange Status Offload Daemon” in the *Tivoli Data Exchange User's Guide*.

---

### **Description**

The FTFSTAT command polls status queues for information about current transfer requests and returns that information. It returns status information from both the status control queue and the status detail queue. The default behavior of FTFSTAT is to return all current day's transactions.

---

#### **Note:**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### **Syntax**

FTFSTAT <queue manager node arguments> <process control arguments><data filter arguments><environment arguments> <help arguments>



## Queue Manager Node Arguments

This section describes arguments that specify the queue managers for which status information is retrieved.

The FTFSTAT command with manager node arguments takes the following form:

```
FTFSTAT -lqm localQueueMgr
```

Where:

- **-lqm** *localQueueMgr* – Determines the local queue manager value where the status messages reside.

### Example

The following FTFSTAT command specifies filter values for each type of queue manager. Only status records that match the specified values are included in the command output.

```
FTFSTAT -lqm PROD15A -oqm PROD22B -sqm PROD22B -dqm  
PROD12C -rqm PROD15A
```

## Process-Control Arguments

This section describes the FTFSTAT arguments used to specify additional values that can be used to filter status information.

The FTFSTAT command with process-control arguments takes the following form:

```
FTFSTAT -purge -format formatOpt -orphans
```

Where:

- **-purge** – Removes the returned status records from the status queues.
- **-format** *formatOpt* – Determines the format of the returned status information. **Valid values:** Terse, Long, Detail **Default:** Terse

## Tivoli Data Exchange Interface Commands

### *FTFSTAT*

- **-orphans** – Forces FTFSTAT to consider status records for which information is available in the detail queue, but not the control queue. This condition results when status control messages do not arrive on the queue manager on which the FTFSTAT command is running. The status detail messages then will have no associated control message.

### Example

The following FTFSTAT command displays and purges records from the status queues. The returned status information takes the *Terse* format. Orphan records are included in the status output and are also purged.

```
FTFSTAT -purge -format terse -orphans
```

## Data Filters Arguments

This section describes the FTFSTAT arguments used to determine additional data filter values for returning status information. The information returned includes source and destination file information, FTFID value, label value, and date ranges.

The FTFSTAT command with source and target file arguments takes the following form:

```
FTFSTAT -sdate dateTimeVal -edate dateTimeVal -dpath destPath -dqm  
destQueueMgr -ftfid ftfIdValue -i -label labelVal -oqm origQueueMgr -rqm  
reqQueueMgr -spath sourcePath -sqm sourceQueueMgr -status statusType
```

Where:

- **-sdate** *dateTimeVal* – Determines the earliest start date and time for which status information is returned. If you specify the beginning date, but no ending, TDE returns all status information logged after the specified date. **Format:** *YYYYMMDDhhmmss*, where *YYYY* is the four-digit year, *MM* is the two-digit month, *DD* is the day, *hh* is the hour (24-hour format), *mm* is the minute, and *ss* is the second.

## Partial Dates

To specify a partial date, you must specify everything to the left of the smallest specified value. In other words, if you want to specify an hour, you must also specify the day, the month, and the year.

- **-edate** *dateTimeVal* – Determines the latest end date and time for which status information is returned. If you specify the end of the date range, but no beginning, TDE returns all status information logged until the specified date. **Format:** *YYYYMMDDhhmmss*, where *YYYY* is the four-digit year, *MM* is the two-digit month, *DD* is the day, *hh* is the hour (24-hour format), *mm* is the minute, and *ss* is the second.
- **-dpath** *destPath* – Determines the fully qualified path and filename of the destination file for which status records are returned.
- **-dqm** *destQueueMgr* – Determines the destination queue manager value for which status records are returned.
- **-ftfid** *ftfIdValue* – Determines the transaction's FTFID value for which status records are returned. The FTFSTAT command normally displays all status records unless this argument is used. To get a partial list displayed, wildcards can be used. An asterisk can be placed in the character strings at any point to get the records that match the characters entered until the asterisk is encountered.
- **-i** – Ignores case sensitivity in destination and source filenames.
- **-label** *labelVal* – Determines the label value for which status records are returned.
- **-oqm** *origQueueMgr* – Determines the transaction's originating queue manager value for which status records are returned.
- **-rqm** *reqQueueMgr* – Determines the transaction's requesting queue manager value for which status records are returned. The *rqm* is the queue manager from which a data-transfer request was entered.

## Tivoli Data Exchange Interface Commands

### FTFSTAT

- **-spath** *sourcePath* – Determines the fully qualified path and filename of the source file for which status records are returned.
- **-sqm** *sourceQueueMgr* – Determines the transaction's source queue manager value for which status records are returned.
- **-status** *statusType* – Determines the transaction's status type for which status information is returned. **Valid values:** COMP, FAIL, ACTIVE, CANC, EXP.

### Example

The following FTFSTAT command returns status records with the specified source and destination filenames, label, and status types. A date range is specified and case sensitivity is ignored.

```
FTFSTAT -spath FTFMQ.SOURCE.RPT1016 -dpath  
FTFMQ.SOURCE.RPT1016 -i -label "report transfer"  
-sdate 19991204 -edate 19991206 -status COMP
```

## Standard Environment Arguments

This section describes the standard TDE environment arguments used with the FTFSTAT command. It contains syntax information and an example.

The FTFSTAT command with standard TDE environment arguments takes the following form:

```
FTFSTAT -cfile configFile -cq configQueueName -ofile optionsFile  
-version
```

Where:

- **-cfile** *configFile* – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.

- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a cfile and a cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.

### **Configuration File and Queue Order of Precedence**

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, TDE checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, TDE checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFSTAT command. In the options file, you can set any of the command-line arguments that can be set for the FTFSTAT command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current TDE version, release, and patch number. Any command entered with this argument ignores all other arguments and returns only the version information.

### **Examples**

The following FTFSTAT command specifies configuration and options files.

```
FTFSTAT FTFMQ.FTFCONFIG.INI -ofile FTFMQ.FTF.OPT
```

The following FTFSTAT command displays the version, release, and patch number for the current TDE software.

```
FTFSTAT -version
```

## Help Arguments

This section describes the TDE help arguments used with the FTFSTAT command. It contains syntax information and an example.

The FTFSTAT command with TDE help arguments takes the following form:

```
FTFSTAT [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFSTAT command-line arguments.

## Examples

The following FTFSTAT command displays the FTFSTAT command-line arguments.

```
FTFSTAT -h
```

## Additional Information

The TDE configuration within the network can substantially affect the performance of the FTFSTAT command. In general, you should set up the Status Detail Queue and Status Control Queue entries in the TDE configuration file so MQSeries transfers all information to a single queue manager. This approach allows the TDE operator to view all activity within the network from a single point rather than having to query information at each individual node.

FTFSTAT output is ordered according to the sequence in which the control records were received. Generally, this order should be the same as the sequence in which the requests were issued. An exception to this is the -orphan parameter, which produces output ordered by FTFID.

## FTFSTAT Samples

The following FTFSTAT samples provide examples of how FTFSTAT works and illustrates the results.

### **Sample 1**

The default behavior of the command shows a terse list of all transfers issued for the current day. Each output section of four lines shows the current status of that request. If -label was specified in the FTFSTAT command, the output contains the TDE user label; otherwise the FTFID value is listed.

### **Input**

```
>ftfstat -lqm TESTAIX41
```

### **Output**

Each status message found on the status queue generates four lines of output. The first line contains the current state and time stamp. The second line contains either the FTFID or the TDE user label. The third line contains the source queue manager name and the source filename. The fourth line contains the destination queue manager name and the target filename.

```
FTF_REQUEST:  REQUEST_SUBMITTED    11/27/1999 13:41:48
- 11/27/1999 13:41:48
c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT3)C:\FTFMQ\TempFiles\FTFbeta2-T1-1869.txt

FTF_REQUEST:  REQUEST_SUBMITTED    11/27/1999 13:46:48
- 11/27/1999 13:46:48
c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT3)C:\FTFMQ\TempFiles\FTFbeta2-T1-1860.txt

FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 12:40:28
- 11/27/1999 12:40:36
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT3)C:\FTFMQ\TempFiles\FTFbeta2-T1-6338.txt

FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 13:30:25
- 11/27/1999 13:40:30
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)C:\FTFMQ\TempFiles\FTFbeta2-T1-6700.txt
```

## Tivoli Data Exchange Interface Commands

### *FTFSTAT*

```
FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 13:28:28
- 11/27/1999 13:28:30
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT2)C:\FTFMQ\TempFiles\FTFbeta2-T1-3211.txt

FTF_MANAGER:  REQUEST_FAILED        11/27/1999 13:25:28
- 11/27/1999 13:25:32
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT2)X:\FTFMQ\TempFiles\FTFbeta2-T1-0211.txt
Error: 630 (637) : Receiver reported a failure
```

#### **Sample 2**

The following is a request for status on a specific data transfer using a wild card on the FTFID. The output shows the long format, which lists the latest status at the rqm, the oqm, the sqm, and the dqm. The command output displays the status of all TDE components by the node on which they operate along with the status of the data transfer at that node.

#### **Input**

```
>ftfstat -lqm TESTAIX41 -ftfid ecbcccb1-e4fd* -format
long
```

#### **Output**

```
FTF Identifier      :
    ecbcccb1-e4fd-11d1-9bbb-8e4e2be42fed
Timestamp           : 11/27/1999 13:25:28
Request QMgr        : TESTNT2

Originating QMgr    : TESTNT2
Status              : REQUEST_FAILED
Status time         : 11/27/1999 13:25:32
Error               : 630 (637) : Receiver reported
    a failure

Source Qmgr         : TESTNT2
Source File Name    :
    C:\ftfdev\bin\testFiles\smtxt01.txt
Status              : REQUEST_TRANSMITTED
Status time         : 11/27/1999 13:25:29
```



```
Target Qmgr          : TESTNT2
Target File Name     :
  X:\FTFMQ\TempFiles\FTFbeta2-T1-0211.txt
Status               : COMPONENT_FAILED
Status time          : 11/27/1999 14:25:30
```

```
Non-Persistent
Priority             : 5
Bytes sent           : 2625440
Data messages        : 11
Transfer Time        : not available
```

### **Sample 3**

The following example uses the same transfer as Sample 2 but uses the detail format to show all log messages. The first several lines of output contain the same information as displayed for the long format, followed by the details. The details reflect the detailed activity and latest status at each TDE component within a given node.

#### **Input**

```
>ftfstat -lqm TESTAIX41 -ftfid ecbcccb1-e4fd* -format
  detail
```

#### **Output**

```
FTF Identifier       :
  ecbcccb1-e4fd-11d1-9bbb-8e4e2be42fed
Timestamp            : 11/27/1999 13:25:28
Request QMgr         : TESTNT2

Originating QMgr     : TESTNT2
Status               : REQUEST_FAILED
Status time          : 11/27/1999 13:25:32
Error                : 630 (637) : Receiver reported
  a failure

Source Qmgr          : TESTNT2
Source File Name     :
  C:\ftfdev\bin\testFiles\smtxt01.txt
Status               : REQUEST_TRANSMITTED
Status time          : 11/27/1999 13:25:29
```

## Tivoli Data Exchange Interface Commands

### *FTFSTAT*

Target Qmgr : TESTNT2  
Target File Name :  
X:\FTFMQ\TempFiles\FTFbeta2-T1-0211.txt  
Status : COMPONENT\_FAILED  
Status time : 11/27/1999 14:25:30

Non-Persistent  
Priority : 5  
Bytes sent : 2625440  
Data messages : 11  
Transfer Time : not available

Detail Log Messages Found: 9 -----

Timestamp : 11/27/1999 13:25:28  
QMgr : TESTNT2  
Component : FTFREQ\_COMMAND\_LINE  
Status : REQUEST\_SUBMITTED

Timestamp : 11/27/1999 13:25:28  
QMgr : TESTNT2  
Component : FTF\_MANAGER  
Status : REQUEST\_SUBMITTED

Timestamp : 11/27/1999 13:25:29  
QMgr : TESTNT2  
Component : FTF\_SENDER  
Status : REQUEST\_RECEIVED

Timestamp : 11/27/1999 13:25:29  
QMgr : TESTNT2  
Component : FTF\_SENDER  
Status : PROCESSING  
Message Number : 11 of 11

Timestamp : 11/27/1999 13:25:30  
QMgr : TESTNT2  
Component : FTF\_SENDER  
Status : REQUEST\_TRANSMITTED

Timestamp : 11/27/1999 13:25:30  
QMgr : TESTNT2  
Component : FTF\_RECEIVER  
Status : REQUEST\_RECEIVED

```
Timestamp           : 11/27/1999 13:25:30
QMgr                : TESTNT2
Component           : FTF_RECEIVER
Status              : REQUEST_FAILED
Error               : 637 (2) File open failure
                   X:\FTFMQ\TempFiles\FTFbeta2-T1-0211.txt, No such
                   file or directory
```

```
Timestamp           : 11/27/1999 13:25:30
QMgr                : TESTNT2
Component           : FTF_RECEIVER
Status              : COMPONENT_FAILED
Error               : 100210637 File open failure
```

```
Timestamp           : 11/27/1999 13:25:32
QMgr                : TESTNT2
Component           : FTF_MANAGER
Status              : REQUEST_FAILED
Error               : 630 (637) Receiver: File open
                   failure X:\FTFMQ\TempFiles\FTFbeta2-T1-0211.txt, No
                   such file or directory
```

#### **Sample 4**

In the following example, the FTFSTAT command returns status information with a status of *Complete*.

#### **Input**

```
>ftfstat -lqm TESTAIX41 -status COMP
```

#### **Output**

```
FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 12:40:28
- 11/27/1999 12:40:36
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT3)C:\FTFMQ\TempFiles\FTFbeta2-T1-6338.txt

FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 13:30:25
- 11/27/1999 13:40:30
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)C:\FTFMQ\TempFiles\FTFbeta2-T1-6700.txt
```

## Tivoli Data Exchange Interface Commands

### *FTFSTAT*

```
FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999 13:28:28
- 11/27/1999 13:28:30
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTNT2)C:\FTFMQ\TempFiles\FTFbeta2-T1-3211.txt
```

#### **Sample 5**

In the following example, the FTFSTAT command returns status information for each of the data transfers that is writing output to the /tmp filesystem on a specified target system (-dpath "/tmp\*"). The output for this query is in the default terse format.

#### **Input**

```
>ftfstat -lqm TESTAIX41 -dpath "/tmp*" -dqm
TESTAIX41
```

#### **Output**

```
FTF_REQUEST:  REQUEST_SUBMITTED    11/27/1999
13:41:48 - 11/27/1999 13:41:48
c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-2992.txt

FTF_REQUEST:  REQUEST_SUBMITTED    11/27/1999
13:46:48 - 11/27/1999 13:46:48
c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-0031.txt

FTF_MANAGER:  REQUEST_COMPLETE      11/27/1999
12:40:28 - 11/27/1999 12:40:36
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-5431.txt

FTF_MANAGER:  REQUEST_SUBMITTED    11/27/1999
13:30:25 - 11/27/1999 13:30:25
97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-4432.txt
```

## **Using the FTF Publish/Subscribe Function**

The following commands relate to FTF's publish/subscribe function, which can be used to distribute information simultaneously to many nodes. For more information, refer to the "Publish/Subscribe" chapter in the *Tivoli Data Exchange User's Guide*.

## **FTFBRK**

### **Supported Operating Systems**

- Win 32
- UNIX Solaris
- UNIX HP-UX
- UNIX AIX
- Linux

---

#### **Note:**

MQSeries 5.1 can be used on all platforms except Win 32, which requires MQSeries 5.2.

---

### **Description**

This command starts the publish/subscribe component FTFBRK. This component serves as the receiving node's communicator with the MQSeries broker network. FTFBRK performs the actual subscribing for the receiving node, serves as a mini-broker for the receiving nodes, and preprocesses the data, marker, and control messages for the Receiver.

### **Syntax**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

**FTFBRK** <queue manager arguments> <process arguments> <environment arguments> <help arguments>

The following sections describe each type of command-line argument used with the FTFBRK command:

- Queue Manager Arguments (see page 87)
- Process Arguments (see page 87)
- Environment Arguments (see page 88)

- Help Arguments (see page 89)

## Queue Manager Arguments

This section describes the FTFBRK arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTFBRK command with the queue manager arguments takes the following form:

**FTFBRK** *-lqm localQueueMgr -bqm brokerQueueMgr*

Where:

- **-lqm *localQueueMgr*** – Name of the queue manager from which the FTFBRK command is issued. If it is not specified, the command attempts to connect to the default queue manager specified.
- **-bqm *brokerQueueMgr*** – Name of the broker queue manager where subscriptions are stored. Currently supports only a local queue manager.

## Example

In the following example, the FTFBRK command is run with an lqm and bqm of PROD16A.

```
FTFBRK -lqm PROD16A -bqm PROD16A
```

## Process Arguments

This section describes the FTFBRK arguments used to specify process values. It includes a syntax model and examples for the command-line options.

The FTFBRK command with the process arguments takes the following form:

**FTFBRK** *-cacheSize cacheNumber -lfile logFileName*

Where:

## Tivoli Data Exchange Interface Commands

### *FTFBRK*

- **-cacheSize** *cacheNumber* – Designates the number of subscriptions that are to be loaded into cache for processing. **Default value:** 100 **Valid values:** 0-1000
- **-lfile** *logFileName* – Name and fully qualified path of the log file where messages are to be written.

### Example

In the following example, the FTFBRK command is run with an lqm and bqm of PROD16A, 40 subscriptions to be loaded, and a log file specified.

```
FTFBRK -lqm PROD16A -bqm PROD16A -cacheSize 40  
-lfile c:\ftf\log
```

### Environment Arguments

This section describes the FTFBRK arguments used to specify environment values. It includes a syntax model and examples for the command-line options.

The FTFBRK command with the environment arguments takes the following form:

```
FTFBRK -cfile configFile -cq configQueueName -ofile optionsFile  
-version
```

Where:

- **-cfile** *configFile*– The fully qualified path and filename of the configuration file to be used by FTFBRK. You cannot specify both a cfile and a cq argument in the same command.
- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this FTF instance on this node. The cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a cfile and a cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.



- **-ofile** *optionsFile* – The fully qualified path and filename of a text file used to contain command-line arguments for the FTFBRK command. In the options file, you can set any of the command-line arguments that can be set for the FTFBRK command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Example

In the following example, the FTFBRK command is run with an lqm and bqm of PROD16A, and configuration file and options filenames are specified.

```
FTFBRK -lqm PROD16A -oqm PROD16A -cfile  
FTFMQ.FTFCONFIG.INI -ofile FTFMQ.FTF.OPT
```

## Help Arguments

This section describes the help arguments used with the FTFBRK command. It contains syntax information and examples.

The FTFBRK command with standard FTF arguments takes the following form:

**FTFBRK** [-help | -h | -?]

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFBRK command-line arguments.

## Examples

In this example, the FTFBRK command-line options are displayed.

```
FTFBRK -h
```

## **FTFPUB**

### **Supported Operating Systems**

- Win 32
- UNIX Solaris
- UNIX HP-UX
- UNIX AIX
- Linux

---

#### **Note:**

MQSeries 5.1 can be used on all platforms except Win 32, which requires MQSeries 5.2.

---

### **Description**

This command publishes a body of information to the MQSeries broker network and specifies characteristics of the information so that it can be sent to subscribers. It contains the source file arguments.

### **Syntax**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

FTFPUB <queue manager node arguments> <process arguments>  
<environment arguments> <authorization arguments> <user exit arguments> <data specification arguments> <help arguments>

The following sections describe each type of command-line argument used with the FTFPUB command:

- Queue Manager Node Arguments (see page 92)
- Process Arguments (see page 92)
- Environment Arguments (see page 95)
- Authorization Arguments (see page 96)
- User Exit Arguments (see page 96)

- Data Specification Arguments (see page 97)
- Help Arguments (see page 99)

## Queue Manager Node Arguments

This section describes the FTFPUB arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTFPUB command with the queue manager arguments takes the following form:

```
FTFPUB -lqm localQueueMgr -oqm origQueueMgr -sqm sourceQueueMgr  
-bqm brokerQueueMgr
```

Where:

- **-lqm** *localQueueMgr* – Name of the queue manager from which the FTFPUB command is issued. This value is required on MVS/ESA systems. On all other systems, if it is not specified, the command attempts to connect to the default queue manager specified.
- **-oqm** *origQueueMgr* – Name of the originating queue manager where the FTF Manager operates. If this value is not specified, the oqm is given the same value as the local queue manager.
- **-sqm** *sourceQueueMgr* – Name of the source queue manager where the FTF Sender operates. If this value is not specified, the sqm is given the same value as the local queue manager.
- **-bqm** *brokerQueueMgr* – Name of the MQSeries broker queue manager where subscriptions are stored.

### Example

In the following example, the FTFPUB command is run with an lqm, oqm, and sqm of PROD16A, and a bqm of PROD7B. The bqm, PROD7B, is the name of the queue manager where the MQSeries broker is running.

```
FTFPUB -lqm PROD16A -oqm PROD16A -sqm PROD16A  
-bqm PROD7B
```

## Process Arguments

This section describes the FTFPUB process arguments. It includes a syntax model and examples for the command-line options.

The FTFPUB command with source and target file arguments takes the following form:

```
FTFPUB -spath sourcePath -topic topicString -stream brokerStream
       -wait waitTime -expiry expireTime -priority priorityCode -tranpersist
       -cancelmode mode -delsrc -label labelValue -ID1 -ID2 -ID3 -DBCS
       -replyq replyQueue -replyqmgr replyQueueMgr
```

Where:

- **-spath** *sourcePath* – The fully qualified path and filename of the source file. This argument is required unless the file is being sent from the staging queue and an FTF ID is specified.
- **-topic** *topicString* – List of the topics to which the messages are to be published.
- **-stream** *brokerStream* – The character stream designating the registered MQSeries broker. If this is not specified, the MQSeries default is used.
- **-wait** *waitTime* – The amount of time, in seconds, to wait for a reply. This argument indicates that the FTFPUB command waits for a response from the FTF Manager to indicate whether a request has succeeded or failed. This indication shows whether the publication was successful. It does not report if all subscribers received the publication. Use FTFSTAT for additional information. If the FTF Manager does not respond within the specified time period, the command times out.
- **-expiry** *expireTime* – The time period, in minutes, after which the file-transfer request expires. If the expiration duration is exceeded, the request is terminated and the FTFRCI\_REQUEST\_EXPIRED message is returned. If the expiration occurs partway through a request, the request is marked as expired. This setting cannot be turned off. **Default value:** 1440 (24 hours) **Valid values:** 1-525600
- **-priority** *priorityCode* – The priority applied to the file-transfer request. **Valid values:** 1 (highest) – 5 (lowest) **Default value:** 1
- **-tranpersist** – Specifies that the data being transferred is persistent. If you specify this argument, the data still exists after a system reboot or FTF shutdown. Using this argument increases the recovery ability of FTF but reduces performance.
- **-cancelmode** *mode* – Provides a command-line override to the preemptive cancel flag in the FTF configuration file. **Valid values:** ON, OFF

If you specify `-cancelmode ON`, FTF looks for a cancel message between the processing of each data block and, if found, cancels the transaction during processing. If you specify `-cancelmode OFF`, FTF does not look for a cancel message between the processing of each data block. By choosing `-cancelmode OFF`, you increase the efficiency of the FTF transaction but you disallow the ability to cancel the transaction after processing begins.

If you do not supply the `-cancelmode` option, the default is the value set from the corresponding preemptive cancel flag that is specified in the FTF configuration file (ReceiverCancel and SenderCancel).

However, the FTF transaction can still be cancelled if the cancel message is received before the processing of the transaction begins and the `-cancelmode` is set to `OFF` or if the preemptive cancel flags in the FTF configuration file are set to `NO`.

- **-delsrc** – Indicates that the source data is to be deleted after the data-transfer request is completed.
- **-label** *labelValue* – Specifies the transaction label.
- **-ID1 (-ID2, -ID3)** – User-defined identifier.
- **-DBCS** – Specifies the option to scan the source file and split each data block at the point where double-byte validity is maintained. Using this option ensures that there is no truncation of a double-byte character when the specified message size does not span the entire character.
- **-replyq** *replyQueue* – Name of the queue to which the reply message is to be routed.
- **-replyqmgr** *replyQueueMgr* – Name of the queue manager to which the reply message is to be routed.

## Example

In the following example, the text information in `c:\pubs\memo.txt` is published on the broker's default stream that is connected to the queue manager named `AIXQMGR`.

```
FTFPUB -topic /Florida/TPA/Development
-spath c:\pubs\memo.txt -bqm AIXQMGR -sqm NTQMGR
-type TEXT
```

## Environment Arguments

This section describes the environment arguments used to direct how the FTFPUB command operates. It includes a syntax model and examples for the command-line options.

The FTFPUB command with environment arguments takes the following form:

```
FTFPUB -cfile configFile -cq configQueueName -ofile optionsFile -version
```

Where:

- **-cfile *configFile*** – The fully qualified path and filename for the FTF configuration file. If no cq argument is specified, this value must be specified.
- **-cq *configQueueName*** – The queue from which the configuration information is to be retrieved for this FTF instance on this node. If no cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file.
- **-ofile *optionsFile*** – The fully qualified path and filename of a text file used to contain command-line arguments for the FTFPUB command. In the options file, you can set any of the command-line arguments that can be set for the FTFPUB command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Example

In this example, the configuration queue is PARMQUEUE and the options file is FTFMQ.FTF.OPT. Required command-line options not listed are specified in the options file.

```
FTFPUB -cq PARMQUEUE -ofile FTFMQ.FTF.OPT
```

## Authorization Arguments

**\*\*ADD oid opw sid spw\*\***

## User Exit Arguments

This section describes the FTFPUB arguments used to specify user exit arguments. It contains syntax information and an example. For more information about user exits see “User Exits” in the *enableNet Tivoli Data Exchange User’s Guide*.

The FTFPUB command with user exit arguments takes the following form:

```
FTFPUB -exit exitNo -exitdll dllName -exitentry entryPoint  
-exitdata commandArg
```

Where:

**-exit *exitNo*** – The exit number to be invoked. Since FTFPUB only deals with the source information, the valid exits are the FTFMGR exits (3, 4), the FTFSDBR exits (5, 6), and the FTFSDBR connector (9).

**Valid values:** 3, 4, 5, 6, 9

---

### Note:

- For more information about custom connector development, contact CommerceQuest ([www.commercequest.com](http://www.commercequest.com)).
  - If you have connectors on a Solaris 2.5.1 operating system, you must install Solaris Patch 103627.
- 

**-exitdll *dllName*** – The DLL, shared object, or load module used to invoke the exit module.

**-exitentry *entryPoint*** – Name of the function in the DLL that contains the exit module.



**-exitdata** *dataValue* – The command-line argument to execute when you invoke a user exit that requires input parameters..

---

### **Note:**

You must keep the exit arguments together and in order on the command line. If you specify an argument that is not associated with the current exit, the command generates an error when you run it.

If you specify an exitdata argument that is not adjacent to the exit's other arguments specified, the exit data is ignored.

---

## **Example**

In this example, the sample user exit for exit 5 is invoked to run Notepad. Required command-line options not listed are specified in the options file.

```
FTFPUB -ofile c:\ftf\FTF.OPT -exit 5
-exitdll FTFEX56 -exitentry FTFSdrPre5
-exitdata "c:\winnt\system32\notepad.exe"
```

## **Data Specification Arguments**

The FTFPUB command with data specification arguments takes the following form:

```
FTFPUB -type fileType -padchar charValue -recpad recordPadValue
-compress -msgsize sizeValue
```

Where:

- **-type** *fileType* – Determines whether the file is text or binary. **Valid values:** TEXT, BINARY **Default value:** BINARY
- **-padchar** *charValue* – The value of the character to be used as the padding character that is to be inserted at the end of the record when the receiving file length is greater than the inbound file.

- **-recpad** *recordPadValue* – Enables or disables the padding facility. FTF provides the following options:
  - **NOPAD** – Specifies that blanks or the padchar value are not to be inserted in each record to fill it out to the length of the other records in the file.
  - **PAD** – Specifies that blanks, or selected padding characters, are to be inserted in each record to fill it out to the length of the other records in the file.

### How Padding Works

FTF padding functionality is governed at the source node (the FTF Sender) and by the characteristics of the data being transferred. No padding decisions are made at the target node (the FTF Receiver). If you request a data transfer in which the source and target nodes both run under operating systems that support record-formatted I/O (such as OS/390), using padding may cause undesired results if you specify different record lengths for your source and target files.

For example, if you send an OS/390 source file with a record length of 80 bytes to an OS/390 target file with a record length of 100 bytes, and if you enable padding, the target file's records are padded up to 80 bytes and contain blanks thereafter.

In addition, if you use the wrap option and the padding option, the padded characters will wrap if the record length of the target file is less than the source file's record length. You can specify padding in the FTF command, the FTF configuration file, and the ISPF panels.

- **-compress** – Specifies that the data being sent is compressed using the internal FTF compression algorithm. Text compression is not supported with FTFPUB.

---

### Note:

This compression utility is not supported on the 4690 platform.

---

- **-msgsize** *sizeValue* – Specifies the message size value that is used to override the MQSeries message size value. The value is specified in kilobytes. **Valid values:** 1-3906 KB (3.9 MB) **Default value:** 512

## Example

The following FTFPUB command has the specified settings in the options file, FTFMQ.PUB.OPT. Required command-line options not listed are specified in the options file.

```
FTFPUB -ofile FTFMQ.PUB.OPT -compress -msgsize 3906  
-type Binary
```

## Help Arguments

- **-help, -h, or -?** – Displays a list and description of the FTFPUB command-line arguments.

## Examples

In this example, the FTFPUB command-line options are displayed.

```
FTFPUB -h
```

## **FTFSUB**

### **Supported Operating Systems**

- Win 32
- UNIX Solaris
- UNIX HP-UX
- UNIX AIX
- Linux

---

#### **Note:**

MQSeries 5.1 can be used on all platforms except Win 32, which requires MQSeries 5.2.

---

### **Description**

This command submits a subscription to the MQSeries broker network. It contains the target file arguments. It also gives the ability to query and purge existing subscriptions.

### **Syntax**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

```
FTFSUB <queue manager node arguments> <environment arguments>  
<subscription arguments> <process arguments> <authorization arguments>  
<user exit arguments> <data specification arguments> <AS/400 arguments>  
<MVS/ESA arguments> <help arguments>
```

The following sections describe each type of command-line argument used with the FTFSUB command:

- Queue Manager Node Arguments (see page 101)
- Environment Arguments (see page 102)
- Subscription Arguments (see page 103)
- Process Arguments (see page 103)
- Authorization Arguments (see page 104)
- User Exit Arguments (see page 104)
- Data Specification Arguments (see page 106)
- AS/400 Arguments (see page 108)
- MVS/ESA Arguments (see page 110)
- Help Arguments (see page 112)

## Queue Manager Node Arguments

This section describes the FTFSUB arguments used to specify queue manager node values. It includes a syntax model and examples for the command-line options.

The FTFSUB command with the queue manager node arguments takes the following form:

```
FTFSUB -lqm localQueueMgr [-node nodeName | -all] -bqm  
brokerQueueMgr
```

Where:

- **-lqm *localQueueMgr*** – Name of the queue manager from which the FTFSUB command is issued. This value is required on MVS/ESA systems. On all other systems, if it is not specified, the FTFSUB command attempts to connect to the default queue manager specified.
- **-node *nodeName*** – Lists the nodes that have subscriptions. Must use either -node or -all argument.
- **-all** – Subscribes to every node in the FTFBRK environment.
- **-bqm *brokerQueueMgr*** – Name of the broker queue manager where subscriptions are stored.

## Example

In the following example, the FTFSUB command is run with an lqm of PROD16A, and a dqm of PROD7B.

```
FTFSUB -lqm PROD16A -bqm PROD16A -node PROD7B,  
PROD16A
```

## Environment Arguments

This section describes the FTFSUB arguments used to specify environment arguments. It contains syntax information and an example.

The FTFSUB command with environment arguments takes the following form:

```
FTFSUB -cfile configFile -cq configQueueName -ofile optionsFile -version
```

Where:

- **-cfile** *configFile* – The fully qualified path and filename for the FTF configuration file. If no cq argument is specified, this value must be specified.
- **-cq** *configQueueName* – The queue from which the configuration information is to be retrieved for this FTF instance on this node. If no cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file.
- **-ofile** *optionsFile* – The fully qualified path and filename of a text file used to contain command-line arguments for the FTFSUB command. In the options file, you can set any of the command-line arguments that can be set for the FTFSUB command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Subscription Criteria Arguments

This section describes the FTFSUB arguments used to specify source and target files. It includes a syntax model and examples for the command-line options.

The FTFSUB command with source and target file arguments takes the following form:

```
FTFSUB -stream brokerStream -topic topicString
```

Where:

- **-stream** *brokerStream* – Name of the queue where messages are to be published. If this is not supplied, the MQSeries default is used. The administrator can create a new stream queue, or the MQSeries broker can create a permanent dynamic queue based on an existing SYSTEM.BROKER.MODEL.STREAM queue. If the administrator creates the stream, then the NOSHARE option must be specified. For more information on streams, please see the *MQSeries Publish/Subscribe User's Guide*.
- **-topic** *topicString* – Contains the topic for desired items published on the MQSeries broker network. You can only subscribe to one topic at a time. You can subscribe many nodes to one topic with one FTFSUB request.

## Example

In the following example, the broker stream is BRKR1016 and the topic is /Florida/TPA/Development.

```
FTFSUB -stream BRKR1016  
-topic /Florida/TPA/Development
```

## Process Arguments

This section describes the FTFSUB arguments used to specify user exit arguments. It contains syntax information and an example.

The FTFSUB command with user exit arguments takes the following form:

```
FTFSUB -query -detail detailValue -purge -purgenode -wait waitValue
```

Where:

- **-query** – Determines that a query of subscriptions should be made. If you use this argument, also use either the node argument to return all the subscriptions for that node, or the topic argument to return all subscriptions for that topic.
- **-detail *detailValue*** – Displays detailed information about subscriptions.  
**Valid values:** MVS, AS400, ALL
- **-purge** – Determines that a particular subscription is to be purged from the specified node. Use the node and the topic arguments to specify the node where purging is to take place and the topic to be purged.
- **-purgenode** – Determines that all of the subscriptions for a particular node are to be purged. Use the node argument to specify the node.
- **-wait *waitValue*** – The time, in seconds, to wait for a reply. This argument indicates that the FTF command waits for a response from the FTF Manager to indicate whether a request has succeeded or failed. If the FTF Manager does not respond within the specified time period, the command times out.

## Example

This example creates a subscription to the topic /Florida/TPA/Development and stores what is published in c:\pubs\development.txt.

```
FTFSUB -topic /Florida/TPA/Development  
-dpath c:\pubs\development.txt -bqm PROD16A  
-type TEXT -node PROD7B
```

## Authorization Arguments

**\*\*ADD did dpw\*\***

## User Exit Arguments

This section describes the FTFSUB arguments used to specify user exit arguments. It contains syntax information and an example.



The FTFSUB command with user exit arguments takes the following form:

FTFSUB -exit *exitNo* -exitdll *dllName* -exitentry *entryPoint* -exitdata  
*commandArg*

Where:

**-exit** *exitNo* – The exit number to be invoked for target information. Since FTFSUB deals only with target information, valid exits are FTFRVC (7, 8) and Receiver connector (10). **Valid values:** 7, 8, 10

---

**Note:**

- For more information about custom connector development, contact CommerceQuest ([www.commercequest.com](http://www.commercequest.com)).
  - If you have connectors on a Solaris 2.5.1 operating system, you must install Solaris Patch 103627.
- 

**-exitdll** *dllName* – The DLL, shared object, or load module used to invoke the exit module.

**-exitentry** *entryPoint* – Name of the function in the DLL that contains the exit module.

**-exitdata** *dataValue* – The command-line argument to execute when you invoke a user exit that requires input parameters..

---

**Note:**

You must keep the exit arguments together and in order on the command line. If you specify an argument that is not associated with the current exit, the command generates an error when you run it.

If you specify an exitdata argument that is not adjacent to the exit's other arguments specified, the exit data is ignored.

---

## Example

In this example, the sample user exit for exit 7 is invoked to run Notepad. Required command-line options not listed are specified in the options file.

```
FTFSUB -ofile c:\ftf\FTF.OPT -exit 7 -exitdll  
FTFEX78 -exitentry LoadParms  
-exitdata "c:\winnt\system32\notepad.exe"
```

## Data Specification Arguments

This section describes the FTFSUB arguments used for data specification. It contains syntax information and examples. The default value for each argument is the value set in the FTF configuration file.

```
FTFSUB -dpath destPath -mkdirs -mode modeValue -pool poolName  
-recwrap wrapMode
```

Where:

- **-dpath** *destPath* – The fully qualified path and filename of the destination file. This argument is required unless the transaction sends the file to the staging queues.

### File-Naming Conventions and MVS/ESA Transfers

When you transfer files to or from an MVS/ESA system, you must use specific data set naming conventions. A data set name that is contained within parentheses and is not accompanied by the symbols “+” or “-” is treated as a PDS member. A data set name that is contained within parentheses and is accompanied by the symbols “+” or “-” is recognized as a Generation Data Group. All file mode options (create, append, and noreplace) and transfer types (binary and text) are supported for PDS members. If the PDS specified for the destination filename does not exist, FTF automatically allocates the PDS and creates the member.

- **-mkdirs** – Creates the directories required to support the dpath value. If this argument is not specified and the specified directory does not exist, the file-transfer request fails with a FILE OPEN ERROR.
- **-mode** *modeValue* – Determines what occurs when the file is written to the target. Although the default value for this argument is CREATE, the only values you can specify from the command line are APPEND and NOREPLACE. Do not use this argument unless you want to override the default CREATE setting. **Valid values:** APPEND, NOREPLACE
- **-pool** *poolName* – Name of the data pool used for transferring file data from the FTF Sender to the FTF Receiver. If this value is not specified, the default pool specified in the FTF configuration file is used. For this option to function, the specified pool must be defined in the FTF configuration file.
- **-recwrap** *wrapMode* – Indicates how records will be processed when they reach the target file and the records are longer than the target record length. FTF provides the following options:
  - WRAP - wraps records that are of greater length than the target file record length.
  - TRUNCATE - truncates records up to the record length of the target file.
  - FAIL - fails the file-transfer request when the record length exceeds the maximum allowed for the target file.

## Example

In this example, the sample user exit for exit 7 is invoked to run Notepad. Required command-line options not listed are specified in the options file.

```
FTFSUB -ofile c:\ftf\FTF.OPT -dpath FTFPUBS.LIB  
-mkdirs -mode APPEND -recwrap WRAP
```

## **AS/400 Arguments**

This section describes the FTFSUB arguments used to specify AS/400 arguments. It contains syntax information and examples. The default value for each argument is the value set in the AS400 DEFAULTS portion of the FTF configuration file.

The FTFSUB command with AS/400 arguments takes the following form:

```
FTFSUB -as400ft fileType -crtlib createLib -libasp libAuxStoragePool  
-fileasp fileAuxStoragePool -libtxt libText -filetxt fileText  
-ccsid codedCharSetId -rcdlen recordLength
```

Where:

- **-as400ft *fileType*** – The value entered specifies the type of AS/400 file.  
**Valid values:** \*DFLT (defaults to the value entered in the configuration table) \*SAVE (specifies an AS/400 Save file) \*SRCPF (specifies an AS/400 source physical file) \*IFS (Specifies an AS/400 IFS file)
- **-crtlib *createLib*** – Specifies that FTF is to create the specified library if it does not exist. **Valid values:** YES, NO
- **-ccsid *codedCharSetId*** – The identifier for the file-transfer request. If CCSID is not specified, FTF uses the CCSID of the job. **Valid values:** 1-65535
- **-fileasp *fileAuxStoragePool*** – The Library Auxiliary Pool for a file that FTF creates for a file-transfer request. **Valid values:** 1-16
- **-filetxt *fileText*** – The file description for a library that FTF creates for a file-transfer request.
- **-libasp *libAuxStoragePool*** – The Library Auxiliary Pool for a library that FTF creates for a file-transfer request. **Valid values:** 1-16
- **-libtxt *libText*** – The library description for a library that FTF creates for a file-transfer request.
- **-rcdlen *recordLength*** – The record length for the target file on AS/400. **Valid values:** 13-32766

## Example

In this example, the values listed in the following table must be set in the FTFSUB command.

Value	Setting
AS/400 File Type	*DFLT
Create Library	YES
Library Aux Storage Pool	1
File Aux Storage Pool	1
Library Text	TARGET LIBRARY
File Text	TARGET FILE
CCSID	129
Record Length	255

The following FTFSUB command requests a file transfer with the specified settings. Required command-line options not listed are specified in the options file.

```
FTFSUB -ofile FTFMQ.ETF.OPT -as400ft *DFLT -crtlib YES
-libasp 1 -fileasp 1 -libtxt TARGET LIBRARY -filetxt TARGET FILE
-ccsid 129 -redlen 255
```

## MVS/ESA Arguments

This section describes the FTFSUB arguments used to specify MVS/ESA arguments. It contains syntax information and examples. The default value for each argument is the value set in the MVSDEFAULTS portion of the FTF configuration file.

### Specifying an Esoteric Unit Name

To specify an esoteric name for the MVS/ESA UNIT value, follow these steps:

- Do not set a value in the MVSVOLUME stanza in the FTF configuration file.
- Specify the unit value in the **-unit** argument or in the MVSUNITNAME stanza in the FTF configuration file on either the FTF Sender or the FTF Receiver

The FTFSUB command with MVS/ESA arguments takes the following form:

```
FTFSUB -org fileOrg -dirblks numBlks -recfmt recordFormat  
-lrecl logRecLength -blksize blockSize -unit unitName  
-volser serialNumber -alcunit allocUnit -primary primAlloc  
-secondary secAlloc -model GDGName
```

Where:

- **-org *fileOrg*** – The file organization of the target file on MVS/ESA. Physical Sequential (PS) and Partitioned Data Set (PDS) are the valid file organization values.
- **-dirblks *numBlks*** – Sets up the number directory blocks used to allocate the target PDS if it does not exist. If this argument is not specified, the value in the configuration file is used. If neither is specified, the PDS allocation will fail. **Valid values:** 1-32760
- **-recfmt *recordFormat*** – The record format for the target file on MVS/ESA. **Valid values:** F (fixed), V (variable), FB (fixed block), and VB (variable block)
- **-lrecl *logRecLength*** – The logical record length for the target file on MVS/ESA. If the value for recfmt is V or VB, then the value for lrecl should be 4 bytes greater than the longest data record. **Valid values:** 1-32760

- **-blksize** *blockSize* – The block size for the target file on MVS/ESA. Specifying a block size of 0 enables the system to choose the optimum block size for the data set during allocation. If the record format is Fixed Block (FB), the block size in the blksize argument must be a multiple of the logical record length, the lrecl parameter. When the record format is variable length (VB), the blksize value must be at least four bytes greater than the lrecl value. **Valid values:** 0-32760
- **-unit** *unitName* – The unit name for the target file on MVS/ESA. This argument's value is installation-dependent. Obtain it from your MVS/ESA System Administrator.
- **-volser** *serialNumber* – The volume serial number for the target on MVS/ESA. This argument's value is installation-dependent. Obtain it from your MVS/ESA administrator.
- **-alcunit** *allocUnit* – The allocation unit used for the target on MVS/ESA. **Valid values:** CYL (cylinder), BLK (block), and TRK (track)
- **-primary** *primAlloc* – The number of primary allocation units required on MVS/ESA.
- **-secondary** *secAlloc* – The number of secondary allocation units required on MVS/ESA.
- **-model** *GDGName* – Indicates a model data set for Generation Data Group (GDG) allocation. Consult your MVS/ESA System Administrator for the available model data sets.

## Example

In this example, the values listed in the following table must be set in the FTFSUB command.

Value	Setting
File Organization	Physical Sequential
Record Format	Fixed Block
Logical Record Length	255
Block Size	25,500
Unit Name	SYSALLDA

## Tivoli Data Exchange Interface Commands

### FTFSUB

Value	Setting
Volume Serial Number	TECH01
Allocation Unit	TRK
Primary Allocation	15
Secondary Allocation	30
Directory Blocks	0

---

### Note:

If a data path has a member name, the file organization setting is physical sequential, and the directory block has a value other than 0, FTF overrides the physical sequential setting and sets the file organization as a partitioned data set.

---

The following FTFSUB command requests a file transfer with the specified settings. Required command-line options not listed are specified in the options file.

```
FTFSUB -ofile FTFMQ.ETF.OPT
-dpath FTFMQ.TARGET.RPT1016 -org PS -recfmt FB
-lrecl 255 -blksize 25500 -unit SYSALLDA
-volser TECH01 -alcunit TRK -primary 15
-secondary 30 -dirblks 0
```

## Help Arguments

This section describes the help arguments used with the FTFSUB command. It contains syntax information and examples.

The FTFSUB command with standard FTF arguments takes the following form:

```
FTFSUB [-help | -h | -?]
```

Where:



- **-help, -h, or -?** – Displays a list and description of the FTFSUB command-line arguments.

## **Example**

In this example, the FTFSUB command-line options are displayed.

```
FTFSUB -h
```



---

---

# Component Configuration Commands

This chapter describes the commands that start and configure Tivoli Data Exchange (TDE) components. It contains the following sections:

Section	Page
Overview	116
FTFCFG	117
FTFEND	121
FTFLOG	127
FTFMGR	134
FTFRCV	141
FTFSDR	148
FTFSTART	155

## Assumptions

This chapter makes the following assumptions:

- You have a working knowledge of TDE (FTF) components and their interrelationships.
- You know how to access and use a command line in the appropriate operating system.

## Overview

The commands in this chapter allow you to start, configure, and shut down specified FTF components or to start the entire FTF environment on OS/390. These commands can be used from any command line in the operating systems specified with each command.

The following information is included for each command:

- Supported operating systems
- Description
- Syntax
- Argument summaries, including examples

---

### **Note:**

In many cases, these commands allow you to use a substantial number of command-line arguments. Because of the volume of the arguments, they are presented in functional groups. Unless otherwise noted, any command-line argument can be used with any other argument in the same functional group or in another functional group.

---

You can use the following commands to configure FTF components:

- FTFCFG (see page 117)
- FTFEND (see page 121)
- FTFLOG (see page 127)
- FTFMGR (see page 134)
- FTFRCV (see page 141)
- FTFSDR (see page 148)
- FTFSTART (see page 155)

## FTFCFG

### Supported Operating Systems

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS

### Description

To accommodate interfaces that process information in queues rather than files, the FTFCFG command populates a queue with the specified FTFconfiguration information.

### General Syntax

FTFCFG <queue manager arguments> <standard environment arguments>  
<help arguments>

### Queue Manager Arguments

This section describes the FTFCFG arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTFCFG command with the queue manager arguments takes the following form:

### Syntax

FTFCFG -lqm *localQueueMgr* -node *nodeName* -nodefile *nodeFilename*

Where:

## Component Configuration Commands

### FTFCFG

- **-lqm** *localQueueMgr* – Determines the queue manager to which the FTFCFG command will attach. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFCFG command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-node** *nodeName* – Lists the FTF nodes that require updating. This argument can be specified more than once, and can be used with the **-nodefile** argument. Duplicates are eliminated. On the AS/400 platform, this argument may be used only once. To specify multiple nodes, use the **-nodefile** argument.
- **-nodefile** *nodeFilename* – Specifies a file which lists the FTF nodes that require updating. This argument may be used with the **-node** argument. Duplicates are eliminated.

### Example

The following FTFCFG command sets the local queue manager (PRD1MQM). PRD1MQM is updated as the command is executed.

```
FTFCFG -lqm PRD1MQM -node PRD1MQM
```

## Standard Environment Arguments

This section describes the FTFCFG arguments used to specify environment values. It includes a syntax model and examples for the command-line options.

The FTFCFG command with the standard environment arguments takes the following form:

### Syntax

```
FTFCFG -cfile configFile -ofile optionsFile -version
```

Where:

- **-cfile** *configFile* – Contains the fully qualified path and filename for the FTF configuration file to be loaded into the configuration queue. The configuration queue to be populated is defined in the configuration file, FTFCfgQueue.

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFCFG command. In the options file, you can set any of the command-line arguments that can be set for the FTFCFG command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

### Example

The following FTFCFG command loads the FTF configuration file, FTFCFIG, into the configuration queue defined in FTFCFIG. The node PRD1MQM is the name of the queue manager where the configuration queue resides.

```
FTFCFG -lqm PRD1MQM -node PRD1MQM -cfile FTFCFIG  
-ofile FTFMQ.ETF.OPT
```

The following FTFCFG command displays the version, release, and patch number for the current FTF software.

```
FTFCFG -version
```

## Help Arguments

This section describes the FTFCFG arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTFCFG command with the queue manager arguments takes the following form:

## Syntax

```
FTFCFG [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFCFG command's command-line arguments.

## Component Configuration Commands

### *FTFCFG*

#### **Example**

The following FTFCFG command displays its command-line options.

```
FTFCFG -h
```



## FTFEND

### Supported Operating Systems

- OS/390
- UNIX
- Win 32
- OS/400
- IBM 4690 OS
- OpenVMS

### Description

The FTFEND command shuts down all or specified FTF components. The components do not process the shutdown message while they are processing a data-transfer request. For example, if the FTF Sender is transferring a large file, it does not read the shutdown message from the control queue until it finishes processing its data.

---

#### Note:

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### Syntax

FTFEND <queue manager arguments> <process-control arguments> <standard environment arguments> <help arguments>

## Queue Manager Arguments

This section describes the FTFEND arguments used to specify queue manager values. It includes a syntax model and examples for the command-line options.

The FTFEND command with the queue manager arguments takes the following form:

## Syntax

FTFEND -lqm *localQueueMgr* -node *nodeName*

Where:

- **-lqm *localQueueMgr*** – Determines the queue manager from which the FTFEND command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFEND command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-node *nodeName*** – Lists the FTF node that requires shutdown. This argument can be specified more than once. Duplicates are eliminated. On the AS/400 platform, this argument may be used only once.

## Example

The following FTFEND command shuts down the FTF Manager, FTF Sender, and FTF Receiver processes running on node PRD1MQM.

```
FTFEND -lqm PRD1MQM -node PRD1MQM
```

## Process-Control Arguments

Process-control arguments allow you to specify arguments that control the processing of the shutdown request.

The FTFEND command with process-control arguments takes the following form:

```
FTFEND [-cpt compNum | -component compNum] -all -immediate -quiesce  
-timeout expireVal
```

Where:

- **[-cpt *compNum* | -component *compNum*]** – Specifies the component being shut down. Use either the -cpt argument or the -component argument.

### Valid values:

- **1** – FTF Manager
- **2** – FTF Sender
- **3** – FTF Receiver
- **4** – FTF logging
- **5** – FTF status
- **6** – FTF broker
- **7** – FTF directory monitor
- **-all** – Shuts down all FTF components that are running.

---

### Note:

If neither **-cpt** nor **-all** argument is entered, then FTFEND shuts down the FTF Manager, Sender, and Receiver by default.

---

- **-immediate** – Shuts the component down immediately, before processing any requests that are currently in the control queue. If you specify this argument, you cannot specify the -quiesce argument.
- **-quiesce** – Shuts the component down after all of the requests ahead of it in the queue are processed. If you specify this argument, you cannot specify the -immediate argument. This is the default value so does not have to be specified.

## Component Configuration Commands

### FTFEND

- **-timeout** *expireVal* – Determines the amount of time until the FTFEND times out. If the time limit is exceeded, the specified component still shuts down, but also generates an error message. This value is measured in seconds. **Valid values:** 1-32767

### Example

The following FTFEND command immediately shuts down the FTF Manager component running on PROD11A. The operation times out after 60 seconds.

```
FTFEND -node PROD11A -cpt 1 -immediate -timeout 60
```

## Standard Environment Arguments

This section describes the standard FTF environment arguments used with the FTFEND command. It contains syntax information and examples.

The FTFEND command with standard FTF environment arguments takes the following form:

```
FTFEND -cfile configFile -cq configQueueName -ofile optionsFile -version
```

Where:

- **-cfile** *configFile* – The fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq** *configQueueName* - The queue from which the configuration information is to be retrieved for this FTF instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.

## Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ofile** *optionsFile* – The fully qualified path and filename of a text file used to contain command-line arguments for the FTFEND command. In the options file, you can set any of the command-line arguments that can be set for the FTFEND command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Examples

The following FTFEND command shuts down the FTF Manager on PROD11A. The configuration and options files are specified.

```
FTFEND -node PROD11A -cpt 1 -cfile  
FTFMQ.FTFCONFIG.INI -ofile FTFMQ.FTF.OPT
```

The following FTFEND command displays the version, release, and patch number for the current FTF software.

```
FTFEND -version
```

## Help Arguments

This section describes the help arguments used with the FTFEND command. It contains syntax information and examples.

The FTFEND command with help arguments takes the following form:

```
FTFEND [-help | -h | -?]
```

- **-help | -h, | -?** – Displays a list and description of the FTFEND command's command-line arguments.

## Examples

The following FTFEND command displays its command-line arguments.

```
FTFEND -h
```

## Additional Information

After the FTFEND command is issued, the specified FTFcomponents accept a shutdown request and log their terminations. No further FTF requests are serviced, all resources are cleaned and released, and the FTF components exit.

---

### Note:

When you submit an FTFEND request to shut down FTF component(s) on a local queue manager, a shutdown message is placed on the queue defined as the FTFLogQueue in the FTF configuration file. If this queue is defined as a remote queue, the FTFLOG program running on the remote queue manager is shut down. This shutdown may be an issue if you are centralizing logging and using the FTFLOG program to write the messages to log files.

---

When using FTFEND with clients, you need to specify shutdown of FTFBRK (publish/subscribe broker) and FTFSTATD (status offload) components. Because these are shared processes, you must explicitly specify their shutdown or you could accidentally shut them down on the server when shutting down client processes.

# FTFLOG

## Supported Operating Systems

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS

## Description

The FTFLOG command moves log information from a specified log queue to log files. It is a daemon process that also directs all new log information to the log files.

---

### Note:

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

## Syntax

FTFLOG <Process Arguments><OS/390 arguments> <Standard arguments><Help Arguments>

## Process Arguments

This section describes the process arguments used with the FTFLOG command. It includes syntax and an example.

The FTFLOG command with process FTF arguments takes the following form:

FTFLOG -lqm *localQueueMgr* -ldir *targetDir* -lfile *logFile* -q *logQueue*

## Component Configuration Commands

### FTFLOG

Where:

- **-lqm** *localQMgr* – Determines the queue manager from which the FTFLOG daemon is started. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFLOG command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-ldir** *targetDir* – Determines the directory to which the log files are to be written.
- **-lfile** *logFile* – Determines the log file for the FTFLOG component. Although log files are not supported in OS/390, the information is written to the standard SYSOUT queue.
- **-q** *logQueue* – Determines the name of the log queue from which the log information is taken. You can specify only one log queue in any FTFLOG command.

## OS/390 Arguments

This section describes the FTFLOG arguments used to specify OS/390 arguments. It contains syntax information and examples. The default value for each argument is the value set in the MVS DEFAULTS portion of the FTF configuration file.

### Specifying an Esoteric Unit Name

To specify an esoteric name for the OS/390 UNIT value, follow these steps:

- Do not set a value in the MVSVOLUME stanza in the FTF configuration file.
- Specify the *Unit* value in the **-unit** argument or in the MVSUNITNAME stanza in the FTF configuration file on either the FTF Sender or the FTF Receiver

The FTFLOG command with OS/390 arguments takes the following form:

```
FTFLOG -alcunit allocUnit -blksize blockSize -org fileOrg -lrecl  
logRecLength -primary primAlloc recfmt recordFormat -secondary secAlloc  
-unit unitName -volser serialNumber
```

Where:



- **-alcunit** *allocUnit* – Determines the allocation unit used for the target on OS/390. **Valid values:** CYL (cylinder), BLK (block), and TRK (track).
- **-blksize** *blockSize* – Determines the block size for the log files on OS/390. **Valid values:** 0-32767
- **-lrecl** *logRecLength* – Determines the logical record length for the target file on OS/390. If the value for recfmt is V or VB then the value for lrecl should be 4 bytes greater than the longest data record. **Valid values:** 1-32760
- **-org** *fileOrg* – Determines the file organization of the log files on OS/390. **Valid values:** Physical Sequential (PS), Partitioned Data Set (PDS)
- **-primary** *primAlloc* – Determines the number of primary allocation units required on OS/390.
- **-recfmt** *recordFormat* – Determines the record format for the log files on OS/390. **Valid values:** F (fixed), V (variable), FB (fixed block), and VB (variable block).
- **-secondary** *secAlloc* – Determines the number of secondary allocation units required on OS/390.
- **-unit** *unitName* – Determines the unit name for the target file on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.
- **-volser** *serialNumber* – Determines the volume serial number for the target on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.

---

### **Note:**

When you submit an FTFEND request to shut down FTF component(s) on a local queue manager, a shutdown message is placed on the queue defined as the FTFLogQueue in the FTF configuration file. If this queue is defined as a remote queue, the FTFLOG program running on the remote queue manager is shut down. This shut down may be an issue if you are centralizing logging and using the FTFLOG program to write the messages to log files.

---

#### Example

In this example, the values listed in the following table are set in the FTFLOG command.

Value	Setting
File Organization	Physical Sequential
Record Format	Fixed Block
Logical Record Length	255
Block Size	65,535
Unit Name	SYSALLDA
Volume Serial Number	TECH01
Allocation Unit	TRK
Primary Allocation	15
Secondary Allocation	30

The following FTFLOG command initiates the daemon with the specified settings. The required command-line arguments not listed on the command line are specified in the options file.

```
FTFLOG -ofile FTFMQ.FTF.OPT -org PS -recfmt FB
      -lrecl 255 -blksize 65535 -unit EDISource
      -volser 3144312 -alcunit BLK -primary 15
      -secondary 30
```

## Standard Arguments

This section describes the standard arguments used with the FTFLOG command. It includes syntax and an example.

The FTFLOG command with standard FTF arguments takes the following form:

```
FTFLOG -cfile configFile -cq configQueueName -ofile optionsFile -version
```

Where:

- **-cfile** *configFile* – Contains the fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq** *configQueueName* -Displays the queue from which the configuration information is to be retrieved for this FTF instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command. Use the FTFCFG command to populate the queue with the configuration information.

### **Configuration File and Queue Order of Precedence**

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFLOG command. In the options file, you can set any of the command-line arguments that can be set for the FTFLOG command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

### **Examples**

In this example, a manager, a receiver, and a sender are running on the PROD1 and PROD2 queue managers. The configuration file governing both sets of components includes the following stanza:

```
FTFLogQueue=PROD1_2.QLOG
```

## Component Configuration Commands

### *FTFLOG*

The PROD1\_2.QLOG queue on queue manager PROD1 is defined as a local queue. The PROD1\_2.QLOG queue on queue manager PROD2 is defined as a remote queue to PROD1\_2.QLOG on queue manager PROD1. This will route all log messages to PROD1.

The following command is run to start the FTFLOG daemon:

```
FTFLOG -lqm PROD1 -ldir c:\ftf\log -q PROD1_2.QLOG
```

In this case, the PROD1 queue manager governs the FTFLOG daemon, and log file information is written to the *c:\ftf\log* directory from components running on PROD1 and PROD2.

When the FTFLOG daemon is started, it creates the following log files and writes log information to them:

- PROD1 Manager – PROD1.MGR.LOG
- PROD1 Sender – PROD1.SDR.LOG
- PROD1 Receiver – PROD1.RCV.LOG
- PROD2 Manager – PROD2.MGR.LOG
- PROD2 Sender – PROD2.SDR.LOG
- PROD2 Receiver – PROD2.RCV.LOG

The following FTFLOG command specifies a configuration file and an options file.

```
FTFLOG -ofile FTFMQ.FTF.OPT -cfile  
FTFMQ.FTFCONFIG.INI -ldir c:\ftf\log  
-q PROD1_2.QLOG
```

The following FTFLOG command displays the version, release, and patch number for the current FTF software.

```
FTFLOG -version
```

## Help Arguments

This section describes the standard arguments used with the FTFLOG command. It includes syntax and an example.

The FTFLOG command with standard FTF arguments takes the following form:

FTFLOG [-help | -h | -?]

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFLOG command's command-line arguments.

## Examples

The following FTFLOG command displays its command-line options.

```
FTFLOG -h
```

## FTFMGR

### Supported Operating Systems

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS
- IBM 4690 OS

### Description

The FTFMGR command starts an FTF Manager according to the conditions set in the command-line parameters.

---

#### Note:

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### Syntax

FTFMGR <startup arguments> <standard environment arguments> <help arguments>

### Startup Arguments

The FTFMGR command with startup arguments takes the following form:

FTFMGR -lqm *localQueueMgr* -lfile *logFile* -ltype *logType* -nodename *nodeName*

Where:

- **-lqm** *localQueueMgr* – Determines the queue manager from which the FTFMGR command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFMGR command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.

Otherwise, whenever a command or interfaces starts up it tries to connect to the lqm. If an lqm value is not specified, the interface attempts to connect to the specified default queue manager on platforms where MQSeries supports them.

- **-lfile** *logFile* – Contains the log file to which the FTF Manager writes. Although log files are not supported on OS/390, the information is written to the standard SYSOUT queue.

---

### Note:

On UNIX platforms, if you start an FTF component as a background process, log information is still generated to standard output. To eliminate standard output, redirect it to /dev/null.

---

- **-ltype** *logType* – The distribution type of the log file. Applies to the 4690 Client only. Refer to the chapter on “Client for 4690” in the *Tivoli Data Exchange User’s Guide* for more information.
- **-nodename** *nodeName* – Associates the node name with the instance of FTF components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.

## Examples

In the following example, the FTF Manager is started on PROD11A. Its configuration file is CONFIGMQ.

```
FTFMGR -lqm PROD11A -cfile CONFIGMQ -nodeName shadow
```

## Component Configuration Commands

### *FTFMGR*

In the following example, the FTF Manager is started as a background process in a UNIX environment. A log file is specified to receive log information and output is directed to /dev/null to prevent standard output from being sent to the console.

```
FTFMGR -lqm PROD11A -cfile /opt/ftfmq/config.ini  
-lfile /opt/ftfmq/ftfmgr.log >/dev/null&
```

## Standard Environment Arguments

This section describes the standard FTF environment arguments used with the FTFMGR command. It contains syntax information and an example.

The FTFMGR command with standard FTF environment arguments takes the following form:

```
FTFMGR -cfile configFile -cq configQueueName -jsdataset datasetName  
-ofile optionsFile -version
```

Where:

- **-cfile** *configFile* – Contains the fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this FTF instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command.



## Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-jsdataset** *datasetName* – Contains the name of the dataset that holds the JCL to be executed when exit 3 or 4 is invoked. The argument works only on the OS/390 platform. FTF returns an error if you use this argument on other platforms.
- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFMGR command. In the options file, you can set any of the command-line arguments that can be set for the FTFMGR command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Examples

The following FTFMGR command starts the manager on PROD11A.

```
FTFMGR PROD11A -ofile OPTIONMQ
```

The FTFMGR command displays the version, release, and patch number for the current FTF software.

```
FTFMGR -version
```

## Help Arguments

This section describes the standard FTF arguments used with the FTFMGR command. It contains syntax information and an example.

The FTFMGR command with standard FTF help arguments takes the following form:

```
FTFMGR [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFMGR command's command-line arguments.

## Examples

The following FTFMGR command displays its command-line options.

```
FTFMGR -h
```

## Additional Information

---

### Note:

If environment variable values are changed, you need to restart a running FTFMGR component for the changes to take effect.

---

The startup parameters for the FTF Sender, FTF Manager, and FTF Receiver can be set up for client deliveries only by using the **-nodename** argument. The **-nodename** argument accepts an identifier that is used in the respective **-sqm** or **-dqm** arguments of the request. For example, if the FTF request looks as follows when using MQSeries servers:

```
FTF -sqm MQM1 -dqm MQM2 -spath C:\FILE.1 -dpath  
C:\FILE.2
```

The command when working with clients would look like the following:

```
FTF -sqm CLIENTA -dqm CLIENTB -spath C:\FILE.1  
-dpath C:\FILE.2
```

However, the FTF components on CLIENTA and CLIENTB must be defined in the FTFconfiguration file. The FTFNodeAlias section of the configuration file allows you to define the client nodes and their associated queue managers. These client definitions will use the default client queue naming convention when resolving queue names.

---

### **Note:**

Versions of TDE prior to V1.2 do not support this configuration. Use FTFNodeOverride to define queue managers and client nodes in previous versions of TDE. In FTFNodeOverride, you must define each client separately.

---

To start the FTF Manager component with a node name of CLIENTA connecting to QMGRA, you must define CLIENTA in the FTFNodeAlias section of the configuration file as follows:

```
FTFNodeAlias:  
AliasQueueManager=QMGRA  
Aliases=CLIENTA
```

To define all of the clients that connect to a queue manager, list them in the FTFNodeAlias section of the configuration file as shown in the following example, where CLIENTA and CLIENTB connect to QMGRA:

```
FTFNodeAlias:  
AliasQueueManager=QMGRA  
Aliases=CLIENTA, CLIENTB
```

## Component Configuration Commands

### *FTFMGR*

If you are using a version of TDE prior to V1.2, you must use the `FTFNodeOverride` section of the configuration file to define the name of the client node and its queue manager, and each client name must be defined in the configuration file at every node that wishes to participate with this client. The following example shows entries for `CLIENTA` and `CLIENTB` in the FTF configuration file:

```
FTFNodeOverride:
name=CLIENTA, CLIENTB

CLIENTA:
QueueManager=QMGRA
ManagerControlQueue=CLIENTA.FTFMGR.CONTROL
ManagerSyncQueue=CLIENTA.FTFMGR.SYNC

SenderNumInstances=1
SenderStageControlQueue=CLIENTA.FTFSDR.STAGE.CONTROL
SenderControlQueue=CLIENTA.FTFSDR.CONTROL
SenderSyncQueue=CLIENTA.FTFSDR.SYNC
SenderStageQueue=CLIENTA.FTFSDR.STAGE
SenderMaxStageQueues=1
SenderSystemQueue=CLIENTA.FTFSDR.SYSTEM
SenderCancel=NO

ReceiverNumInstances=1
ReceiverControlQueue=CLIENTA.FTFRCV.CONTROL
ReceiverSyncQueue=CLIENTA.FTFRCV.SYNC
ReceiverStageQueue=CLIENTA.FTFRCV.STAGE
ReceiverSystemQueue=CLIENTA.FTFRCV.SYSTEM
ReceiverCancel=NO

CLIENTB:
QueueManager=QMGRA
ReceiverNumInstances=1
ReceiverControlQueue=CLIENTB.FTFRCV.CONTROL
ReceiverSyncQueue=CLIENTB.FTFRCV.SYNC
ReceiverStageQueue=CLIENTB.FTFRCV.STAGE
ReceiverSystemQueue=CLIENTB.FTFRCV.SYSTEM
ReceiverCancel=NO
```

## FTFRCV

### Supported Operating Systems

- OS/390
- UNIX
- Win 32
- OS/400
- IBM 4690 OS
- OpenVMS

### Description

The FTFRCV command starts the FTF Receiver on the specified queue manager. If you change environment variable values, restart a running FTFRCV component so that the changes can take effect.

---

#### Note:

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### Syntax

FTFRCV <startup arguments><security exit arguments> <standard environment arguments> <help arguments> <connector arguments>

### Startup Arguments

This section describes the startup arguments that allow you to invoke process-control exits. It contains syntax information and an example.

The FTFRCV command with startup arguments takes the following form:

```
FTFRCV -lfile logFile -lqm localQueueMgr -ltype logType -ttype  
targetType -nodename nodeName -sync queueName
```

Where:

- **-lfile** *logFile* – Contains the log file to which the FTF Receiver writes. Although log files are not supported on OS/390, the information is written to the standard SYSOUT queue.

---

### Note:

On UNIX platforms, if you start an FTF component as a background process, log information is still generated to standard output. To eliminate standard output, redirect it to /dev/null.

---

- **-lqm** *localQueueMgr* – Determines the queue manager from which the FTFRCV command is issued. This value is required on OS/390 systems.
- **-ltype** *logType* – The distribution type of the log file. Applies to the 4690 Client only. Refer to the chapter on “Client for 4690” in the *Tivoli Data Exchange User's Guide* for more information.
- **-ttype** *targetType* – The distribution type for the target files. Applies to the 4690 Client only. Refer to the chapter on “Client for 4690” in the *Tivoli Data Exchange User's Guide* for more information.
- **-nodename** *nodeName* – Associates the node name with the instance of FTF components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.
- **-sync** *queueName* – Determines the sync queue used by the FTF Receiver. The FTF Receiver requires an exclusive sync queue to keep track of its units of work. If this option is not specified, the receiver determines for itself which queue to use.

The queue specified with the sync argument have a base *queueName* defined in the *ftfconfig.ini*. For example, in the *ftfconfig.ini* file the default sync queue stanza is as follows:

```
SenderSyncQueue=FTFSDR.SYNC
```

A valid sync *queueName* can be FTFS DR.SYNC.1 or FTFS DR.SYNC.2. The *queueName* is valid as long as it contains the name FTFS DR.SYNC and a number extension that is within the range of senders specified in the `ftfconfig.ini`'s `SenderNumInstances` stanza. The sender sync queue default name can be changed to anything, but the sync *queueName* must contain the text specified in the `SenderSyncQueue` stanza.

## Example

The following FTFRCV command starts the FTF Receiver on PROD11A.

```
FTFRCV -lqm PROD11A -cfile CONFIGMQ
```

In the following example, the FTF Receiver is started as a background process in a UNIX environment. A log file is specified to receive log information and output is directed to `/dev/null` to prevent standard output from being sent to the console.

```
FTFRCV -lqm PROD11A -lfile /opt/ftfmq/ftfmgr.log  
>/dev/null&
```

## Security Exit Arguments

This section describes the arguments that allow you to invoke security authorization exits. It contains syntax information and an example.

The FTFRCV command with security authorization arguments takes the following form:

```
FTFRCV -exitdll dllName -exitentry entryPoint -ftfauth
```

Where:

- **-exitdll** *dllName* – The name of the DLL, shared library, service program, or load module that contains the authorization module.
- **-exitentry** *entryPoint* – The name of the function in the authorization module to be invoked. This function must exist within the DLL. This value is case sensitive.

## Component Configuration Commands

### *FTFRCV*

- **-fttfauth** – Indicates that a security authorization exit is being used on the FTF Receiver.

### Example

The following FTFRCV command invokes a security authorization exit for the specified FTF Receiver.

```
FTFRCV -lqm PROD11A -fttfauth -exitdll authdll  
-exitentry authentry
```

## Standard Environment Arguments

This section describes the standard FTF environment arguments used with the FTFRCV command. It contains syntax information and an example.

The FTFRCV command with standard FTF environment arguments takes the following form:

```
FTFRCV -cfile configFile -cq configQueueName -jsdataset datasetName  
-ofile optionsFile -version
```

Where:

- **-cfile *configFile*** – Contains the fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq *configQueueName*** – Displays the queue from which the configuration information is to be retrieved for this FTF instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command.



## Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-jsdataset** *datasetName* – Contains the name of the dataset that holds the JCL to be executed when exit 7 or 8 is invoked. This argument works only on the OS/390 platform. FTF returns an error if you use this argument on other platforms.
- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFRCV command. In the options file, you can set any of the command-line arguments that can be set for the FTFRCV command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.
- **-su** – (UNIX only) Switches the user account which the FTFRCV file I/O processing is running under. It will switch to the user ID of the user that initiated the request. This will allow the UNIX system to validate the requestor's write access to the files being transferred. Please refer to the *Installation Guide* for configuration details.

---

### Note:

When using the -su option on UNIX, any requests being initiated from MVS or AS/400 require that a lowercase user ID be created on the UNIX machine that matches that of the requestor.

---

---

#### Note:

Requests originating from 4690 are not supported with the -su option.

---

### Examples

The following FTFRCV command starts the FTF Receiver. Its sync queue is SYNC1 and its options file is FTFOPTS.

```
FTFRCV -sync SYNC1 -ofile FTFOPTS
```

The following FTFRCV command displays the version, release, and patch number for the current FTF software.

```
FTFRCV -version
```

### Help Arguments

This section describes the help arguments used with the FTFRCV command. It contains syntax information and an example.

The FTFRCV command with help arguments takes the following form:

```
FTFRCV [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFRCV command's command-line arguments.

### Examples

The following FTFRCV command displays its command-line options.

```
FTFRCV -h
```

---

### Note:

If environment variable values are changed, you need to restart a running FTFRCV component for the changes to take effect.

---

## Connector Arguments

This section describes the arguments that allow you to use connectors on the current receiver. It contains syntax information and an example.

---

The FTFRCV command with connector arguments takes the following form:

```
FTFRCV -connector -connectordll dllName -connectoreentry  
entryPointName -connectordata dataString
```

Where:

- **-connectordll** *dllName* – The name of the DLL, shared library, service program, or load module used to invoke the connector module.
- **-connectoreentry** *entryPoint* – The name of the function in the DLL, shared object, or load module that contains the connector module.
- **-connectordata** *dataString* – Contains data being passed through the FTF Receiver to the connector. This value is optional.

### Example

In the following example, the FTF Receiver on queue manager PROD12A is configured as a connector receiver for the connector module *receiveDailyReport* in the DLL rcvrept.dll.

```
FTFRCV -lqm PROD12A -connector -connectordll  
rcvrept.dll -connectoreentry receiveDailyReport
```

## **FTFSDR**

### **Supported Operating Systems**

- OS/390
- UNIX
- Win 32
- OS/400
- OpenVMS
- IBM 4690 OS

### **Description**

The FTFSDR command starts the FTF Sender on the specified queue manager. If you change environment variable values, restart a running FTFSDR component so the changes can take effect.

---

#### **Note:**

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

### **Syntax**

FTFSDR <startup arguments><security exit arguments> <standard environment arguments> <help arguments> <connector arguments>

### **Startup Arguments**

This section describes the startup arguments that allow you to invoke process-control exits. It contains syntax information and an example.

The FTFSDR command with startup arguments takes the following form:

```
FTFSDR -lqm localQueueMgr -nodename nodeName -lfile logFile  
-ltype logType -sync queueName
```

Where:

- **-lqm** *localQueueMgr* – Determines the queue manager from which the FTFSDR command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFSDR command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-nodename** *nodeName* – Associates the node name with the instance of FTF components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.
- **-lfile** *logFile* – Contains the log file to which the FTF Receiver writes. Although log files are not supported on OS/390, the information is written to the standard SYSOUT queue.

---

### Note:

On UNIX platforms, if you start an FTF component as a background process, log information is still generated to standard output. To eliminate standard output, redirect it to /dev/null.

---

- **-ltype** *logType* – The distribution type of the log file. Applies to the 4690 Client only. Refer to the chapter on “Client for 4690” in the *Tivoli Data Exchange User's Guide* for more information.
- **-sync** *queueName* – Determines the sync queue used by the FTF Sender. The FTF Sender requires an exclusive sync queue to keep track of its units of work. If this option is not specified, the Sender determines for itself which queue to use.

The queue specified with the sync argument have a base *queueName* defined in the `ftfconfig.ini`. For example, in the `ftfconfig.ini` file the default sync queue stanza follows:

```
SenderSyncQueue=FTFSDR.SYNC
```

## Component Configuration Commands

### FTFSDR

A valid sync *queueName* can be FTFSDR.SYNC.1 or FTFSDR.SYNC.2. The *queueName* is valid as long as it contains the name FTFSDR.SYNC and a number extension that is within the range of senders specified in the `ftfconfig.ini`'s `SenderNumInstances` stanza. The sender sync queue default name can be changed to anything but the sync *queueName* must contain the text specified in the `SenderSyncQueue` stanza.

### Example

The following FTFSDR command starts the FTF Sender on PROD11A.

```
FTFSDR -lqm PROD11A -cfile CONFIGMQ
```

In the following example, the FTF Sender is started as a background process in a UNIX environment. A log file is specified to receive log information and output is directed to `/dev/null` to prevent standard output from being sent to the console.

```
FTFSDR -lqm PROD11A -lfile /opt/ftfmq/ftfmgr.log  
>/dev/null&
```

## Security Exit Arguments

This section describes the arguments that allow you to invoke security authorization exits. It contains syntax information and an example.

The FTFSDR command with security authorization arguments takes the following form:

```
FTFSDR -exitdll dllName -exitentry entryPoint -ftfauth
```

Where:

- **-exitdll** *dllName* – The name of the DLL, shared library, service program, or load module that contains the authorization module.
- **-exitentry** *entryPoint* – The name of the function in the authorization module to be invoked. This function must exist within the DLL. This value is case sensitive.
- **-ftfauth** – Indicates that a security authorization exit is being used on the FTF Sender.

## Example

The following FTFSDR command invokes a security authorization exit for the specified FTF Sender.

```
FTFSDR -lqm PROD11A -ftfauth -exitdll authdll  
-exitentry authentry
```

## Standard Arguments

This section describes the standard arguments used with the FTFSDR command. It contains syntax information and an example.

The FTFSDR command with standard arguments takes the following form:

```
FTFSDR -cfile configFile -cq configQueueName -jsdataset datasetName  
-ofile optionsFile -version
```

Where:

- **-cfile *configFile*** – Contains the fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq *configQueueName*** – Displays the queue from which the configuration information is to be retrieved for this FTFInstance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command.

## Configuration File and Queue Order of Precedence

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

- **-jsdataset** *datasetName* – Contains the name of the dataset that holds the JCL to be executed when exit 5 or 6 is invoked. The argument works only on the OS/390 platform. FTF returns an error if you use this argument on other platforms.
- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFSDR command. In the options file, you can set any of the command-line arguments that can be set for the FTFSDR command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

## Examples

The following FTFSDR command starts the FTF Sender. Its sync queue is SYNC1 and its options file is FTFOPTS.

```
FTFSDR -sync SYNC1 -ofile FTFOPTS
```

The following FTFSDR command displays the version, release, and patch number for the current FTF software.

```
FTFSDR -version
```



## Help Arguments

This section describes the help arguments used with the FTFSDR command. It contains syntax information and an example.

The FTFSDR command with help arguments takes the following form:

```
FTFSDR [-help | -h | -?]
```

Where:

- **-help, -h, or -?** – Displays a list and description of the FTFSDR command's command-line arguments.

## Examples

The following FTFSDR command displays its command-line options.

```
FTFSDR -h
```

---

### Note:

If environment variable values are changed, you need to restart a running FTFSDR component for the changes to take effect.

---

## Connector Arguments

This section describes the arguments that allow you to use connectors on the current FTF Sender. It contains syntax information and an example.

The FTFSDR command with connector arguments takes the following form:

```
FTFSDR -connector -connectordll dllName -connectorentry  
          entryPointName -connectordata dataString
```

Where:

- **-connectordll *dllName*** – The name of the DLL, shared library, load module, or service program used to invoke the connector module.

## Component Configuration Commands

### *FTFSDR*

- **-connectorentry** *entryPoint* – The name of the function in the DLL, shared object, load module, or service program that contains the connector module.
- **-connectordata** *dataString* – Contains data being passed through the FTF Sender to the connector. This value is optional.

### Example

The following FTFSDR command starts the FTF Sender on queue manager PROD12A. It connector uses for the entry point of *sendDailyReport* in the DLL sendrept.dll.

```
FTFSDR -lqm PROD12A -connector -connectordll  
sendrept.dll -connectorentry sendDailyReport
```

# FTFSTART

## Supported Operating Systems

- OS/390

## Description

The FTFSTART command starts all FTF components identified in the FTF configuration file.

---

### Note:

Because of the volume of arguments used with this command, they are documented in functional groups. Unless otherwise specified, the arguments in each group can be mixed with those of other functional groups.

---

## Syntax

FTFSTART <security exit arguments> <sysoutclass argument> <startup arguments> <standard environment arguments>

## Security Exit Arguments

The security exit arguments are used to invoke the security authorization exits on the FTF Sender and the FTF Receiver. The FTFSTART command with security authorization arguments takes the following form:

```
FTFSTART -sdrauth -exitdll sDllName -exitentry sEntryPoint  
-rcvauth -exitdll rDllName -exitentry rEntryPoint
```

Where:

- **-sdrauth** – Specifies that a sender user authorization module will be invoked.
- **-exitdll *sDllName*** – Contains the name of the DLL that contains the FTF Sender's authorization module.

## Component Configuration Commands

### *FTFSTART*

- **-exitentry** *sEntryPoint* – Contains the FTF Sender’s entry point to the DLL.
- **-rcvauth** – Specifies that a receiver user authorization module will be invoked.
- **-exitdll** *rDllName* – Contains the name of the DLL that contains the FTF Receiver’s authorization module.
- **-exitentry** *rEntryPoint* – Contains the FTF Receiver’s entry point to the DLL.

### Example

The following FTFSTART command installs the sample security authorization provided with FTF.

```
FTFSTART -sdrauth -exitdll FTFAUTH -exitentry  
FTFAuthex1-rcvauth -exitdll FTFAUTH -exitentry  
FTFAuthex1
```

## SYSOUTCLASS Argument

The SYSOUTCLASS argument allows you to allocate the sysout class for the FTFSTART command, rather than using the default sysout class for the job or the started task. Using this class allows installations where the output from started tasks are automatically purged to view the output from FTF components, even if FTFSTART is initiated as a started task. The FTFSTART command with the SYSOUTCLASS argument takes the following form:

```
FTFSTART -sysoutclass classChar
```

Where:

- **-sysoutclass** *classChar* – Contains the one-character sysout class designation.

## Example

The following FTFSTART command sets the sysoutclass character to H.

```
FTFSTART -lqm PROD00 -cfile FTF230.FTFCONFIG.INI  
-sysoutclass H
```

## Startup Arguments

Startup arguments allow you to specify the local queue manager and the FTF configuration file for the components being started by FTFSTART. The FTFSTART command with standard FTF arguments takes the following form:

FTFSTART -lqm *localQueueMgr* -cfile *configFile* -cq *configQueueName*

Where:

- **-lqm *localQueueMgr*** – Determines the queue manager from which the FTFSTART command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFSTART command attempts to connect to the default queue manager. Otherwise, it tries to connect to the local queue manager (lqm) specified.
- **-cfile *configFile*** – Contains the fully qualified path and filename for the FTF configuration file. On OS/390 platforms, if no -cq argument is specified, this value must be specified. You cannot specify both a -cfile and a -cq argument in the same command.
- **-cq *configQueueName*** – Names the queue from which the configuration information is to be retrieved for this FTF instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points FTF to the queue name rather than to the standard configuration file. You cannot specify both a -cfile and a -cq argument in the same command.

## **Configuration File and Queue Order of Precedence**

On platforms other than OS/390, if you do not specify either a configuration file or a configuration queue, FTF checks the FTF\_CONFIG\_QUEUE environment variable and uses the specified queue. If this environment variable is not set, FTF checks the FTF\_CONFIG\_FILE environment variable and uses the specified file. If neither environment variable is set and no command-line argument is set, the command fails.

### **Example**

The following FTFSTART command starts the manager using a configuration file of FTFV230.FTFCONFIG.INI and its local queue manager is PROD00.

```
FTFSTART -cfile FTF230.FTFCONFIG.INI -lqm PROD00
```

## **Standard Environment Arguments**

This section describes the standard FTF environment arguments used with the FTFSTART command. It contains syntax information and an example.

The FTFSTART command with standard FTF environment arguments takes the following form:

```
FTFSTART -ofile optionsFile -version [-help | -h | -?]
```

Where:

- **-ofile** *optionsFile* – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTFSTART command. In the options file, you can set any of the command-line arguments that can be set for the FTFSTART command. Any values specified on the command line override the values in the options file.
- **-version** – Returns the current FTF version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.
- **-help, -h, or -?** – Displays a list and description of the FTFSTART command's command-line arguments.

## Examples

The following FTFSTART command starts the manager on PROD11A. The options file is FTFV230.FTF.OPTS.

```
FTFSTART -ofile FTFV230.FTF.OPTS
```

The following FTFSTART command displays the version, release, and patch number for the current FTF software.

```
FTFSTART -version
```

The following FTFSTART command displays its command-line options.

```
FTFSTART -h
```





# FTF COBOL API

This chapter describes the Tivoli Data Exchange FTFCOBOL, which allows you to leverage Tivoli Data Exchange for custom business solutions. The FTF COBOL APIs allow the user to submit the FTF requests, FTFPING requests, and FTFStat requests from a COBOL program. The following information is listed for each API function:

This chapter contains the following sections:

Section	Page
The FTFCOBOL Copybook	162
FTFH-HEADER	166
FTFPing Fields	173
FTFStatus Summary List Fields	174
FTF STAT Filter Fields	175
Issuing FTF Transfer Requests	176
Issuing FTF Ping Requests	179
Issuing FTF Status Requests	179

# The FTFCOBOL Copybook

The FTFCOBOL copybook can be found in the COBSAMP dataset. This copybook contains all of the necessary arguments required to make FTF transfer requests, FTFPing requests, and FTFStat requests. The copybook is designed to be as close to the standard command line arguments as possible. The following is the current version of the copybook.

```
.....
*      CommerceQuest Data Integrator (CQDI)      *
*      *                                          *
*      Copyright (c) 1999 - 2001, CommerceQuest, Inc. *
*      All Rights Reserved. *
*      *                                          *
*      MQSeries is a licensed trademark of IBM Corporation. *
*      *                                          *
* CommerceQuest Data Integrator and CommerceQuest *
* are trademarks of CommerceQuest, Inc. *
.....
* DESCRIPTION: COBOL COPY BOOK FOR FTF API INTERFACE *
.....
*$Revision: 25 $
.....
* FTFCOBOL API STRUCTURE
.....
*
01  FTFCOBOL-DATA SYNC.
.....
* HEADER - Common fields for all functions
.....
05  FTFH-HEADER.
    10  FTFH-FUNCTION          PIC X(08).
    10  FTFH-CONFIG-FILE      PIC X(256).
    10  FTFH-MQHANDLE          PIC S9(9) BINARY.
    10  FTFH-RETURN-CODE1      PIC S9(9) BINARY.
    10  FTFH-RETURN-CODE2      PIC S9(9) BINARY.
    10  FTFH-ERROR-MESSAGE     PIC X(1 024).
    10  FTFH-FTFID-ASCII       PIC X(37).
    10  FTFH-FTFID-NATIVE      PIC X(37).
    10  FTFH-LQM               PIC X(49).
    10  FTFH-OQM               PIC X(49).
    10  FTFH-SQM               PIC X(49).
    10  FTFH-DQM               PIC X(49).
.....
* REQUEST - FTFRequest fields
```

\*\*\*\*\*  
05 FTR-REQUEST.  
\*\*\*\*\*

- \* FTR Request fields are divided into two separate sections
- \* the first are all the numeric fields
- \* and then come all the alpha fields
- \* This is because of full-word alignment concerns between
- \* cobol and C.
- \* FTR = Regular request fields
- \* FTRS= Source File attributes
- \* FTRT= Target File attributes
- \* FTRJ= Job attributes
- \* FTRU= User attributes
- \* FTRA= AS400 file attributes
- \* FTRE = Exit information

```
*****
10 FILLER          PIC X(1)  VALUE SPACES.
10 FTR-IS-REPLY     PIC 9(4)  BINARY VALUE 0.
10 FTR-TIMEOUT      PIC 9(9)  BINARY VALUE 60.
10 FTR-REPLY-WAIT-TIME PIC 9(9) BINARY VALUE 60.
10 FTRS-IS-STAGED   PIC 9(4)  BINARY VALUE 0.
10 FTRS-IS-STAGEPERS PIC 9(4)  BINARY VALUE 0.
10 FTRS-IS-DATA-PERS PIC 9(4)  BINARY VALUE 0.
10 FTRS-IS-COMPRESSED PIC 9(4) BINARY VALUE 0.
10 FTRS-IS-DELETE   PIC 9(4)  BINARY VALUE 0.
10 FTRS-FILE-TYPE-INFO PIC 9(4) BINARY VALUE 1.
   88 FTRS-FILE-TYPE-BINARY VALUE 1.
   88 FTRS-FILE-TYPE-TEXT  VALUE 2.
10 FTRS-RECORD-PADDING PIC 9(4) BINARY VALUE 1.
   88 FTRS-PAD VALUE 1.
   88 FTRS-NO-PAD VALUE 2.
10 FTRS-C-STYPE      PIC S9(9) BINARY VALUE 0.
10 FTRS-BUFFNO       PIC S9(9) BINARY VALUE 0.
10 FTRT-IS-COMPRESSED PIC 9(4)  BINARY VALUE 0.
10 FTRT-FILE-TYPE-INFO PIC 9(4) BINARY VALUE 1.
   88 FTRT-FILE-TYPE-BINARY VALUE 1.
   88 FTRT-FILE-TYPE-TEXT  VALUE 2.
10 FTRT-FILE-MODE-INFO PIC 9(4) BINARY VALUE 1.
   88 FTRT-FILE-MODE-CREATE VALUE 1.
   88 FTRT-FILE-MODE-APPEND VALUE 2.
   88 FTRT-FILE-MODE-NOREPLACE VALUE 3.
10 FTRT-FILE-ORG-INFO PIC 9(4) BINARY VALUE 1.
   88 FTRT-FILE-ORG-PARTITIONED VALUE 1.
   88 FTRT-FILE-ORG-SEQUENTIAL VALUE 2.
10 FTRT-DIRECTORY-BLKS PIC S9(9) BINARY VALUE 0.
10 FTRT-RECORD-FORMAT PIC 9(4) BINARY VALUE 1.
   88 FTRT-RECFM-FIXED VALUE 1.
   88 FTRT-RECFM-VARIABLE VALUE 2.
   88 FTRT-RECFM-FIXED-BLOCKED VALUE 3.
   88 FTRT-RECFM-VARIABLE-BLOCKED VALUE 4.
10 FTRT-RECORD-LENGTH PIC S9(9) BINARY VALUE 0.
10 FTRT-BLOCK-SIZE     PIC S9(9) BINARY VALUE -1.
10 FTRT-ALLOCATION      PIC 9(4) BINARY VALUE 3.
   88 FTRT-ALLOCATION-UNIT-CYL VALUE 1.
   88 FTRT-ALLOCATION-UNIT-BLK VALUE 2.
   88 FTRT-ALLOCATION-UNIT-TRK VALUE 3.
10 FTRT-PRIMARY-ALLOC PIC S9(9) BINARY VALUE 5.
10 FTRT-SECOND-ALLOC  PIC S9(9) BINARY VALUE 2.
```

```

10 FTFT-TEXT-WRAP-INFO PIC 9(4) BINARY VALUE 0.
88 FTFT-TEXT-WRAP VALUE 1.
88 FTFT-FAIL VALUE 2.
88 FTFT-TRUNC VALUE 3.
10 FTFT-CREATE-DIR PIC 9(4) BINARY VALUE 0.
10 FTFT-ISDATA-PERSIST PIC 9(4) BINARY VALUE 0.
10 FTFT-C-DTYPE PIC 9(9) BINARY VALUE 0.
10 FTFT-BUFFNO PIC 9(9) BINARY VALUE 0.
10 FTFRJ-PRIORITY PIC 9(9) BINARY VALUE 3.
10 FTFRJ-RETRIES PIC 9(9) BINARY VALUE 0.
10 FTFRJ-EXP-DATE-TIME PIC 9(9) BINARY VALUE 0.
10 FTFRJ-MQM-MSG-SIZE PIC 9(9) BINARY VALUE 524288.
10 FTFRJ-IS-TRUSTED PIC 9(4) BINARY VALUE 0.
10 FTFRJ-RESERVED PIC 9(4) BINARY VALUE 1.
10 FTFRJ-IS-STAGE-ONLY PIC 9(4) BINARY VALUE 0.
10 FTFRJ-IS-STAGED-FTID PIC 9(4) BINARY VALUE 0.
10 FTFRJ-IS-STAGED-FILE PIC 9(4) BINARY VALUE 0.
10 FTFRJ-CANCEL-MODE PIC 9(4) BINARY VALUE 0.
10 FTFRJ-DELETE-SOURCE PIC 9(4) BINARY VALUE 0.
10 FTFRJ-IMMEDIATE PIC 9(4) BINARY VALUE 0.
10 FTFRJ-XML-FORMAT PIC 9(4) BINARY VALUE 0.
10 FTFRJ-NOTIFY-STATUS PIC 9(4) BINARY VALUE 0.
10 FTFRJ-RECORD-LENGTH PIC 9(9) BINARY VALUE 0.
10 FTFRJ-FILE-TYPE PIC 9(4) BINARY VALUE 0.
10 FTFRJ-FILE-TYPE-400 PIC 9(4) BINARY VALUE 1.
88 FTFRJ-BAD VALUE 0.
88 FTFRJ-TYPE-SAVE VALUE 1.
88 FTFRJ-TYPE-SRC VALUE 2.
88 FTFRJ-TYPE-IFS VALUE 3.
88 FTFRJ-TYPE-QSYSUB VALUE 4.
88 FTFRJ-TYPE-PHYSICAL VALUE 5.
88 FTFRJ-TYPE-LOGICAL VALUE 6.
88 FTFRJ-TYPE-DATABASE VALUE 7.
10 FTFRJ-CREATE-LIBRARY PIC 9(4) BINARY VALUE 0.
10 FTFRJ-LIB-ASP PIC 9(9) BINARY VALUE 1.
10 FTFRJ-FILE-ASP PIC 9(9) BINARY VALUE 1.
10 FTFRJ-C-EXIT-INFO PIC 9(9) BINARY VALUE 0.
.....
* FTF Request alpha fields
.....
10 FTFRS-SPATH PIC X(257) VALUE SPACES.
10 FTFRS-STYPE PIC X(26) VALUE SPACES.
10 FTFRS-PAD-CHARACTER PIC X VALUE SPACES.
10 FTFRS-SOURCETYPE-DATA PIC X(257) VALUE SPACES.
10 FTFRS-DPATH PIC X(257) VALUE SPACES.
10 FTFRS-DTYPE PIC X(26) VALUE SPACES.
10 FTFRS-UNIT-NAME PIC X(6) VALUE SPACES.
10 FTFRS-VOL-SER PIC X(6) VALUE SPACES.
10 FTFRS-MODEL-DATASET PIC X(44) VALUE SPACES.
10 FTFRS-DESTTYPE-DATA PIC X(257) VALUE SPACES.
10 FTFRS-DEST-DIR PIC X(257) VALUE SPACES.
10 FTFRJ-POOL-NAME PIC X(48) VALUE SPACES.
10 FTFRJ-USER-FTFR-IDENTIFIERS.
15 FTFRJ-ID1 PIC X(50) VALUE LOW-VALUES.
15 FTFRJ-ID2 PIC X(50) VALUE LOW-VALUES.
15 FTFRJ-ID3 PIC X(50) VALUE LOW-VALUES.
10 FTFRJ-BROKER-STREAM PIC X(257) VALUE SPACES.
10 FTFRJ-GROUP-NAME PIC X(20) VALUE SPACES.

```

```

10 FTRU-LABEL          PIC X(20)  VALUE LOW-VALUES.
10 FTRU-REPLYQ         PIC X(48)  VALUE LOW-VALUES.
10 FTRU-REPLYQ/MGR     PIC X(48)  VALUE LOW-VALUES.
10 FTRU-NOTIFYDATA     PIC X(40)  VALUE SPACES.
10 FTRU-NOTIFYTYPE     PIC X(25)  VALUE LOW-VALUES.
10 FTRU-SUBSTRING      PIC X(257) VALUE SPACES.
10 FTRA-CCSD          PIC X(5)   VALUE SPACES.
10 FTRA-LIB-TEXT      PIC X(50)  VALUE SPACES.
10 FTRA-FILE-TEXT     PIC X(50)  VALUE SPACES.
10 FTRE-EXIT-INFO.
    15 FTRE-EXIT-COUNT PIC XX    VALUE ZERO.
    15 FTRE-EXIT-LINE OCCURS 10 TIMES.
        20 FTRE-EXIT-NUMBER PIC XX    VALUE ZERO.
        20 FTRE-EXIT-DLL PIC X(16)  VALUE SPACES.
        20 FTRE-EXIT-ENTRY PIC X(16)  VALUE SPACES.
        20 FTRE-EXIT-USERID PIC X(16)  VALUE SPACES.
        20 FTRE-EXIT-PASSWORD PIC X(16)  VALUE SPACES.
        20 FTRE-EXIT-DATA PIC X(256)  VALUE SPACES.
.....
* PING - FTFPing fields
.....
05 FTFP-PING.
    10 FTFP-TIME-OUT      PIC S9(9) BINARY VALUE 0.
    10 FTFP-MSG-SIZE      PIC S9(9) BINARY VALUE 0.
    10 FTFP-PRIORITY      PIC S9(9) BINARY VALUE 0.
    10 FTFP-KBYTE-SWRITTEN PIC S9(9) BINARY VALUE 0.
.....
* STATUS-LIST - Status summary list fields
.....
05 FTFS-STATUS-LIST.
    10 FTFS-ROWS          PIC S9(9) BINARY VALUE 0.
    10 FTFS-ROW-POINTER  USAGE IS POINTER.
    10 FTFS-STAT-PRINT.
        15 FTFS-STAT-PRINT-LINE PIC X(80) OCCURS 20 TIMES.
.....
* STATUS-FILTER - Status filter
.....
05 FTFS-STATUS-FILTER.
    10 FTFS-FTF-ID        PIC X(37)  VALUE SPACES.
    10 FILLER              PIC X(3)   VALUE LOW-VALUES.
    10 FTFS-IGNORE-ACTIVE PIC S9(9) BINARY VALUE 0.
    10 FTFS-IGNORE-COMPLETE PIC S9(9) BINARY VALUE 0.
    10 FTFS-IGNORE-FAILED PIC S9(9) BINARY VALUE 0.
    10 FTFS-IGNORE-CANCELLED PIC S9(9) BINARY VALUE 0.
    10 FTFS-IGNORE-EXPIRED PIC S9(9) BINARY VALUE 0.
    10 FTFS-LQM           PIC X(25)  VALUE SPACES.
    10 FILLER              PIC X(3)   VALUE LOW-VALUES.
    10 FTFS-CASE-SENSITIVE PIC 9(4)  BINARY VALUE 0.
    10 FTFS-OQM           PIC X(25)  VALUE SPACES.
    10 FTFS-ORIGIN-START-TIME PIC X(25)  VALUE SPACES.
    10 FTFS-ORIGIN-END-TIME PIC X(25)  VALUE SPACES.
    10 FTFS-SQM           PIC X(25)  VALUE SPACES.
    10 FTFS-SOURCE-START-TIME PIC X(25)  VALUE SPACES.
    10 FTFS-SOURCE-END-TIME PIC X(25)  VALUE SPACES.
    10 FTFS-SOURCE-FILE-NAME PIC X(256)  VALUE SPACES.
    10 FTFS-DQM           PIC X(25)  VALUE SPACES.
    10 FTFS-TARGET-START-TIME PIC X(25)  VALUE SPACES.
    10 FTFS-TARGET-END-TIME PIC X(25)  VALUE SPACES.

```

```

10 FTFSF-TARGET-FILE-NAME PIC X(256) VALUE SPACES.
10 FTFSF-GROUP-NAME      PIC X(25)  VALUE SPACES.
10 FTFSF-LABEL           PIC X(25)  VALUE SPACES.
10 FILLER                 PIC X(3)   VALUE LOW-VALUES.
.....

```

```

** End of FTF COBOL-DATA, that is passed to FTF COBOL program **
.....

```

## FTFH-HEADER

This section describes all of the variables that are common to the three request types (FTF, FTFPING, and FTFSTAT).

Level	Name	Definition	Description	Command Line Reference
05	FTFH-HEADER		NA	NA
10	FTFH-FUNCTION	PIC X(08)	Tells the FTF COBOL DLL, which request type to process.	NA
10	FTFH-CONFIG-FILE	PIX X(256)	Location of the FTF Configuration File.	-cfile
10	FTFH-MQHANDLE	PIC S9(9) BINARY	MQSeries Handle	NA
10	FTFH-RETURN-CODE1	PIC S9(9) BINARY	FTF Primary Return Code. Examine the contents of this field for feedback on requests.	NA
10	FTFH-RETURN-CODE2	PIC S9(9) BINARY	FTF Secondary Return Code. Examine the contents of this field for feedback on requests.	NA
10	FTFH-ERROR-MESSAGE	PIC X(1024)	Will contain error message in text form, when appropriate. Use in conjunction with FTFH-RETURN-CODE1 and FTFH-RETURN-CODE2.	NA

Level	Name	Definition	Description	Command Line Reference
10	FTFH-FTFID-ASCII	PIC X(37)	The FTF ID in ascii format. Note that all FTF IDs are carried through the system in ascii format.	NA
10	FTFH-FTFID-NATIVE	PIC X(37)	The FTF ID in native format.	NA
10	FTFH-LQM	PIC X(49)	Local Queue Manager	-lqm
10	FTFH-OQM	PIC X(49)	Originating Queue Manager	-oqm
10	FTFH-SQM	PIC X(49)	Source Queue Manager	-sqm
10	FTFH-DQM	PIC X(49)	Destination Queue Manager	-dqm

## Request Fields

FTF Request fields are divided into two numeric and alpha fields. Note the following:

- FTFR = Regular request fields
- FTFRS = Source file attributes
- FTFRT = Target file attributes
- FTFRJ = Job attributes
- FTFRU = User attributes
- FTFRA = AS400 file attributes
- FTFRE = Exit information

Level	Name	Definition	Description	Command Line Reference
05	FTFR-REQUEST		NA	NA
10	FTFR-IS-REPLY	PIC 9(4) BINARY VALUE 0		-wait
10	FTFR-REPLY-WAIT-TIME	PIX 9(9) BINARY VALUE 60	Reply wait time in seconds	-wait
10	FTFRS-IS-STAGED	PIC 9(4) BINARY VALUE 0	Stage the source file	-staged
10	FTF-IS-STAGEPERS	PIC 9(4) BINARY VALUE 0	Stage persistent messages	-stagepersist
10	FTFRS-IS-DATA-PERS	PIC 9(4) BINARY VALUE 0	Internal use only	Internal use only
10	FTFRS-IS-COMPRESSED	PIC 9(4) BINARY VALUE 0	Set compression	-compress
10	FTFRS-IS-DELETE	PIC 9(4) BINARY VALUE 0	Delete source file	-delsrc
10	FTFRS-FILE-TYPE-INFO	PIC 9(4) BINARY VALUE 1	Type of data for the transfer (text or binary)	-type
88	FTFRS-FILE-TYPE-BINARY	VALUE 1	Binary transfer	-type binary
88	FTFRS-FILE-TYPE-TEXT	VALUE 2	Text transfer	-type text
10	FTFRS-RECORD-PADDING	PIC 9(4) BINARY VALUE 1	Set record padding for OS/390 datasets	-recpad
88	FTFRS-PAD	VALUE 1	Pad records	-recpad pad
88	FTFRS-NO-PAD	VALUE 2	Don't Pad records	-recpad nopad
10	FTFRS-C-STYPE	PIC S9(9) BINARY VALUE 0	Length of the DTYPE string	NA
10	FTFRS-BUFFNO	PIC S9(9) BINARY VALUE 0	MVS buffer number	-bufno
10	FTFRT-FILE-MODE-INFO	PIC 9(4) BINARY VALUE 1	Transfer mode	-mode
88	FTFRT-FILE-MODE-CREATE	VALUE 1		-mode
88	FTFRT-FILE-MODE-APPEND	VALUE 2	APPEND	-mode APPEND



Level	Name	Definition	Description	Command Line Reference
88	FTFRT-FILE-MODE-NOREPLACE	VALUE 3	NOREPLACE	-mode NO REPLACE
10	FTFRT-FILE-ORG-INFO	PIC 9(4) BINARY VALUE 1	MVS file organization	-org
88	FTFRT-FILE-ORG-PARTITIONED	VALUE 1	-org PDS	-org
88	FTFRT-FILE-ORG-SEQUENTIAL	VALUE 2	-org PS	-org
10	FTFRT-DIRECTORY-BLKS	PIC S9(9) BINARY VALUE 0	MVS PDS directory blocks	-dirblks
10	FTFRT-RECORD-FORMAT	PIC 9(4) BINARY VALUE 1	MVS record format	-recfmt
88	FTFRT-RECFM-FIXED	VALUE 1	-recfmt F	-recfmt
88	FTFRT-RECFM-VARIABLE	VALUE 2	-recfmt V	-recfmt
88	FTFRT-RECFM-FIXED-BLOCKED	VALUE 3	-recfmt FB	-recfmt
88	FTFRT-RECFM-VARIABLE-BLOCKED	VALUE 4	-recfmt VB	-recfmt
10	FTFRT-RECORD-LENGTH	PIC S9(9) BINARY VALUE 0	MVS logical record length	-lrecl
10	FTFRT-BLOCK-SIZE	PIC S9(9) BINARY VALUE -1	MVS block size	-blksize
10	FTFRT-ALLOCATION	PIC 9(4) BINARY VALUE 3	MVS allocation unit	-alcunit
88	FTFRT-ALLOCATION-UNIT-CYL	VALUE 1	-alcunit CYL	-alcunit
88	FTFRT-ALLOCATION-UNIT-BLK	VALUE 2	-alcunit BLK	-alcunit
88	FTFRT-ALLOCATION-UNIT-TRK	VALUE 3	-alcunit TRK	-alcunit
10	FTFRT-TEXT-WRAP-INFO	PIC 9(4) BINARY VALUE 0	Record wrapping	-recwrap
88	FTFRT-TEXT-WRAP	VALUE 1	-recwrap wrap	-recwrap
88	FTFRT-FAIL	VALUE 2	-recwrap fail	-recwrap
88	FTFRT-TRUNC	VALUE 3	-recwrap trunc	-recwrap

Level	Name	Definition	Description	Command Line Reference
10	FTFRT-CREATE-DIR	PIC 9(4) BINARY VALUE 0	Create destination directories	-mkdirs
10	FTFRT-ISDATA-PERSIST	PIC 9(4) BINARY VALUE 0	Internal use only	NA
10	FTFRT-C-DTYPE	PIC S9(9) BINARY VALUE 0	Length of the DTYPE string	-dtype
10	FTFRT-BUFFNO	PIC S9(9) BINARY VALUE 0	MVS buffer number	-bufno
10	FTFRJ-PRIORITY	PIC S9(9) BINARY VALUE 3	Transaction priority	-priority
10	10 FTFRJ-RETRIES	PIC S9(9) BINARY VALUE 0	Internal use only	NA
10	FTFRJ-EXP-DATE-TIME	PIC S9(9) BINARY VALUE 0	Expiry time for transaction	-expiry
10	FTFRJ-MQM-MSG-SIZE	PIC S9(9) BINARY VALUE 524288	Maximum message size	-msgsize
10	FTFRJ-IS-TRUSTED	PIC 9(4) BINARY VALUE 0	Trusted transfer request	-trusted
10	FTFRJ-RESERVED	PIC 9(4) BINARY VALUE 1	Internal use only	NA
10	FTFRJ-IS-STAGE-ONLY	PIC S9(4) BINARY VALUE 0	Only stage the source file	-stageonly
10	FTFRJ-IS-STAGEDFID	10 PIC S9(4) BINARY VALUE 0	Transaction ID of the staged file	-ftfid
10	FTFRJ-IS-STAGED-FILE	PIC S9(4) BINARY VALUE 0	Retrieve file from staging area	-fromstage
10	FTFRJ-CANCEL-MODE	PIC S9(4) BINARY VALUE 0	Preemptive cancel of transaction	-cancel
10	FTFRJ-DELETE-SOURCE	PIC S9(4) BINARY VALUE 0	Delete the source file	-delsrc
10	FTFRJ-IMMEDIATE	PIC S9(4) BINARY VALUE 0	Immediate transfer mode	-immed
10	FTFRJ-XML-FORMAT	PIC S9(4) BINARY VALUE 0	Internal use only	Internal use only

Level	Name	Definition	Description	Command Line Reference
10	FTFRU-NOTIFYSTATUS	PIC S9(4) BINARY VALUE 0	Notify status {FAILURE, SUCCESS, NONSUCCESS}	-notify
10	FTFRA-RECORD-LENGTH	PIC S9(9) BINARY VALUE 0	AS400 record length	-rcdlen
10	FTFRA-FILE-TYPE-400	PIC 9(4) BINARY VALUE 1	AS400 file type	-as400ft
88	FTFRA-BAD	VALUE 0	Internal use only	-as400ft
88	FTFRA-TYPE-SAVE	VALUE 1	-as400ft savf	-as400ft
88	FTFRA-TYPE-SRC	VALUE 2	-as400ft srcpf	-as400ft
88	FTFRA-TYPE-IFS	VALUE 3	Internal use only	NA
88	FTFRA-TYPE-QSYSLIB	VALUE 4	Internal use only	NA
88	FTFRA-TYPE-PHYSICAL	VALUE 5	Internal use only	NA
88	FTFRA-TYPE-LOGICAL	VALUE 6	Internal use only	NA
88	FTFRA-TYPE-DATABASE	VALUE 7	Internal use only	NA
10	FTFRA-CREATE-LIBRARY	PIC 9(4) BINARY VALUE 0	AS400 Create library	-crtlib
10	FTFRA-LIB-ASP	PIC 9(9) BINARY VALUE 1	AS400 library auxiliary storage pool	-libasp
10	FTFRA-FILE-ASP	PIC 9(9) BINARY VALUE 1	AS400 file auxiliary storage pool	-fileasp
10	FTFRE-C-EXIT-INFO	PIC S9(9) BINARY VALUE 0	Number of exits included. Used for the occurs clause	NA
10	FTFRS-SPATH	PIC X(257) VALUE SPACES	Source fullpath filename	-spath
10	FTFRS-STYLE	PIC X(26) VALUE SPACES	Source file type	-stype
10	FTFRS-PAD-CHARACTER	PIC X VALUE SPACES	Record padding character	-padchar
10	FTFRS-SOURCE TYPE- DATA	PIC X(257) VALUE SPACES	Source file type data string	-stype
10	FTFRT-DPATH	PIC X(257) VALUE SPACES	Destination fullpath filename	-dpath

Level	Name	Definition	Description	Command Line Reference
10	FTFRT-DTYPE	PIC X(26) VALUE SPACES	Destination file type	-dtype
10	FTFRT-UNIT-NAME	PIC X(8) VALUE SPACES	MVS unit name	-unit
10	FTFRT-VOL-SER	PIC X(6) VALUE SPACES	MVS volume serial number	-volser
10	FTFRT-MODEL-DATASET	PIC X(44) V VALUE SPACES	MVS GDG model dataset	-model
10	FTFRT-DESTTYPE-DATA	PIC X(257) VALUE SPACES	Destination file type data string	-dtype
10	FTFRT-DEST-DIR	PIC X(257) VALUE SPACES	Internal use only	NA
10	FTFRJ-POOL-NAME	PIC X(48) VALUE SPACES	Data pool	-pool
10	FTFRJ-USER-FTFR-IDENTIFIERS		User defined Identifier	NA
15	FTFRJ-ID1	PIC X(50) VALUE LOW-VALUES	User defined Identifier	-id1
15	FTFRJ-ID2	PIC X(50) VALUE LOW-VALUES	User defined Identifier	-id2
15	FTFRJ-ID3	PIC X(50) VALUE LOW-VALUES	User defined Identifier	-id3
10	FTFRJ-BROKER-STREAM	PIC X(20) VALUE SPACES	Internal use only	NA
10	FTFRU-GROUP-NAME	PIC X(257) VALUE SPACES	Internal use only	NA
10	FTFRU-LABEL	PIC X(20) VALUE LOW-VALUES	Transaction label	-label
10	FTFRU-REPLYQ	PIC X(48) VALUE LOW-VALUES	Queue to send reply message	-replyq
10	FTFRU-REPLYQMGR	PIC X(48) VALUE LOW-VALUES	Queue manager to send reply message	-replyqmgr
10	FTFRU-NOTIFYDATA	PIC X(40) VALUE SPACES	Notification Data	-notifydata
10	FTFRU-NOTIFYTYPE	PIC X(25) VALUE LOW-VALUES	Type of notification	-notifytype

Level	Name	Definition	Description	Command Line Reference
10	FTFRU-SUBSTRING	PIC X(257) VALUE SPACES	Internal use only	NA
10	FTFRA-CCSID	PIC X(5) VALUE SPACES	AS400 coded character set	-ccsid
10	FTFRA-LIB-TEXT	PIC X(50) VALUE SPACES	AS400 library text	-libtxt
10	FTFRA-FILE-TEXT	PIC X(50) VALUE SPACES	AS400 file text	-filetxt
10	FTFRE-EXIT-INFO			NA
15	FTFRE-EXIT-COUNT	PIC XX VALUE ZERO	Number of exits	NA
15	FTFRE-EXIT-LINE	OCCURS 10 TIMES	NA	NA
20	FTFRE-EXIT-NUMBER	PIC XX VALUE ZERO	Exit number	-exit
20	FTFRE-EXIT-DLL	PIC X(16) VALUE SPACES	Dll library	-exit dll
20	FTFRE-EXIT-ENTRY	PIC X(16) VALUE SPACES	Entry point	-entry
20	FTFRE-EXIT-USERID	PIC X(16) VALUE SPACES	Internal use only	NA
20	FTFRE-EXIT-PASSWORD	PIC X(16) VALUE SPACES	Internal use only	NA
20	FTFRE-EXIT-DATA	PIC X(256) VALUE SPACES	Data string to be passed to exit	-exitdata

## FTFPing Fields

FTFPing fields are used to issue FTFPing requests. The variables are similar to the FTFPING command line interface.

Level	Name	Definition	Description	Command Line Reference
05	FTF-PING		NA	NA
10	FTFP-TIMEOUT	PIC S9(9) BINARY VALUE 0	Time out value in seconds	-timeout
10	FTFP-MSGSIZE	PIC S9(9) BINARY VALUE 0	Message size (kilobytes)	-msgsize
10	FTFP-PRIORITY	PIC S9(9) BINARY VALUE 0	Priority	-priority
10	FTFP-KBYTESWRITTEN	PIC S9(9) BINARY VALUE 0	Number of kilobytes written as a result of the message size. This field is set after the ping completes successfully	NA

## FTFStatus Summary List Fields

FTFS-STATUS-LIST is used by CQDI to pass status information for requests to the user.

Level	Name	Definition	Description	Command Line Reference
05	FTF-STATUS-LIST		NA	NA
10	FTFS-ROWS	PIC S9(9) BINARY VALUE 0	Number of status “rows” returned.	NA
10	FTFS-ROW-POINTER	USAGE IS POINTER	A pointer to the returned status information	NA
10	FTFS-STAT-PRINT			
10	FTFS-STAT-PRINT-LINE	PIC X(80) occurs 20 times	Status records in display format	NA

## FTF STAT Filter Fields

FTFS-STATUS-FILTER fields are used to filter out only the status records that the user requests.

Level	Name	Definition	Description	Command Line Reference
05	FTFS-STATUS-FILTER		NA	NA
10	FTFSF-FTF-ID	PIC X(37)	The ftfid to retrieve.	-ftfid
10	FTFSF-IGNORE-ACTIVE	PIC S9(9) BINARY VALUE 0	Ignore all active requests	-status
10	FTFSF-IGNORE-COMPLETE	PIC S9(9) BINARY VALUE 0	Ignore all completed requests	-status
10	FTFSF-IGNORE-FAILED	PIC S9(9) BINARY VALUE 0	Ignore all failed requests	-status
10	FTFSF-IGNORE-CANCELLED	PIC S9(9) BINARY VALUE 0	Ignore all cancelled requests	-status
10	FTFSF-IGNORE-EXPIRED	PIC S9(9) BINARY VALUE 0	Ignore all expired requests	-status
10	FTFSF-LQM	PIC X(25) VALUE SPACES	Requesting queue manager	-rqm
10	FTFSF-CASE-SENSITIVE	PIC 9(4) BINARY VALUE 0	Specifies whether the filter options are case sensitive for -spath and -dpath	-i
10	FTFSF-OQM	PIC X(25) VALUE SPACES	Originating queue manager	-oqm
10	FTFSF-ORIGIN-START-TIME	PIC X(25) VALUE SPACES	FTF Manager start time	NA
10	FTFSF-ORIGIN-END-TIME	PIC X(25) VALUE SPACES	FTF Manager end time	NA
10	FTFSF-SQM	PIC X(25) VALUE SPACES	Source queue manager	-sqm
10	FTFSF-SOURCE-START-TIME	PIC X(25) VALUE SPACES	FTF Sender Start Time	NA
10	FTFSF-SOURCE-END-TIME	PIC X(25) VALUE SPACES	FTF Sender End Time	NA

Level	Name	Definition	Description	Command Line Reference
10	FTFSF-SOURCE-FILE-NAME	PIC X(256) VALUE SPACES	Source fullpath filename	-spath
10	FTFSF-DQM	PIC X(25) VALUE SPACES	Destination queue manager	-dqm
10	FTFSF-TARGET-START-TIME	PIC X(25) VALUE SPACES	Destination fullpath filename	-dpath
10	FTFSF-TARGET-END-TIME	PIC X(25) VALUE SPACES	FTF Receiver start time	NA
10	FTFSF-TARGET-FILE-NAME	PIC X(256) VALUE SPACES	FTF Receiver end time	NA
10	FTFSF-GROUP-NAME	PIC X(25) VALUE SPACES	Internal use only	NA
10	FTFSF-LABEL	PIC X(25) VALUE SPACES	User-supplied identifier	-label

## Issuing FTF Transfer Requests

The sample COBOL program, “FTFCREQ” illustrates how to utilize FTFCOBOL to issue transfer requests. The sample program can be found in the COBSAMP library. There are 2 arguments to the program, LQM and CFILE. Both are required arguments on any CQDI request. The LQM is used to specify the name of the Queue Manager to connect to MQSeries, and CFILE is used to specify the location of the CQDI configuration file. We will examine code snippets to explain important aspects of issuing CQDI requests from COBOL.

In order to setup a transfer request, the appropriate variables must be set in the COPYBOOK.



```

*****
* Setup the FTF request parameters
*****
*
→ Always initialize the return codes
  MOVE ZERO TO FTFH-RETURN-CODE1
  MOVE ZERO TO FTFH-RETURN-CODE2
  MOVE SPACES TO FTFH-ERROR-MESSAGE
→ Identify to CQDI that this is a transfer request
  MOVE 'FTFREQ' TO FTFH-FUNCTION
→ Pass the HQ Handle after you have connected
  MOVE CONNECTION-HANDLE TO FTFH-MQHANDLE
→ Setup the LQM, OQM, DQM and SQM (corresponds to -lqm, -oqm, -sqm and -dqm)
  MOVE QUEUE-MANAGER-NAME TO FTFH-LQM
  MOVE QUEUE-MANAGER-NAME TO FTFH-OQM
  MOVE QUEUE-MANAGER-NAME TO FTFH-SQM
  MOVE QUEUE-MANAGER-NAME TO FTFH-DQM
→ Specify the source and target files (corresponds to -spath and -dpath)
  * Set up Source Path
  MOVE 'DEVXXX.TEST.FILE' TO FTFRS-SPATH
  * Set up Target path
  MOVE 'DEVXXX.TEST.FILE.OUT' TO FTFRT-DPATH
→ Specify a label (corresponds to -label)
  * Set up Label
  MOVE 'TEST API LABEL' TO FTFRU-LABEL
→ Specify the amount of time to wait (Corresponds to -wait 60)
  * Set up Reply fields to wait 60 secs for a reply
  MOVE 1 TO FTFR-IS-REPLY
  MOVE 60 TO FTFR-REPLY-WAIT-TIME
→ Specify DCB attributes (Corresponds to -lrecl, -blksize, -unit, -volser, -org, -recfmt,
-alcunit, -primary, -secondary)
  * Set up MVS specific fields
  * MOVE 80 TO FTFRT-RECORD-LENGTH
  * MOVE 3120 TO FTFRT-BLOCK-SIZE
  * MOVE 'SYSDA' TO FTFRT-UNIT-NAME
  * MOVE 'GEN050' TO FTFRT-VOL-SER
  * MOVE 2 TO FTFRT-FILE-ORG-INFO
  * MOVE 3 TO FTFRT-RECORD-FORMAT
  * MOVE 3 TO FTFRT-ALLOCATION
  * MOVE 5 TO FTFRT-PRIMARY-ALLOC
  * MOVE 2 TO FTFRT-SECOND-ALLOC
  *
→ If compression is desired, then both of these fields must be populated as true
  * Set up Compression if needed - set both fields
  * MOVE 1 TO FTFRS-IS-COMPRESSED
  * MOVE 1 TO FTFRT-IS-COMPRESSED
→ In order to specify exits initialise FTFRE-EXIT-INFO and specify the number of exits to
be called
  * Set up Exit parameters
  MOVE SPACES TO FTFRE-EXIT-INFO
  MOVE 6 TO FTFRE-EXIT-COUNT
→ In the following example, we are specifying to execute exits on NT which will execute
the program "NOTEPAD" (corresponds to -exit, -exitdll, -exitentry, -exitdata)
  MOVE 3 TO FTFRE-EXIT-NUMBER(1)
  MOVE 'FTFEXALL' TO FTFRE-EXIT-DLL(1)
  MOVE 'FTFMgrPre3' TO FTFRE-EXIT-ENTRY(1)
  MOVE 'notepad d:\ftf\ex3.txt' TO FTFRE-EXIT-DATA(1)
  MOVE 4 TO FTFRE-EXIT-NUMBER(2)
  MOVE 'FTFEXALL' TO FTFRE-EXIT-DLL(2)
  MOVE 'FTFMgrPost4' TO FTFRE-EXIT-ENTRY(2)
  MOVE 'notepad d:\ftf\ex4.txt' TO FTFRE-EXIT-DATA(2)
  MOVE 5 TO FTFRE-EXIT-NUMBER(3)
  MOVE 'FTFEXALL' TO FTFRE-EXIT-DLL(3)
  MOVE 'FTFSdrPre5' TO FTFRE-EXIT-ENTRY(3)
  MOVE 'notepad d:\ftf\ex5.txt' TO FTFRE-EXIT-DATA(3)
  MOVE 6 TO FTFRE-EXIT-NUMBER(4)

```

```

MOVE 'FTFEXALL'          TO FTFRE-EXIT-DLL(4)
MOVE 'FTF3drPost6'       TO FTFRE-EXIT-ENTRY(4)
MOVE 'notepad d:\ftf\ex6.txt' TO FTFRE-EXIT-DATA(4)
MOVE 7                   TO FTFRE-EXIT-NUMBER(5)
MOVE 'FTFEXALL'          TO FTFRE-EXIT-DLL(5)
MOVE 'FTFRcvPre7'        TO FTFRE-EXIT-ENTRY(5)
MOVE 'notepad d:\ftf\ex7.txt' TO FTFRE-EXIT-DATA(5)
MOVE 8                   TO FTFRE-EXIT-NUMBER(6)
MOVE 'FTFEXALL'          TO FTFRE-EXIT-DLL(6)
MOVE 'FTFRcvPost8'       TO FTFRE-EXIT-ENTRY(6)
MOVE 'notepad d:\ftf\ex8.txt' TO FTFRE-EXIT-DATA(6)

```

→ After the copybook is filled out, the request can be submitted, here the results are displayed

```

*****
* Call the FTF COBOL API                                     *
*****
*
  CALL 'FTFCOBOL' USING BY REFERENCE FTF COBOL-DATA.
  DISPLAY '*****'
  DISPLAY '* RESULT FROM THE FTF COBOL API *'
  DISPLAY '*****'
  DISPLAY 'RC1          = ' FTFH-RETURN-CODE1
  DISPLAY 'RC2          = ' FTFH-RETURN-CODE2
  DISPLAY 'MSG          = ' FTFH-ERROR-MESSAGE(1:60)
  DISPLAY 'FTFID ASCII  = ' FTFH-FTFID-ASCII
  DISPLAY 'FTFID NATIVE = ' FTFH-FTFID-NATIVE
*

```

## Issuing FTF Ping Requests

The sample COBOL program, “FTFCPING” illustrates how to utilize FTF COBOL to issue transfer requests. The sample program can be found in the COBSAMP library. There are 3 arguments to the program, LQM, DQM and CFILE. All three are required arguments for this program. The LQM is used to specify the name of the Queue Manager to connect to MQSeries, and CFILE is used to specify the location of the CQDI configuration file. DQM is used to specify the target Queue Manager to ping. We will examine code snippets to explain important aspects of issuing CQDI requests from COBOL.

```
→ Specify the FTFPing values
*****
* Setup the FTFPING request
*****
*
      MOVE 5              TO FTFP-TIMEOUT
      MOVE 100            TO FTFP-MSGSIZE
      MOVE 5              TO FTFP-PRIORITY

→ Tell FTF COBOL that this is an FTFPing request
      MOVE 'FTFPING'      TO FTFH-FUNCTION
*

→ Call the FTF COBOL API and DISPLAY the results
      CALL 'FTFCOBOL' USING BY REFERENCE FTF COBOL-DATA.
      DISPLAY '*****'
      DISPLAY '* RESULT FROM THE FTF COBOL API *'
      DISPLAY '*****'
      DISPLAY 'RC1 = ' FTFH-RETURN-CODE1
      DISPLAY 'RC2 = ' FTFH-RETURN-CODE2
      DISPLAY 'MSG = ' FTFH-ERROR-MESSAGE(1:60)
```

## Issuing FTF Status Requests

The sample COBOL program, “FTFCSTAT” illustrates how to utilize FTF COBOL to issue status requests. The sample program can be found in the COBSAMP library. There are 2 arguments to the program, LQM, and CFILE. Both are required arguments for this program. The LQM is used to specify the name of the Queue Manager to connect to MQSeries, and CFILE is used to specify the location of the CQDI configuration file. We will examine code snippets to explain important aspects of issuing CQDI requests from COBOL.

```

→ Setup the COPYBOOK to issue a status request, by not specifying any filter options,
all status records will be returned
→ Tell FTF COBOL to issue an FTFStat request

      MOVE 'FTFSTAT'          TO FTFH-FUNCTION
*
→ Call FTF COBOL to issue the FTFStat request

      CALL 'FTFCOBOL' USING BY REFERENCE FTF COBOL-DATA.
      DISPLAY '*****'
      DISPLAY '* RESULT FROM THE FTF COBOL API *'
      DISPLAY '*****'
      DISPLAY 'RC1 = ' FTFH-RETURN-CODE1
      DISPLAY 'RC2 = ' FTFH-RETURN-CODE2
      DISPLAY 'MSG = ' FTFH-ERROR-MESSAGE(1:60)
*
→ Print the results
      DISPLAY 'Total Rows = ' FTFH-ROWS
      DISPLAY 'Pointer = ' FTFH-ROW-POINTER
      IF FTFH-ROWS = ZERO
          DISPLAY 'No status to print'
          GO TO EC099-EXIT
      END-IF
      SET ADDRESS OF WS-STAT-INFO TO FTFH-ROW-POINTER
      PERFORM EC100-FTFSTAT-ROWS
          VARYING I FROM 1 BY 1
          UNTIL I > FTFH-ROWS
*
      -
      EC099-EXIT.
      EXIT.
*
      EC100-FTFSTAT-ROWS SECTION.
      *****
      * Process each Status Row - one Row per request *
      *****
*
      DISPLAY ' '
      DISPLAY 'STATUS MESSAGE RECORD - ' I ' START *****'
      MOVE WS-STAT-ENTRY(1) TO FTFH-STAT-PRINT
      PERFORM EC200-FTFSTAT-PRINT
          VARYING J FROM 1 BY 1
          UNTIL J > 20
          OR FTFH-STAT-PRINT-LINE(J) = SPACES OR LOW-VALUES
      DISPLAY 'STATUS MESSAGE RECORD - ' I ' END *****'
*
      -
      EC199-EXIT.
      EXIT.
*
      EC200-FTFSTAT-PRINT SECTION.
      *****
      * Print the status messages *
      *****
*
      DISPLAY FTFH-STAT-PRINT-LINE(J)
*
      -
      EC299-EXIT.
      EXIT.
*

```

---

# C API Reference

This chapter describes the Tivoli Data Exchange (TDE) C API, which allows you to leverage TDE for custom business solutions. The API functions use data structures to pass much of the required information. The following information is listed for each API function:

- Description
- Prototype
- Parameters
- Related data types
- Usage notes, if applicable

This chapter contains the following sections:

Section	Page
FTFCancel	183
FTFPing	185
FTFReq	187
FTFShutdown	192
FTFStage	194
FTFStatusDelete	196
FTFStatusFreeDetailList	198
FTFStatusFreeSummaryList	200
FTFStatusGetDetailList	202
FTFStatusGetSummaryList	204
FTFSubmitStatusMsg	207

---

### Note:

For information on linking 'C' applications in the OS/390 environment, see Appendix A, "Security Authorization Exit JCL." Although the sample JCL listed in those pages is presented for COBOL applications, you can also use it for 'C' applications.

---

# FTFCancel

## Description

This API cancels a data-transfer request that has been submitted to the TDE subsystem. The cancel request must be submitted to the same TDE Manager as the initial data-transfer request. FTFCancel submits a cancel request to the system and does not wait for a response stating that the cancel occurred. For additional status information, a query can be performed using the TDE status messages.

## General Prototype

```
#include "ftfc.h"

FTFVOID FTFCancel(MQHCONN hQM,
FTFCancelInfo *pCancelInfo,
FTFCA *pftfca);
```

## OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFCAN(MQHCONN hQM,
FTFCancelInfo *pCancelInfo,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFCAcancel API's parameters.

Parameter (Data Type)	Type	Description
hQM (MQHCONN) - input	Connection handle	Represents the connection to the queue manager. The value of MQHconn was returned by a previous MQCONN call.
pCancelInfo (FTFCAcancelInfo *) -input	Pointer to FTFCAcancelInfo data structure	Represents a properly formatted data structure used by the FTFCAcancel API. This data structure contains all of the attributes and input criteria to cancel a request that has already been submitted to the TDE subsystem.
pftfca (FTFCA *) - output	Pointer to TDE reply message data structure	Represents an output area that returns properly formatted API reply information. This information includes the return codes and optional messages.

## Related Data Structures

- FTFCA (page 215)
- FTFCAcancelInfo (page 217)



# FTFPing

## Description

This API allows you to ping specified the TDE components. The ping message starts at the lqm, the machine from which you are currently working. It is sent to the TDE Manager, as identified by the oqm. The TDE Manager sends it to the TDE Sender, as identified by the sqm. The TDE Sender sends it to the TDE Receiver, as identified by the dqm. The TDE Receiver then sends an acknowledgment to the TDE Manager.

The FTFPing API allows you to ensure that all TDE components involved in a data transfer are available before you start the transfer. You can also use FTFPing with the message size setting to test various message size values until you determine the optimal message size setting.

## General Prototype

```
#include "ftfc.h"

FTFVOID FTFPing (MQHCONN hQM,
FTFPingInfo *pPingInfo,
FTFCA *pftfca);
```

## OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFPING (MQHCONN hQM);
FTFPingInfo *pPingInfo,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFPing API's parameters.

Parameter (Data Type)	Type	Description
hQM (MQHCONN) - input	Connection handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pPingInfo (FTFPingInfo *) -input	Pointer to a PingInfo data structure	Represents a properly formatted data structure used by the FTFPing call. This data structure contains all of the attributes and input criteria to submit a ping message.
pftfca (FTFCA *) - output	Pointer to TDE reply message data structure	Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.  <b>Valid Values</b> - A valid FTFCA pointer, null.

## Related Data Structures

- FTFCA (page 215)
- FTFPingInfo (page 251)

# FTFReq

## Description

This API generates an FTF request transaction. It allows you to take advantage of all transfer-related options, including:

- Specifying queue managers for the transfer.
- Invoking exits to handle installation-specific processing.
- Using connectors to handle input and output from a transfer.
- Automatically converting data to and from formats that FTF does not otherwise allow.
- Specifying a queue pool to make transfers more effective.

The FTFJobInfo and FTFUserInfo data structures that are passed to FTFReq contain data elements to implement the publish/subscribe function. Refer to the “C Data Structures” chapter for more information.

A publish transaction from the FTFPUB command is PUBLISH\_SUBMITTED. The FTFReq API looks to see if the new field isPublish in the FTFJobInfo data structure is set to true. If this is the case, the FTFReq API submits a status message. These status messages are required for the FTFStat process to limit the status information returned from publish-related transactions or from normal transactions.

Various options dictate how this API returns control to the calling application. For example, it can be asynchronous in nature and return immediately or it can wait until a response is received from the FTF subsystem before returning control to the application.

---

### Note:

When a data-transfer request fails and the file mode for the FTF Receiver is set to “Create” or “Replace”, the FTF Receiver deletes the file. If the data-transfer request fails before the FTF Receiver starts processing to the target file, cleanup is not required. However, if the mode is set to “Create” or “Replace”, and the FTF Receiver has started writing data to the target file and the data-transfer request fails, the FTF Receiver deletes the file.

---

General Prototype

```
#include "ftfc.h"

FTFVOID FTFReq(MQHCONN hQM,
FTFRequestMsgInfo *pRequestInfo,
FTFCA *pftfca);
```

OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFREQ(MQHCONN hQM,
FTFRequestMsgInfo *pRequestInfo,
FTFCA *pftfca);
```

Parameters

The following table lists and describes the FTFReq API’s parameters.

Parameter (Data Type)	Type	Description
hQM (MQHCONN) - input	Connection Handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pRequest (FTFRequestMsgInfo *) - input	Pointer to a Request Message data structure	Represents a properly formatted data structure to be used by the FTFReq API call. This data structure contains all the attributes and input criteria for the TDE data transfer, the details for delivery, and the criteria of the source and target file.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.

Related Data Structures

- FTFAS400FileInfo (page 213)
- FTFCA (page 215)
- FTFExitInfo (page 225)

- FTFExitJobInfo (page 229)
- FTFQMgrsInfo (page 254)
- FTFRequestMsgInfo (page 256)
- FTFSourceFileInfo (page 263)
- FTFTargetFileInfo (page 304)
- FTFUserInfo (page 308)

## **FTFReq API**

The FTFReq API is modified slightly in order to implement the publish/subscribe process. This API call initiates publish transactions with the addition of some new data elements in a data structure.

The FTFRequestMsgInfo structure that is passed to FTFReq is updated with the three new parameters:

- `isPublish` – Contains the indicator that this message is a publish transaction.
- `pStream` – Contains the name of the broker stream on which to publish. If the broker stream is not specified on the FTTPUB command line, the broker stream queue defaults to the default broker stream `SYSTEM.BROKER.STREAM.DEFAULT`.
- `pSubString` – Contains the subscription topic.

## **Status Message Changes**

A publish transaction from the FTTPUB command is `PUBLISH_SUBMITTED`. The FTFReq API looks to see if the new field `isPublish` is set to true. If this is the case, the FTFReq API submits the above status message. These new status messages are required for the FTFStat process to limit the status information returned from publish related transactions or from normal transactions.

## **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFReq(MQHCONN hQM,
FTFRequestMsgInfo *pRequestInfo,
FTFCA *pftfca);
```

## **MVS Prototype**

```
#include "ftfc.h"

FTFVOID FTFREQ(MQHCONN hQM,
```

```
FTFRequestMsgInfo *pRequestInfo,  
FTFCA *pftfca);
```

## Parameters

The following table contains *FTFReq*'s parameters.

Parameter Name	Type	Description
hQM (MQHCONN) - input	Connection Handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pRequest (FTFRequestMsgInfo *) - input	Pointer to a Request Message data structure	Represents a properly formatted data structure to be used by the <i>FTFReq</i> API call. This data structure contains all the attributes and input criteria for the FTF/MQ file transfer, the details for delivery, and the criteria of the source and target file.
pftfca (FTFCA *) - output	Pointer to an FTF Reply Message data structure	Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.

## **FTFShutdown**

### **Description**

This API shuts down specified TDE components. A shutdown request is submitted and can target all components or a specific component (TDE Manager, TDE Sender, or TDE Receiver) on a local or remote queue. Various options dictate how this API returns control to the calling application. For example, the API call can be asynchronous in nature and return immediately or wait until a response is received from the TDE subsystem.

The TDE components being shut down do not process the shutdown message while they are processing a data-transfer request. For example, if the TDE Sender is transferring a large file, it does not read the shutdown message from the control queue until it finishes processing its data. Ending the TDE Sender by other means may cause the failure of the data-transfer request.

### **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFShutdown(MQHCONN hQM,
FTFShutdownInfo *pShutdownInfo,
FTFShutdownReply *pShutdownReply,
FTFCA *pftfca);
```

### **OS/390 Prototype**

```
#include "ftfc.h"

FTFVOID FTFEND(MQHCONN hQM,
FTFShutdownInfo *pShutdownInfo,
FTFShutdownReply *pShutdownReply,
FTFCA *pftfca);
```



## Parameters

The following table lists and describes the FTFShutdown API's parameters.

Parameter (Data Type)	Type	Description
hQM (MQHCONN) - input	Connection handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pShutdownInfo (FTFShutdownInfo *) -input	Pointer to input data structure	Represents the input data structure that contains all the required and optional attributes that pertain to the shutdown options.
pShutdownReply (FTFShutdownReply *) -output	Pointer to output data structure	Represents the output data structure that contains all the output and results from the shutdown API request.
pftfca (FTFCA *) - output	Pointer to TDE reply message data structure	Represents an output area that returns properly formatted API reply information. This includes the return codes and optional messages.

## Related Data Structures

- FTFCA (page 215)
- FTFShutdownInfo (page 259)
- FTFShutdownReply (page 261)

## **FTFStage**

### **Description**

This API allows you either to query the items in the staging queue for the specified TDE Sender, or to purge the items in a specified TDE Sender's staging area. If the function is used to query the staged items, the information returned from the query is placed in the FTFStagedList data structure.

### **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFStage
(MQHCONN hQM,
FTFStageMsgInfo *pRequest,
FTFStagedListMsgInfo *pStagedListMsgInfo,
FTFCA *pftfca);
```

### **OS/390 Prototype**

```
#include "ftfc.h"

FTFVOID FTFSTAGE
(MQHCONN hQM,
FTFStageMsgInfo *pRequest,
FTFStagedListMsgInfo *pStagedListMsgInfo,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFStage API's parameters.

Parameter (Data Type)	Type	Description
hQM (MQHCONN) - input	Connection handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pRequest (FTFStageMsgInfo *) -input	Pointer to input data structure	Represents the input data structure that contains the information required for the staging operation to function.
pStagedListMsgInfo (FTFStagedListMsgInfo *) -output	Pointer to output data structure	Represents the output data structure that contains all data structures with staging query information.
pftfca (FTFCA *) - output	Pointer to TDE reply message data structure	Represents an output area that returns properly formatted API reply information. This includes the return codes and optional messages.

## Related Data Structures

- FTFCA (page 215)
- FTFStagedListMsgInfo (page 266)
- FTFStagedTransactionMsgInfo (page 267)
- FTFStageMsgInfo (page 269)

# FTFStatusDelete

## Description

This API purges status messages from the MQSeries environment. It removes all status messages associated with a specific FTFID from the TDE status subsystem. You should call this API periodically to prevent the status queues from filling up. For example, you can use the FTFStatusDelete API with the FTFStatusGetSummaryList API to purge all status records associated with data transfers that are older than a given date.

## General Prototype

```
#include "ftfc.h"

FTFVOID FTFStatusDelete(MQHCONN hQM,
FTFStatusDeleteInfo *info,
FTFCA *pftfca);
```

## OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFSD(MQHCONN hQM,
FTFStatusDeleteInfo *info,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFStatusDelete API’s parameters.

Parameter (Data Type)	Type	Description
HQM (MQHCONN) - input	Connection Handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.

Parameter (Data Type)	Type	Description
info (FTFStatusDelete Info *)	Pointer to a status delete data structure	Represents the input area that determines the status records being deleted.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.

## Related Data Structures

- FTFCA (page 215)
- FTFStatusDeleteInfo (page 288)

# FTFStatusFreeDetailList

## Description

This API frees the memory previously allocated by the FTFStatusGetDetail API. This API accepts a pointer to the FTFStatDetailList data structure and frees all memory associated with the pointer. This call should be used in a program that runs continuously and calls the FTFStatusGetDetail API.

## General Prototype

```
#include "ftfc.h"

FTFVOID FTFStatusFreeDetailList
(FTFStatDetailList *pStatDetailList,
 FTFCA *pftfca);
```

## OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFSFDL
(FTFStatDetailList *pStatDetailList,
 FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFStatusFreeDetailList API's parameters.

Parameter (Data Type)	Type	Description
pStatDetailList (FTFStatDetailList *) - input	Pointer to a Status Detail List data structure	Represents the data structure that contains all the detailed status information for a complete TDE data transfer.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.

## Usage Notes

- Use a temporary pointer to pRow element in the pStatDetailList data structure while processing the status information returned by the FTFStatusGetDetail API.
- The FTFStatusFreeDetailList API call fails if you increment the pRow data element before you call the API.

## Related Data Structures

- FTFCA (page 215)
- FTFStatDetail (page 271)
- FTFStatDetailList (page 281)

## **FTFStatusFreeSummaryList**

### **Description**

This API frees the memory previously allocated by the FTFStatusGetSummary API. It accepts a pointer to the FTFStatSummaryList data structure and frees all memory associated with the pointer. This call should be used in a program that runs continuously and calls the FTFStatusGetSummary API.

### **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFStatusFreeSummaryList
(FTFStatSummaryList *pStatSummaryList,
 FTFCA *pftfca);
```

### **OS/390 Prototype**

```
#include "ftfc.h"

FTFVOID FTFSFSL
(FTFStatSummaryList *pStatSummaryList,
 FTFCA *pftfca);
```



## Parameters

The following table lists and describes the FTFStatusFreeSummaryList API's parameters.

Parameter (Data Type)	Type	Description
pStatSummaryList (FTFStatSummaryList *) - input	Pointer to a Status Summary List data structure	Represents the data structure that contains the summary status information of the TDE components for all data-transfer requests that meet the filter criteria. It contains the most recent status updates for each TDE component in each matching data transfer.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information, including return codes and optional messages.

## Usage Notes

- Use a temporary pointer to pRow element in the pStatSummaryList data structure while processing the status information returned by the FTFStatusGetSummary API.
- The FTFStatusFreeSummaryList API call fails if you increment the pRow data element before you call the API.

## Related Data Structures

- FTFCA (page 215)
- FTFStatSummary (page 282)
- FTFStatSummaryList (page 286)

## **FTFStatusGetDetailList**

### **Description**

This API retrieves all detailed summary updates from TDE components during a TDE data transfer. It accepts an FTFID as an argument and retrieves all of the detailed status information for the transaction that matches it. This API supplies the calling program with a complete view of the TDE data transfer.

### **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFStatusGetDetailList
(MQHCONN hQM,
FTFStatusGetDetailListInfo *info,
FTFStatDetailList *pStatDetailList,
FTFCA *pftfca);
```

### **OS/390 Prototype**

```
#include "ftfc.h"

FTFVOID FTFDL
(MQHCONN hQM,
FTFStatusGetDetailListInfo *info,
FTFStatDetailList *pStatDetailList,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFStatusGetDetailList API's parameters.

Parameter (Data Type)	Type	Description
HQM (MQHCONN) - input	Connection Handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
pStatDetailList (FTFStatDetailList *) - output	Pointer to a Status Detail List data structure	Represents the output data structure that contains all the detailed status information for the complete TDE transaction.
info (FTFStatusGetDetailListInfo*)	Pointer to TDE Status Get Detail List Information Data Structure	Represents an input area that determines the detail information being returned.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information. This includes the return codes and optional messages.

## Usage Notes

- If you use the FTFStatusGetDetailList API in combination with the associated FTFStatusFreeDetailList API, use a temporary pointer to element pRow of pStatDetailList while processing the status information returned by the FTFStatusGetDetailList API.
- The FTFStatusFreeDetailList API call fails if you increment the pRow data element before you call the API.

## Related Data Structures

- FTFCA (page 215)
- FTFStatDetail (page 271)
- FTFStatDetailList (page 281)
- FTFStatusGetDetailListInfo (page 290)

## **FTFStatusGetSummaryList**

Each TDE component sends status updates during the processing cycle of a TDE data transfer. The FTFStatusGetSummaryList API retrieves the latest summary updates from each of the components during a data transfer. It accepts a filter argument and retrieves updates for each transaction that matches the given filter. For example, you can use this API to retrieve the status summary for all transactions that occurred on a specified date within a specified date range. You can also use this API to retrieve the status summary information for all data transfers that have failed. FTFStatusGetSummaryList allows the calling program to track the status of current data transfers and the status of completed data transfers.

### **General Prototype**

```
#include "ftfc.h"

FTFVOID FTFStatusGetSummaryList
(MQHCONN hQM,
FTFStatusGetSummaryListInfo *info,
FTFStatusSummaryFilter *pSummaryFilter,
FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```

### **OS/390 Prototype**

```
#include "ftfc.h"

FTFVOID FTFSL
(MQHCONN hQM,
FTFStatusGetSummaryListInfo *info,
FTFStatusSummaryFilter *pSummaryFilter,
FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFStatusGetSummaryList API's parameters.

Parameter	Type	Description
hQM (MQHCONN) - input	Connection Handle	Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.
info (FTFStatusGetSummaryList Info*)	Pointer to TDE Status Get Summary List Information Data Structure	Represents an input area that determines the summary information being returned.
pSummaryFilter (FTFStatusSummaryFilter *) - input	Pointer to a Status Summary Filter data structure	Represents a properly formatted data structure used by the FTFStatusGetSummaryList API call. It contains the filter information that is used by the API to retrieve summary status information for specific TDE transactions.
pStatSummaryList (FTFStatSummaryList *) - output	Pointer to a Status Summary List data structure	Represents the output data structure that contains the summary status information of the TDE components of all transactions that meet the filter criteria. It contains the most recent status updates for each TDE component associated with the transactions.
pftfca (FTFCA *) - output	Pointer to TDE Reply Message Data Structure	Represents an output area that returns properly formatted API reply information, including return codes and optional messages.

## Usage Notes

- If you use the FTFStatusGetSummaryList API in combination with the associated FTFStatusFreeSummaryList API, use a temporary pointer to element pRow of pStatDetailList while processing the status information returned by the FTFStatusGetDetailList API.
- The FTFStatusFreeDetailList API call fails if you increment the pRow data element before you call the API.

## Related Data Structures

- [FTFCA](#) (page 215)
- [FTFStatSummaryList](#) (page 286)
- [FTFStatusGetSummaryListInfo](#) (page 292)
- [FTFStatusSummaryFilter](#) (page 294)

# FTFSubmitStatusMsg

## Description

This API allows you to submit a status message from a user exit module to the TDE status subsystem. The message is then processed in the same manner as TDE status messages are. It appears in status detail information retrieved by FTFStatusGetDetailList.

## General Prototype

```
#include "ftfc.h"

FTFVOID FTFSubmitStatusMsg
(FTFExitInfo *info,
 FTFCHAR *customComponent,
 FTFCHAR *customType,
 FTFCHAR *detailText,
 FTFCA *ftfca);
```

## OS/390 Prototype

```
#include "ftfc.h"

FTFVOID FTFSSM
(FTFExitInfo *pFTFExitInfo,
 FTFCHAR customComponent[],
 FTFCHAR customStatusType[],
 FTFCHAR errorText[],
 FTFCA *pftfca);
```

## Parameters

The following table lists and describes the FTFSubmitStatusMsg API's parameters.

Parameter	Type	Description
*pFTFExitInfo (FTFExitInfo) - input	Pointer to an <i>FTFExitInfo</i> data structure	Contains the address of the variable that contains the FTFExitInfo data structure. This data structure contains the information required for processing user exit modules.
customComponent (FTFCHAR) - input	Character array	Contains the user-specified custom component submitted to the status subsystem.
customStatusType (FTFCHAR) - input	Character array	Contains the custom status type being submitted to the status subsystem.
errorText (FTFCHAR) - input	Character array	Contains a textual error text submitted to the status subsystem.
pftfca (FTFCA *)	Pointer to TDE Reply Message Data Structure	Contains the address of the variable that returns the properly formatted API reply information, including return codes and optional messages.

## Related Data Structures

- FTFCA (page 215)
- FTFExitInfo (page 225)



## C Data Structures

This chapter lists the data structures that handle data input and output for the C APIs. It also lists component status messages generated as part of a data-transfer process.

You can find these data structures in the `ftfc.h` file, which resides in the Tivoli Data Exchange (TDE) root directory.

This chapter includes the following sections:

Section	Page
Component Status Messages	210
FTFAS400FileInfo	213
FTFCA	215
FTFCancelInfo	217
FTFExitAuthInfo	219
FTFExitFileInfo	221
FTFExitInfo	225
FTFExitJobInfo	229
FTFExitQMgrsInfo	232
FTFExitRequestInfo	234
FTFExitSourceFileInfo	236
FTFExitStatusOffload	239
FTFExitTargetFileInfo	241
FTFExitUserInfo	245
FTFIdentifiersInfo	247
FTFJobInfo	248
FTFPingInfo	251
FTFQMgrsInfo	254

<b>Section</b>	<b>Page</b>
FTFRequestMsgInfo	256
FTFShutdownInfo	259
FTFShutdownReply	261
FTFSourceFileInfo	263
FTFStagedListMsgInfo	266
FTFStagedTransactionMsgInfo	267
FTFStageMsgInfo	269
FTFStatDetail	271
FTFStatDetailList	281
FTFStatSummary	282
FTFStatSummaryList	286
FTFStatusDeleteInfo	288
FTFStatusGetDetailListInfo	290
FTFStatusGetSummaryListInfo	292
FTFStatusSummaryFilter	294
FTFSubscribeMsgInfo	298
FTFSubscriptionListMsgInfo	301
FTFSubscriptionMsgInfo	302
FTFTargetFileInfo	304
FTFUserInfo	308

## **Component Status Messages**

This section lists and describes the most common FTF component and status combinations used in a data-transfer request. It lists status messages generated by the FTF Manager, the FTF Sender, and the FTF Receiver.

## FTF Manager Status Messages

FTF Manager Status Message	Description
FTFSTAT_REQUEST_SUBMITTED	The FTF Manager has sent the data-transfer request to the FTF Sender for processing.
FTFSTAT_REQUEST_FLAGGED_FOR_EXPIRATION	The FTF Manager has marked the data-transfer request for expiration.
FTFSTAT_REQUEST_EXPIRED	The data-transfer request has expired.
FTFSTAT_REQUEST_FLAGGED_FOR_CANCEL	The FTF Manager has marked the data-transfer request for cancellation.
FTFSTAT_REQUEST_CANCELLED	The data-transfer request has been canceled.
FTFSTAT_REQUEST_FAILED	The data-transfer request has failed.
FTFSTAT_REQUEST_COMPLETE	The data-transfer request has completed successfully.

## FTF Sender Status Messages

FTF Sender Status Message	Description
FTFSTAT_REQUEST_RECEIVED	The FTF Sender has received the data-transfer request.
FTFSTAT_PROCESSING	The FTF Sender is processing the data-transfer request.
FTFSTAT_REQUEST_TRANSMITTED	The FTF Sender has transmitted the required messages to the FTF Receiver.
FTFSTAT_REQUEST_RECOVERING	FTF could not complete the data-transfer request. The FTF Sender is now recovering.
FTFSTAT_REQUEST_FAILED	The data-transfer request has failed.
FTFSTAT_FAILED	The FTF Sender could not process the data-transfer request.
FTFSTAT_REQUEST_COMPLETE	The FTF Sender has completed processing the data-transfer request.

**FTF Receiver Status Messages**

<b>FTF Receiver Status Message</b>	<b>Description</b>
FTFSTAT_REQUEST_RECEIVED	The FTF Receiver has received the request.
FTFSTAT_PROCESSING	The FTF Receiver is processing the messages transmitted by the FTF Sender.
FTFSTAT_REQUEST_RECOVERING	FTF could not complete the data-transfer request. The FTF Receiver is now recovering.
FTFSTAT_REQUEST_EXPIRED	The request has expired.
FTFSTAT_REQUEST_CANCELLED	The request has been cancelled.
FTFSTAT_FAILED	The FTF Receiver component could not process the request.
FTFSTAT_REQUEST_COMPLETE	The FTF Receiver has completed processing the request.

# FTFAS400FileInfo

## Description

The FTFAS400FileInfo data structure contains information about target files written to the AS/400 platform.

## Data Elements

The following table lists and describes the FTFAS400FileInfo data structure's elements.

Name (Data Type)	Description
ccsid (FTFCHAR[ ])	Fixed length array that contains the CCSID value used as an identifier for the data-transfer request. If no CCSID is specified, FTF uses the job's CCSID.
rdclen (FTFLONG)	Determines the logical record length for the AS/400 target file. <b>Valid values:</b> 1-32767
fileType (FTFFileTypeInfo)	Enumerated data type that determines the type of file being written to the AS/400 target. <b>Valid values:</b> FTF_BINARY, FTF_TEXT
fileTypeAS400 (FTFAS400FileType)	Enumerated data type that determines the AS/400 file type for the target file. <b>Valid values:</b> FTF_AS400_TYPE_SAVE, FTF_AS400_TYPE_PHYSICAL
isCreateLibrary (FTFBOOL)	Determines whether FTF should create the specified library if it does not already exist.
libAsp (FTFLONG)	Specifies the library auxiliary storage pool (ASP) for a library created as part of the data-transfer request.
fileAsp (FTFLONG)	Specifies the file ASP for a file created as part of the data-transfer request.
libText (FTFCHAR[ ])	Fixed length array that specifies the description for a library created as part of the data-transfer request.
fileText (FTFCHAR[ ])	Fixed length array that specifies the description for a file created as part of the data-transfer request.

## Related Functions

- [FTFCancel](#) (page 183)

## Related Data Structures

- [FTFRequestMsgInfo](#) (page 256)

## Data Structure

```
typedef struct _FTFAS400FileInfo {
    FTFCCHAR          ccsid[FTF_AS400_CCSDID_SIZE];
    FTFLONG           rcdlen;
    FTFFFileTypeInfo  fileType;
    FTFAS400FileType  fileTypeAS400;
    FTFBUILD          isCreateLibrary;
    FTFLONG           libAsp;
    FTFLONG           fileAsp;
    FTFCCHAR          libText[FTF_AS400_TEXT_SIZE];
    FTFCCHAR          fileText[FTF_AS400_TEXT_SIZE];
} FTFAS400FileInfo;
```

# FTFCA

## Description

The FTFCA data structure receives output from all user exit modules and most C API functions. It contains various return codes and optional error messages that describe the actions requested by the application.

## Data Elements

The following table lists and describes the FTFCA data structure's elements.

<b>Name (Data Type)</b>	<b>Description</b>
rc1 (FTFRC)	Contains the primary return code or FTFRC. Primary return codes are defined in the ftfc.h header file and represent an FTF internal code with an associated message.
rc2 (FTFRC)	Contains the secondary return code. This data element contains external return codes or reason codes. For example, if FTF encounters an MQSeries error, the primary return code is an FTF return code and the secondary return code is an MQSeries return code (found in the cmqc.h). However, if the error is an FTF internal error the secondary code (rc2) can be either 0 or an additional FTF return code.
FTFErrMsg (FTFCHAR[ ]) )	Fixed length array that contains message text and is optional in many cases.

## Related Functions

- [FTFCancel](#) (page 183)
- [FTFPing](#) (page 185)
- [FTFReq](#) (page 187)
- [FTFShutdown](#) (page 192)
- [FTFStage](#) (page 194)
- [FTFStatusDelete](#) (page 196)
- [FTFStatusFreeDetailList](#) (page 198)
- [FTFStatusFreeSummaryList](#) (page 200)
- [FTFStatusGetDetailList](#) (page 202)
- [FTFStatusGetSummaryList](#) (page 204)
- [FTFSubmitStatusMsg](#) (page 207)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFCA {  
    FTFRCL          rc1;  
    FTFRCL          rc2;  
    FTFCCHAR        FTFErrMsg[1024];  
} FTFCA;
```



# FTFCancelInfo

## Description

The FTFCancelInfo data structure contains input information required by the FTFCancel API.

## Data Elements

The following table lists and describes the FTFCancelInfo data structure's elements.

Name (Data Type)	Description
lqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the <i>FTFCancel</i> API has a connection. If it is not provided, the local queue manager takes the value of the default queue manager. <b>Note:</b> This parameter is optional on all platforms except OS/390.
oqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager used by the <i>FTFCancel</i> API. This value represents the queue manager for which the FTF Manager has serviced the initial request.
ftfid (FTFCHAR[ ])	Fixed length array that contains the unique FTFID that identifies the data-transfer request being canceled.
cFile (FTFCHAR[ ])	Fixed length array that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue.
timeout (FTFLONG)	Determines the number of seconds from execution until the cancel request times out.
cq (FTFCHAR[ ])	Fixed length array that contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.

## Related Functions

- FTFCancel (page 183)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFCancelInfo {  
    FTFCHAR          lqm[MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR          oqm[MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR          ftfid[FTF_ID_SIZE];  
    FTFCHAR          cFile[FTF_MAX_PATH + 1];  
    FTFLONG          timeout;  
    FTFCHAR          cq[MQ_Q_NAME_LENGTH + 1];  
} FTFCancelInfo;
```

# FTFExitAuthInfo

## Description

The FTFExitAuthInfo data structure is passed to the security authorization module. It contains the information required for authorization processing.

## Data Elements

The following table lists and describes the FTFExitAuthInfo data structure's elements.

Name (Data Type)	Description
pUserId (FTFCHAR *)	Pointer to a variable that contains the user name for the person who initiated the data-transfer request. This value is retrieved from the operating system, must be eight or fewer characters long, and can contain only letters and numbers.
pPassword (FTFCHAR *)	Included for possible future use. Leave this data element null.
pFileName (FTFCHAR *)	Pointer to a variable that contains the name of the file FTF will read or write.
pSystem (FTFCHAR *)	Pointer to a variable that contains the NT domain name on Win 32 machines. For other operating systems, this data element should be left null.
accessFlag (FTFLONG)	Set this data element to <i>FTF_READ</i> on the FTF Sender and <i>FTF_WRITE</i> on the FTF Receiver.

## Related Functions

The security exit authorization module “Security Authorization Exits” in the *Tivoli Data Exchange User's Guide*.

## Related Data Structures

None

## **Data Structure**

```
typedef struct _FTFExitAuthInfo {  
    FTFCHAR      *pUserId;  
    FTFCHAR      *pPassword;  
    FTFCHAR      *pFileName;  
    FTFCHAR      *pSystem;  
    FTFLONG      accessFlag;  
} FTFExitAuthInfo;
```

# FTFExitFileInfo

## Description

The FTFExitFileInfo data structure contains the file information required to create an OS/390 file. All of the characteristics of a mainframe file are described.

The following table lists and describes the FTFExitFileInfo data structure's elements.

Name (Data Type)	Description
pFileName (FTFCHAR*)	
isStaged (FTFBOOL)	Determines whether the file is to be placed in the staging queue. After the source file has been read, it can be archived in a staging queue until the file needs to be transmitted to each of its destinations. <b>Valid values:</b> 1 (place file in staging queue), 0 (do not place file in staging queue).
isStagePersistent (FTFBOOL)	Determines whether the data in the stage area is persistent or nonpersistent. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the staged data is not present, you must reaccess the source before you can send it again. <b>Valid values:</b> 1 (make staged data persistent), 0 (do not make staged data persistent).
isDataPersistent (FTFBOOL)	Determines whether the data can be recovered after a system restart. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the data is not present, you must reaccess the source before you can send it again. This is a performance option which defaults to non-persistent. <b>Valid values:</b> 1 (transmission is persistent), 0 (transmission not persistent).
isCompressed (FTFBOOL)	Determines whether the file being sent is compressed using the FTF internal compression algorithm. <b>Valid values:</b> 1 (compression is used) 0 (compression is not used)
fileType (FTFFileTypeInfo)	Should reflect the format of the data that is being transmitted. If you select a binary format, no conversion or formatting takes place. If you select a text format, data-type translation occurs as is necessary for the target platform. <b>Valid values:</b> FTF_BINARY (binary), FTF_TEXT (text).

## C Data Structures

### FTFExitFileInfo

Name (Data Type)	Description
fileMode (FTFFileModeInfo)	Determines how the data is processed at the destination. <b>Valid values:</b> <ul style="list-style-type: none"><li>• FTF_CREATE_FILE (create a new file),</li><li>• FTF_APPEND_FILE (append to an existing file),</li><li>• FTF_NOREPLACE_FILE (do not replace an existing file).</li></ul>
fileOrg (FTFFileOrgInfo)	Determines the target file's OS/390 file organization. <b>Valid values:</b> PS (Physical Sequential), PDS (Partitioned Data Set) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
cDirectoryBlocks (FTFLONG)	Determines the number of directory blocks allocated for the target file on OS/390.
recordFormat (FTFRecordFormatInfo)	Determines the target file's OS/390 record format. <b>Valid values:</b> F (fixed), V (variable), FB (fixed block), and VB (variable block) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
lrecl (FTFLONG)	Determines the target file's OS/390 logical record length. <b>Valid values:</b> 1 - 32760. <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
blockSize (FTFLONG)	Determines the target file's OS/390 block size. Specifying a block size of 0 enables the system to choose the optimum block size for the data set during allocation. If the record format is fixed block (FB), the block size must be a multiple of the logical record length value (the lrecl data element). When the record format is variable length (VB), the blksize value must be at least four bytes greater than the lrecl value. <b>Valid values:</b> 0 - 32760
unitName (FTFCHAR[ ])	Fixed length array that determines the target file's OS/390 unit name.
volser (FTFCHAR[ ])	Fixed length array that determines the target file's volume serial number.
allocationUnit (FTFAllocUnitInfo)	Determines the target file's OS/390 allocation. <b>Valid values:</b> CYL (cylinder), BLK (block), and TRK (track)
primaryAllocSize (FTFLONG)	Determines the target file's OS/390 primary allocation unit size.
secondaryAllocSize (FTFLONG)	Determines the target file's OS/390 secondary allocation unit size.

<b>Name (Data Type)</b>	<b>Description</b>
textWrapRecord (FTFTextFileWrapInfo)	Indicates how records are processed when they are longer than the specified target-file record length. <b>Valid values:</b> <ul style="list-style-type: none"><li>• FTFREQ_WRAP (wraps records)</li><li>• FTFREQ_FAIL (causes the data transfer to fail)</li><li>• FTFREQ_TRUNC (truncates records at the specified record length)</li></ul>
createDirectory (FTFBOOL)	Determines whether the specified target directory is created when it does not already exist. <b>Valid values:</b> 1 (create directory), 0 (do not create a directory).

## Related Functions

None

## Related Data Structures

- FTFExitAuthInfo (page 219)
- FTFExitJobInfo (page 229)
- FTFExitQMgrsInfo (page 232)
- FTFExitRequestInfo (page 234)
- FTFExitSourceFileInfo (page 236)
- FTFExitTargetFileInfo (page 241)
- FTFExitUserInfo (page 245)

**Data Structure**

```
typedef struct _FTFEXITFILEINFO {
    FTFCHAR          *pFileName;
    FTFBOOL           isStaged;
    FTFBOOL           isStagePersistent;
    FTFBOOL           isDataPersistent;
    FTFBOOL           isCompressed;
    FTFFileInfo       fileType;
    FTFFileInfo       fileMode;
    FTFFileInfo       fileOrg;
    FTFULONG          cDirectoryBlocks;
    FTFRecordFormatInfo recordFormat;
    FTFULONG          lrecl;
    FTFULONG          blockSize;
    FTFCHAR           unitName
                      [FTFREQ_UNITNAME_SIZE];
    FTFCHAR           volser   [FTFREQ_VOLSER_SIZE];
    FTFAllocUnitInfo  allocationUnit;
    FTFULONG          primaryAllocSize;
    FTFULONG          secondaryAllocSize;
    FTFTextFileWrapInfo textWrapRecord;
    FTFBOOL           createDirectory;
} FTFEXITFILEINFO;
```



# FTFExitInfo

## Description

The FTFExitInfo data structure contains the exit information required to use the pre- and post- processing exits supplied with FTF. This data structure is passed to the exit being called for the user-written program to reference. Codes returned in the FTFRC data element can be analyzed for possible error conditions.

---

### Notes:

- For more information about custom connector development, contact CommerceQuest ([www.commercequest.com](http://www.commercequest.com)).
  - If you are running connectors on a Solaris 2.5.1 operating system, you must install Solaris Patch 103627.
- 

## Data Elements

The following table lists and describes the FTFExitInfo data structure's elements.

Name (Data Type)	Description
ftfid (FTFCHAR[ ])	Fixed length array that contains an FTF identifier used to identify the current data-transfer request. The FTFReq API generates this unique identifier.
exitNumber (FTFLONG)	Contains the number of the exit to be called. The following exits are available:
	3. Manager pre-process exit User-written exit for pre-processing requirements at the FTF Manager. This exit is invoked before the data-transfer request is sent to the FTF Sender for processing.

Name (Data Type)	Description
	<p>4. Manager post-process exit  User-written exit for pre-processing requirements at the FTF Manager. This exit is invoked after the data-transfer request is complete. It is useful for processing acknowledgments to user programs. Each exit receives the current FTFRC as input. Therefore, the FTF Manager post-process exit can be used to audit file transfer for all FTF requests that it processes in order to detect failure or success.</p>
	<p>5. Sender pre-process exit  User-written exit for pre-processing requirements at the FTF Sender. This exit is invoked before the FTF Sender reads the source file.</p>
	<p>6. Sender post-process exit  User-written exit for post-processing requirements at the FTF Sender. This exit is invoked after the FTF Sender deposits the outbound messages on the transmission queues.</p>
	<p>7. Receiver pre-process exit  User-written exit for pre-processing requirements at the FTF Receiver. This exit is invoked before the FTF Receiver processes inbound messages and writes them to the target file.</p>
	<p>8. Receiver post-process exit  User-written exit for post-processing requirements at the FTF Receiver. This exit is invoked by the FTF Receiver after the target file is written. Because the FTF receiver performs the last step in the data-transfer request just before the FTF Manager logs the final status, this exit is useful for auditing failed or successful data transfer requests.</p>
	<p>9. Sender connector. For information about custom connector development, contact CommerceQuest (<a href="http://www.Commercequest.com">www.Commercequest.com</a>).</p>
	<p>10. Receiver connector. For information about custom connector development, contact CommerceQuest (<a href="http://www.commercequest.com">www.commercequest.com</a>).</p>
pDllName (FTFCHAR*)	<p>Pointer to a variable that contains the name of the run-time module to be called for the specified exit. This data element is case sensitive and must be in the run-time path of the environment calling the exit (unless the DLL name contains the fully qualified path of the run-time module).</p>
pEntryPoint (FTFCHAR*)	<p>Pointer to a variable that contains the entry point within the DLL that contains the exit module being executed. Typically, multiple exits can use the same run-time module but have different entry points.</p>

<b>Name (Data Type)</b>	<b>Description</b>
pUserId (FTFCHAR*)	Pointer to a variable that contains a user ID. FTF performs no action based on the value in this data element. It is passed to the called exit and can be used for any purpose by the exit module.
pPassword (FTFCHAR*)	Pointer to a variable that contains a password. FTF performs no action based on the value of this field. It is passed to the exit module and can be used for any application-specific purpose.
cUserData (FTFLONG)	Contains the number of bytes within the <i>userData</i> data element.
pUserData (FTFCHAR *)	Pointer to a variable that contains a user data section that is passed to the exit. This data can be in any format and any length required by the exit module.
rc (FTFRC)	Contains a return code that is populated as input to the exit for the post exits. This return code can be used to determine if the data-transfer request is successful or has failed.
rc2 (FTFRC)	Contains a secondary return code that is populated as input to the exit for the post exits. This return code can be used to determine if the data-transfer request is successful or has failed.
pInternal (FTFVOID*)	
Ids (FTFIdentifiersInfo)	Contains the three user-defined fields which are associated with the data-transfer request.

## Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit
- Sender pre-process exit
- Sender post-process exit
- Receiver pre-process exit
- Receiver post-process exit
- Security exit authorization module

## Related Data Structures

- FTFExitAuthInfo (page 219)
- FTFExitJobInfo (page 229)

- FTFExitQMgrsInfo (page 232)
- FTFExitRequestInfo (page 234)
- FTFExitSourceFileInfo (page 236)
- FTFExitTargetFileInfo (page 241)
- FTFExitUserInfo (page 245)
- FTFIdentifiersInfo (page 247)

## Data Structure

```
typedef struct _FTFExitInfo {
    FTFCHAR          ftfid[FTF_MAX_FTFID_SIZE];
    FTFLONG          exitNumber;
    FTFCHAR          *pDllName;
    FTFCHAR          *pEntryPoint;
    FTFCHAR          *pUserId;
    FTFCHAR          *pPassword;
    FTFLONG          cUserData;
    FTFCHAR          *pUserData;
    FTFRC            rc;
    FTFRC            rc2;
    FTFVOID          *pInternal
    FTFIdentifiersInfo Ids;
} FTFExitInfo;
```

# FTFExitJobInfo

## Description

The FTFExitJobInfo data structure passes information about the current data-transfer to an invoked exit module.

## Data Elements

The following table lists and describes the FTFExitJobInfo data structure's elements.

Name (Data Type)	Description
poolName (FTFCHAR[ ])	Fixed length array that represents the target pool name for the data associated with the data-transfer request. For more information about using pools, see the <i>Tivoli Data Exchange User's Guide</i> , "Using Pools."
priority (FTFLONG)	Represents the priority of the current data-transfer request. <b>Valid values:</b> 1 (highest) - 5 (lowest)
expirationDateTime (FTFLONG)	Represents the number of seconds between the current time and the time that the data-transfer request expires.
isTrusted (FTFBOOL)	Represents the rules governing the recovery protection. By default, all data-transfer requests are protected under syncpoint control. If the trusted option is enabled (meaning the transfer is trusted without any additional FTF protection), FTF does not perform syncpoint control. In the event of a failure, the data-transfer request is terminated with a failing notification. <b>Valid values:</b> 1 (trusted), 0 (not trusted). <b>Note:</b> If you specify that the data-transfer is trusted and it fails, no attempts are made to recover. All aspects of the failing data-transfer request are purged from the system and the appropriate notifications are posted.
reserved (FTFBOOL)	For internal use only. Do not populate this data element.
isStageOnly (FTFBOOL)	Determinates whether the source information is to be sent only to a staging queue and not to another destination. <b>Valid values:</b> 1 (send only to a staging queue), any other value (do not send staging queue)

Name (Data Type)	Description
isStagedftfid (FTFBOOL)	This data element determines whether the <i>pSource</i> data element in the <i>FTFExitRequestInfo</i> data structure is an FTFID. If this value is set to True, the source file with the matching FTFID is sent in the file transfer. <b>Valid values:</b> 1 (True, use the FTFID), any other value (False, do not use the FTFID)
isStagedFile (FTFBOOL)	This data element determines whether the <i>pSource</i> data element in the <i>FTFExitRequestInfo</i> data structure is a filename. If this value is set to True, the source file with the matching filename is sent in the data-transfer request. <b>Valid values:</b> 1 (True, use the filename), any other value (False, do not use the filename)
cancelMode (FTFCancelMode)	Determines how pre-emptive cancellation should be handled. <b>Valid values:</b> <ul style="list-style-type: none"> <li>FTF_PREEMPTCANCEL_NEUTRAL (uses the cancellation value set in the FTF configuration file),</li> <li>FTF_PREEMPTCANCEL_ON (allows the file transfer to be preemptively canceled),</li> <li>FTF_PREEMPTCANCEL_OFF (does not allow preemptive cancellation)</li> </ul>
delsrc (FTFBOOL)	Indicates whether the source data is to be deleted after being transferred.
immed (FTFBOOL)	Indicates that the data transfer transaction is an immediate transfer.
Ids (FTFIdentifiersInto)	Contains the three user-defined fields which are associated with the data-transfer request.

## Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit

## Related Data Structures

- FTFExitRequestInfo (page 234)
- FTFFIdentifiersInfo (page 247)
- FTFJobInfo (page 248)

## Data Structure

```
typedef struct _FTFExitJobInfo {
    FTFCHAR          poolName [MQ_Q_NAME_LENGTH + 1];
    FTFLONG          priority;
    FTFLONG          expirationDateTime;
    FTFBOOL          isTrusted;
    FTFBOOL          reserved;
    FTFBOOL          isStageOnly;
    FTFBOOL          isStagedftfid;
    FTFBOOL          isStagedFile;
    FTFCancelMode    cancelMode;
    FTFBOOL          delsrc;
    FTFBOOL          immed;
    FTFFIdentifiersInfo Ids;
} FTFExitJobInfo;
```

## FTFExitQMgrsInfo

### Description

The FTFExitQMgrsInfo data structure passes information about the current data-transfer request's queue managers. This information is passed to the exit module being invoked as part of the data-transfer request.

### Data Elements

The following table lists and describes the FTFExitQMgrsInfo data structure's elements.

Name (Data Type)	Description
local (FTFCHAR[ ])	Fixed length array that contains the name of the local queue manager to which the interface connects to submit the data-transfer request to the FTF subsystem. If not specified, the requesting interface connects to the default queue manager. <b>Note:</b> This element is required on OS/390. <b>Valid values:</b> lqm name (OS/390 systems), null string (non-OS/390 systems).
originating (FTFCHAR[ ])	Fixed length array that contains the originating queue manager on which the FTF data-transfer request originates. If omitted, the FTF subsystem uses the lqm value for the oqm.
source (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager from where the source file is sent. The FTF Sender must be running on this node.
target (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager where the destination file is received. The FTF Receiver must be running on this node.

### Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit



## Related Data Structures

- FTFExitRequestInfo (page 234)
- FTFQMgrsInfo (page 254)

## Data Structure

The FTFExitQMgrsInfo data structure is derived from the FTFQMgrsInfo data structure. This section lists both structures.

### FTFExitQMgrsInfo Data Structure

```
typedef FTFQMgrsInfo FTFExitQMgrsInfo;
```

### FTFQMgrsInfo Data Structure

```
typedef struct _FTFQMgrsInfo {  
    FTFCHAR local           [MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR originating    [MQ_Q_MGR_NAME_LENGTH +  
                             MQ_Q_MGR_NAME_LENGTH + 2];  
    FTFCHAR source         [MQ_Q_MGR_NAME_LENGTH +  
                             MQ_Q_MGR_NAME_LENGTH + 2];  
    FTFCHAR target         [MQ_Q_MGR_NAME_LENGTH +  
                             MQ_Q_MGR_NAME_LENGTH + 2];  
} FTFQMgrsInfo;
```

## FTFExitRequestInfo

### Description

The FTFExitRequestInfo data structure passes information about the data-transfer request to the exit module.

### Data Elements

The following table lists and describes the FTFExitRequestInfo data structure's elements.

Name (Data Type)	Description
pQMgrs (FTFExitQMgrsInfo*)	Pointer to a variable that contains queue manager information that resides in the FTFExitQMgrsInfo data structure.
pSource (FTFExitSourceFileInfo*)	Pointer to a variable that contains source file information that resides in the FTFExitSourceFileInfo data structure.
pTarget (FTFExitTargetFileInfo*)	Pointer to a variable that contains target file information that resides in the FTFExitTargetFileInfo data structure.
pJob (FTFExitJobInfo*)	Pointer to a variable that contains information about the data-transfer request within which the exit is being run. This information resides in the FTFExitJobInfo data structure.
pUser (FTFExitUserInfo*)	Pointer to a variable that contains information about the data-transfer group name and label. This information resides in the FTFExitUserInfo data structure.

### Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit

## Related Data Structures

- FTFExitAuthInfo (page 219)
- FTFExitInfo (page 225)
- FTFExitJobInfo (page 229)
- FTFExitQMgrsInfo (page 232)
- FTFExitSourceFileInfo (page 236)
- FTFExitTargetFileInfo (page 241)
- FTFExitUserInfo (page 245)
- FTFRequestMsgInfo (page 256)

## Data Structure

```
typedef struct _FTFExitRequestInfo {
    FTFExitQMgrsInfo      *pQMgrs;
    FTFExitSourceFileInfo *pSource;
    FTFExitTargetFileInfo *pTarget;
    FTFExitJobInfo        *pJob;
    FTFExitUserInfo       *pUser;
} FTFExitRequestInfo;
```

# FTFExitSourceFileInfo

## Description

The FTFExitSourceFileInfo data structure passes information about the current data-transfer request’s source file to the exit module.

## Data Elements

The following table lists and describes the FTFExitSourceFileInfo data structure’s elements.

Name (Data Type)	Description
pFileName (FTFCHAR*)	Pointer to a variable that contains the source file’s name and fully qualified path. This path must be syntactically correct for the source platform that is accessing the file.
isStaged (FTFBOOL)	Determines whether the file is to be placed in the staging queue. After the source file has been read, it can be archived in a staging queue until the file needs to be transmitted to each of its destinations. <b>Valid values:</b> 1 (place file in staging queue), 0 (do not place file in staging queue).
isStagePersistent (FTFBOOL)	Determines whether the data in the stage area is persistent or nonpersistent. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the staged data is not present, you must reaccess the source before you can send it again. <b>Valid values:</b> 1 (make staged data persistent), 0 (do not make staged data persistent).
isDataPersistent (FTFBOOL)	Determines whether the data can be recovered after a system restart. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the data is not present, you must reaccess the source before you can send it again. This is a performance option which defaults to non-persistent. <b>Valid values:</b> 1 (transmission is persistent), 0 (transmission not persistent).
isCompressed (FTFBOOL)	Determines whether the file being sent is compressed using the FTF internal compression algorithm. <b>Valid values:</b> 1 (compression is used) 0 (compression is not used)

<b>Name (Data Type)</b>	<b>Description</b>
fileType (FTFFileTypeInfo)	Should reflect the format of the data that is being transmitted. If you select a binary format, no conversion or formatting takes place. If you select a text format, data-type translation occurs as is necessary for the target platform. <b>Valid values:</b> FTF_BINARY (binary), FTF_TEXT (text).
bufNo (FTFLONG)	Determines the number of buffers used during a data-transfer request on the OS/390 platform.

## Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit
- Sender pre-process exit
- Sender post-process exit

## Related Data Structures

- FTFExitAuthInfo (page 219)
- FTFExitInfo (page 225)
- FTFExitJobInfo (page 229)
- FTFExitQMgrsInfo (page 232)
- FTFExitSourceFileInfo (page 236)
- FTFExitTargetFileInfo (page 241)
- FTFExitUserInfo (page 245)
- FTFExitSourceFileInfo (page 236)

**Data Structure**

```
typedef struct _FTFExitSourceFileInfo {  
    FTFCHAR          *pFileName;  
    FTFBOOL           isStaged;  
    FTFBOOL           isStagePersistent;  
    FTFBOOL           isDataPersistent;  
    FTFBOOL           isCompressed;  
    FTFFileTypeInfo   fileType;  
    FTFLONG           bufNo;  
} FTFExitSourceFileInfo;
```

# FTFExitStatusOffload

## Description

The FTFExitStatusOffload data structure pulls status messages, formats them into XML data streams, and passes them to a user-defined exit.

## Data Elements

The following table lists and describes the FTFExitStatusOffload data structure's elements.

<b>Name (Data Type)</b>	<b>Description</b>
FTF_STATD_ACTION	Action to take when the exit is called (initiate, submit, end).
pszDBName (FTFCHAR)	Name of the data source.
pszDBUserID (FTFCHAR)	Database user ID.
pszDBPassword (FTFCHAR)	Database password.
StatMsgBuff (FTFCHAR)	Pointer to an array of XML strings.
pszUserData (FTFCHAR)	Pointer to user exit information.
lBatchSize (FTFLONG)	Number of records in this batch.
cControlMsg (FTFLONG)	Count of the number of control messages (always at the top of the buffer).
cDetailMsg (FTFLONG)	Count of the number of detail messages.
pLogFile	Pointer to the FTFSTATD log file to log any SQL error text
rc (FTFLONG)	Contains the primary return code value.
rc2 (FTFLONG)	Contains the secondary (SQL) return code value.

## Related Functions

None

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFExitStatusOffload {  
    FTF_STATD_ACTION  Action;  
    FTFCHAR            *pszDBName;  
    FTFCHAR            pszDBUserID;  
    FTFCHAR            pszDBPassword;  
    FTFCHAR            **StatMsgBuf;  
    FTFCHAR            *pszUserData;  
    FTFLONG            lBatchSize;  
    FTFLONG            cControlMsg;  
    FTFLONG            cDetailMsg;  
    FILE               *pLogFile;  
    FTFLONG            rc;  
    FTFLONG            rc2;  
} FTFExitStatusOffload;
```



# FTFExitTargetFileInfo

## Description

The FTFExitTargetFileInfo data structure passes information about the current transfer's target file to the exit module.

## Data Elements

The following table lists and describes the FTFExitTargetFileInfo data structure's elements.

Name (Data Type)	Description
pFileName (FTFCHAR*)	Pointer to a variable that contains the target file's fully qualified path and name. This path must be syntactically correct for the target platform that will be depositing the file.
isCompressed (FTFBOOL)	Reflects whether inbound data is compressed or uncompressed. <b>Valid values:</b> 1 (compressed), 0 (uncompressed).
fileType (FTFFileTypeInfo)	Reflects the format of the data that is being transmitted. If you select a binary format, no conversion or formatting takes place. If you select a text format, data-type translation occurs as is necessary for the particular code page of the target platform. <b>Valid values:</b> FTF_BINARY (binary), FTF_TEXT (text).
fileMode (FTFFileModeInfo)	Determines how the data is processed at the destination. <b>Valid values:</b> <ul style="list-style-type: none"> <li>FTF_CREATE_FILE (create a new file),</li> <li>FTF_APPEND_FILE (append to an existing file),</li> <li>FTF_NOREPLACE_FILE (do not replace an existing file).</li> </ul>
fileOrg (FTFFileOrgInfo)	Determines the target file's OS/390 file organization. <b>Valid values:</b> PS (Physical Sequential), PDS (Partitioned Data Set) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
cDirectoryBlocks (FTFLONG)	Determines the number of directory blocks allocated for the target file on OS/390.
recordFormat (FTFRecordFormatInfo)	Determines the target file's OS/390 record format. <b>Valid values:</b> F (fixed), V (variable), FB (fixed block), and VB (variable block) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .

## C Data Structures

### FTFExitTargetFileInfo

Name (Data Type)	Description
lrecl (FTFLONG)	Determines the target file's OS/390 logical record length. <b>Valid values:</b> 1 - 32760. <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
blockSize (FTFLONG)	Determines the target file's OS/390 block size. Specifying a block size of 0 enables the system to choose the optimum block size for the data set during allocation. If the record format is fixed block (FB), the block size must be a multiple of the logical record length value (the lrecl data element). When the record format is variable length (VB), the blksize value must be at least four bytes greater than the lrecl value. <b>Valid values:</b> 0 - 32760
unitName (FTFCHAR[ ]) )	Fixed length array that determines the target file's OS/390 unit name.
volser (FTFCHAR[ ]) )	Fixed length array that determines the target file's volume serial number.
allocationUnit (FTFAllocUnitInfo)	Determines the target file's OS/390 allocation. <b>Valid values:</b> CYL (cylinder), BLK (block), and TRK (track)
primaryAllocSize (FTFLONG)	Determines the target file's OS/390 primary allocation unit size.
secondaryAllocSize (FTFLONG)	Determines the target file's OS/390 secondary allocation unit size.
textWrapRecord (FTFTextFileWrapInfo)	Indicates how records are processed when they are longer than the specified target-file record length. <b>Valid values:</b> <ul style="list-style-type: none"><li>• FTFREQ_WRAP (wraps records)</li><li>• FTFREQ_FAIL (causes the data transfer to fail)</li><li>• FTFREQ_TRUNC (truncates records at the specified record length)</li></ul>
createDirectory (FTFBOOL)	Determines whether the specified target directory is created when it does not already exist. <b>Valid values:</b> 1 (create directory), 0 (do not create a directory).

Name (Data Type)	Description
isDataPersistent (FTFBOOL)	Determines whether the data is persistent or nonpersistent. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the data is not persistent, you must reaccess it before you can send it again. <b>Valid values:</b> 1 (transmission is persistent), 0 (transmission not persistent).
bufNo (FTFLONG)	Determines the number of buffers used during a data-transfer request on the OS/390 platform.

## OS/390 File Allocation Items

The items listed in the FTFExitTargetFileInfo data structure represent the rules governing how files are dynamically file allocated on OS/390. These items are optional. Entries in the FTF configuration file fill in the missing entries. If items are specified in the FTF configuration file at the source FTF Sender's node or the target FTF Receiver's node, they are resolved at those specific points respectively. The items provided through the API take precedence. Next, items are resolved at the FTF Sender. Any outstanding items are attempted for resolution at the FTF Receiver.

## Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit
- Receiver pre-process exit
- Receiver post-process exit

## Related Data Structures

- FTFExitAuthInfo (page 219)
- FTFExitInfo (page 225)
- FTFExitJobInfo (page 229)
- FTFExitQMgrsInfo (page 232)
- FTFExitSourceFileInfo (page 236)
- FTFExitUserInfo (page 245)
- FTFTargetFileInfo (page 304)

## Data Structure

```
typedef struct _FTFExitTargetFileInfo {
    FTFCHAR          *pFileName;
    FTFBOOL          isCompressed;
    FTFFFileTypeInfo  fileType;
    FTFFFileModeInfo  fileMode;
    FTFFFileOrgInfo   fileOrg;
    FTFULONG          cDirectoryBlocks;
    FTFRecordFormatInfo recordFormat;
    FTFULONG          lrecl;
    FTFULONG          blockSize;
    FTFCHAR          unitName[FTFREQ_UNITNAME_SIZE];
    FTFCHAR          volser[FTFREQ_VOLSER_SIZE];
    FTFFAllocUnitInfo allocationUnit;
    FTFULONG          primaryAllocSize;
    FTFULONG          secondaryAllocSize;
    FTFTextFileWrapInfo textWrapRecord;
    FTFBOOL          createDirectory;
    FTFBOOL          isDataPersistent;
    FTFULONG          bufNo;
} FTFExitTargetFileInfo;
```

# FTFExitUserInfo

## Description

The FTFExitUserInfo data structure passes information about the current transfer's group and label to the exit module.

## Data Elements

The following table lists and describes the FTFExitUserInfo data structure's elements.

Name (Data Type)	Description
groupName (FTFCHAR[ ])	Fixed length array that determines the name of the group to which this data-transfer request belongs.
label (FTFCHAR[ ])	Fixed length array that contains a user-specified label associated with the data being transmitted. The value of the label has no bearing on the internal processing of an data-transfer request The label value is used only for output and query specifications.

## Related Functions

The entry point function for the following exits, as described in the *Tivoli Data Exchange User's Guide*:

- Manager pre-process exit
- Manager post-process exit

## Related Data Structures

- FTFExitRequestInfo (page 234)
- FTFUserInfo (page 308)

**Data Structure**

```
typedef struct _FTFExitUserInfo {  
    FTFCHAR      groupName [FTFREQ_GROUPNAME_SIZE];  
    FTFCHAR      label     [FTFREQ_LABEL_SIZE];  
} FTFExitUserInfo;
```

# FTFIdentifiersInfo

## Description

The FTFIdentifiersInfo data structure contains the data placed in the three data identifier fields associated with a data transfer. This data is defined by the user.

## Data Elements

The following table lists and describes the FTFIdentifiersInfo data structure's elements.

Name (Data Type)	Description
Id1 (FTFCHAR[ ])	Fixed length array that contains the text data that was entered into the data identifier 1 field.
Id2 (FTFCHAR[ ])	Fixed length array that contains the text data that was entered into the data identifier 2 field.
Id3 (FTFCHAR[ ])	Fixed length array that contains the text data that was entered into the data identifier 3 field.

## Related Functions

None.

## Related Data Structures

None.

## Data Structure

```
typedef struct _FTFIdentifiersInfo {
    FTFCHAR Id1[FTF_IDENTIFIER_SIZE];
    FTFCHAR Id2[FTF_IDENTIFIER_SIZE];
    FTFCHAR Id3[FTF_IDENTIFIER_SIZE];
} FTFIdentifiersInfo;
```

## FTFJobInfo

### Description

The FTFJobInfo data structure contains the rules governing the associated file transfer transaction. It contains information about the transfer's pool, priority, recovery capabilities, message size, and staging.

### Data Elements

The FTFJobInfo data structure contains the following data elements.

Name (Data Type)	Description
poolName (FTFCHAR[ ])	Fixed-length array that represents the target pool name for the data associated with the data-transfer request. For more information about creating and using pools, see “Using Pools” in the <i>Tivoli Data Exchange User's Guide</i> .
transferPriority (FTFLONG)	The priority of the current data-transfer request. <b>Valid values:</b> 1 (highest) - 5 (lowest)
numberOfRetries (FTFLONG)	Number of retry attempts for the transfer request.
expirationDateTime (FTFLONG)	Number of seconds between the current time and the time that the data-transfer request expires.
MQMsgSize (FTFLONG)	The size (in bytes) of each individual MQSeries message created and used during the data-transfer request's execution.
isTrusted (FTFBOOL)	The rules governing recovery protection. By default, all data-transfer requests are protected under syncpoint control. If the trusted option is enabled (meaning the transfer is trusted without any additional FTF protection), FTF does not perform syncpoint control. If a failure occurs, the transfer request ends with a notification of failure. <b>Valid values:</b> 1 (trusted), 0 (not trusted) <b>Note:</b> If you specify that the data transfer is trusted and it fails, no attempts are made to recover. All aspects of the failing data-transfer request are purged from the system and the appropriate notifications are posted.
reserved (FTFBOOL)	Reserved. Do not use this data element.



<b>Name (Data Type)</b>	<b>Description</b>
isStageOnly (FTFBOOL)	Determines that the current transaction is staged only. No destination is specified.
isStagedftfid (FTFBOOL)	This data element determines whether the <i>pSource</i> data element in the FTFRequestInfo data structure is an FTFID. If this value is set to True, the source file with the matching FTFID is sent in the file transfer. <b>Valid values:</b> 1 (True, use the FTFID), any other value (False, do not use the FTFID)
isStagedFile (FTFBOOL)	This data element determines whether the <i>pSource</i> data element in the FTFExitRequestInfo data structure is a filename. If this value is set to True, the source file with the matching filename is sent in the data-transfer request. <b>Valid values:</b> 1 (True, use the filename), any other value (False, do not use the filename)
cancelMode (FTFCancelMode)	Records how a preemptive cancellation should be handled. <b>Valid values:</b> <ul style="list-style-type: none"> <li>FTF_PREEMPTCANCEL_NEUTRAL (uses the cancellation value set in the FTF configuration file),</li> <li>FTF_PREEMPTCANCEL_ON (allows the file transfer to be preemptively canceled),</li> <li>FTF_PREEMPTCANCEL_OFF (does not allow preemptive cancellation)</li> </ul> <b>Default value:</b> FTF_PREEMPTCANCEL_NEUTRAL
delsrc (FTFBOOL)	Indicates that the source data is to be deleted after transfer.
Ids (FTFIdentifiersInfo)	Contains the three user-defined fields which are associated with the data-transfer request.
immed (FTFBOOL)	Indicates that the data transfer transaction is an immediate transfer.
isFormatXML (FTFBOOL)	Indicates that the request is in XML format.
retryCount (FTFLONG)	
isPublish (FTFBOOL)	Indicates that the transaction is to be published via MQSeries to subscribers.
*pStream (FTFCHAR)	Specifies the broker stream that is subscribed to for a designated topic.

## Related Functions

- [FTFReq](#) (page 187)

## Related Data Structures

- [FTFExitJobInfo](#) (page 229)
- [FTFRequestMsgInfo](#) (page 256)

## Data Structure

```
typedef struct _FTFJobInfo {
    FTFCCHAR          poolName [MQ_Q_NAME_LENGTH+1];
    FTFLONG            transferPriority;
    FTFLONG            numberOfRetries;
    FTFLONG            expirationDateTime;
    FTFLONG            MQMsgSize;
    FTTFBOOL           isTrusted;
    FTTFBOOL           reserved;
    FTTFBOOL           isStageOnly;
    FTTFBOOL           isStagedftfid;
    FTTFBOOL           isStagedFile;
    FTFCancelMode      cancelMode;
    FTTFBOOL           delsrc;
    FTFIdentifiersInfo Ids;
    FTTFBOOL           immed;
    FTTFBOOL           isFormatXML;
    FTFLONG            retryCount;
    FTFLONG            isPublish;
    FTFCCHAR           *pStream;
} FTFJobInfo;
```

# FTFPingInfo

## Description

The FTFPingInfo data structure contains input information required by the FTFPing API. Its data elements describe all of the components included in the ping operation.

## Data Elements

The following table lists and describes the FTFPingInfo data structure's elements.

Name (Data Type)	Description
lqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the FTFPing API has a connection handle. This parameter is mandatory on the OS/390 platform. On other platforms, if it is not provided, the lqm takes the value of the MQSeries default queue manager.
oqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the FTF Manager is connected. The ping message is forwarded from the lqm to this queue manager. This data element is mandatory on the OS/390 platform. On other operating systems, if it is not provided, the oqm takes the value of the MQSeries default queue manager.
sqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the FTF Manager at the oqm forwards the ping message. This parameter is optional. If it is not provided, FTF uses the lqm value. In other words, the FTF Sender is on the lqm.
dqm (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the FTF Sender at the sqm forwards the ping message. This parameter is optional. If it is not provided, FTF uses the lqm value. In other words, the FTF Receiver is on the lqm.
cFile (FTFCHAR[ ])	Fixed length array that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue.

## C Data Structures

### *FTFPingInfo*

Name (Data Type)	Description
timeout (FTFLONG)	Determines how long the FTFPing API should wait for the final response from the FTF Manager. This value is specified in seconds. <b>Valid values:</b> A value between 0 and ULONGMAX.
msgSize (FTFLONG)	Determines the ping message size. This value is specified in kilobytes (KB). You can use this value to test various message size values until you determine the optimal message size setting. <b>Valid values:</b> A value between 1 and the maximum message size of MQSeries in KB (this value is platform specific). MQSeries V5 maximum message size is 100 megabytes (MB). Other MQSeries platforms have a maximum message size of 4 MB.
priority (FTFLONG)	Determines the ping message's priority. <b>Valid values:</b> 1 (highest) - 5 (lowest).
kBytesWritten (FTFLONG)	Contains the actual number of bytes written to MQSeries for the ping message. This value is represented in kilobytes. <b>Valid values:</b> A value between 1 and the maximum message size of MQSeries in kilobytes.
cq (FTFCHAR[ ])	Fixed length array that contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.

## Related Functions

- FTFPing (page 185)

## Related Data Structures

- None

## Data Structure

```
typedef struct _FTFPingInfo
{
    FTFCHAR          lqm[MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR          oqm[MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR          sqm[MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR          dqm[MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR          cFile[FTF_MAX_PATH + 1];
    FTFLONG          timeout;
    FTFLONG          msgSize;
    FTFLONG          priority;
    FTFLONG          kBytesWritten;
    FTFCHAR          cq[MQ_Q_NAME_LENGTH + 1];
} FTFPingInfo;
```

# FTFQMgrsInfo

## Description

The FTFQMgrsInfo data structure provides details about the path that a data-transfer request will take during processing. It identifies each of the queue managers used by FTF components during the transfer request.

Users can specify the client name (node) and queue manager name on the command line, in order to take advantage of the support for large client configurations. See chapter X, “Using the Default Queue Naming Convention,” for more information.

## Data Elements

The following table lists and describes the FTFQMgrsInfo data structure’s elements.

Name (Data Type)	Description
local (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager to which the interface connects to submit the data-transfer request to the FTF subsystem. If this element is omitted, the requesting interface connects to the MQSeries default queue manager. <b>Note:</b> This element is required on OS/390.
originating (FTFCHAR[ ])	Fixed length array that contains the name of the originating queue manager on which the FTF data-transfer request originates. The FTF Manager must be running on this node. If this element is omitted, the FTF subsystem uses the lqm value for the oqm.
source (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager from where the source file is sent. The FTF Sender must be running on this node.
target (FTFCHAR[ ])	Fixed length array that contains the name of the queue manager where the destination file is received. The FTF Receiver must be running on this node.

## Related Functions

- FTFReq (page 187)

## Related Data Structures

- FTFExitQMgrsInfo (page 232)
- FTFRequestMsgInfo (page 256)
- FTFStageMsgInfo (page 269)

## Data Structure

```
typedef struct _FTFQMgrsInfo {
    FTFCHAR    local[MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR    originating[MQ_Q_MGR_NAME_LENGTH +
                          MQ_Q_MGR_NAME_LENGTH + 2];
    FTFCHAR    source[MQ_Q_MGR_NAME_LENGTH +
                      MQ_Q_MGR_NAME_LENGTH + 2];
    FTFCHAR    target[MQ_Q_MGR_NAME_LENGTH +
                      MQ_Q_MGR_NAME_LENGTH + 2];
} FTFQMgrsInfo;
```

## FTFRequestMsgInfo

### Description

The FTFRequestMsgInfo data structure contains input information required by the FTFReq API. It is also passed to the FTF Manager, FTF Sender, and FTF Receiver exits.

A user-written application that makes use of the FTFRequest API is required to format the data correctly for input to the API. A user-written exit may be required to review various data elements where necessary.

### Data Elements

The following table lists and describes the FTFRequestMsgInfo data structure's elements.

Name (Data Type)	Description
qmgrs (FTFQMgrsInfo)	Contains the data structure that contains the local, originating, source, and destination queue managers used in the data-transfer request.
sourceFile (FTFSourceFileInfo)	Contains the data structure that contains information about the data-transfer request's source file.
targetFile (FTFTargetFileInfo)	Contains the data structure that contains information about the data-transfer request's target file, as well as any information required for allocating disk space for the target file.
job (FTFJobInfo)	Contains the data structure that contains the rules governing the data-transfer request.
userInfo (FTFUserInfo)	Contains the data structure that contains label and group information for the current data-transfer request.
isReply (FTFBOOL)	Determines whether FTF should wait for a reply before returning control to the calling application. <b>Valid Values:</b> True (wait for reply), another value (do not wait for reply).
timeout (FTFULONG)	Determines the length of time (in seconds) that the data-transfer request stays active before it completes. It represents the length of time that will be calculated within FTF and MQSeries for the expiration of this request within the system.



<b>Name (Data Type)</b>	<b>Description</b>
replyWaitTime (FTFULONG)	Determines the length of time (in seconds) to wait for a reply before control is returned to the calling application. If a response is received before the timeout is exhausted, the appropriate reply and control will be returned to the calling interface.
configFile (FTFCHAR *)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue. Otherwise, this argument is required on the OS/390 platform and optional on other platforms if the FTF_CONFIG_FILE environment variable is set.
cExitInfo (FTFLONG)	Determines the number of exits that to be invoked during the data-transfer request.
exitInfo (FTFEXITINFO *)	Pointer to a variable that contains a data structure that governs how user exits are invoked and executed.
as400 (FTFAS400FileInfo)	Contains a data structure that describes information about the target files to be written to the AS/400 platform.
ftfid (FTFCHAR[ ])	Fixed length array that contains an FTF identifier used to identify the current data-transfer request. The FTFReq API generates this unique identifier.
cq (FTFCHAR *)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.
isDBCS (FTFBOOL)	Indicates whether DBCS processing is required.

## Related Functions

- FTFReq (page 187)

## Related Data Structures

- FTFAS400FileInfo (page 213)
- FTFExitInfo (page 225)
- FTFJobInfo (page 248)
- FTFQMgrsInfo (page 254)
- FTFSourceFileInfo (page 263)
- FTFTargetFileInfo (page 304)
- FTFUserInfo (page 308)

**Data Structure**

```
typedef struct _FTFRequestMsgInfo {
    FTFQMgrsInfo      qmgrs;
    FTFSourceFileInfo sourceFile;
    FTFTargetFileInfo targetFile;
    FTFJobInfo        job;
    FTFUserInfo       userInfo;
    FTFBOOL           isReply;
    FTFULONG          timeout;
    FTFULONG          replyWaitTime;
    FTFCHAR           *configFile;
    FTFULONG          cExitInfo;
    FTFEXITINFO       *exitInfo;
    FTFAS400FileInfo  as400;
    FTFCHAR           ftfid[FTF_ID_SIZE];
    FTFCHAR           *cq;
    FTFBOOL           isDBCS;
} FTFRequestMsgInfo;
```

# FTFShutdownInfo

## Description

The FTFShutdownInfo data structure contains the input information required for the FTFShutdown API. The FTFShutdown API will shut down specified FTF components and tracks which components have actually shut down.

## Data Elements

The following table lists and describes the FTFShutdownInfo data structure's elements.

Name (Data Type)	Description
lqm (FTFCHAR )	Pointer to a variable that contains the name of the queue manager to which the FTFShutdown API has a connection handle.
node (FTFCHAR )	Determines the node to which the FTFShutdown API directs its request.
component (FTFCOMPONENT)	Determines the component or components to which the shutdown API directs the shutdown request. This data element can specify a single component or each of the components at a specified node.
immediate (FTFBOOL)	Determines whether FTF performs a quiesce or immediate shutdown. A quiesce shutdown (the default) waits for all work to finish before the shutdown request is processed. An immediate shutdown preempts any work that is pending and immediately shuts down the specified component(s). <b>Valid values:</b> 1 (immediate), 0 (quiesce).
timeout (FTFULONG)	Determines the length of time, in seconds, to wait for a response from each specified component and its instances. If all responses are returned before the timeout, the FTF shutdown API returns the status. Otherwise, a timeout condition occurs. <b>Valid values:</b> 0 to ULONGMAX.

Name (Data Type)	Description
configFile (FTFCHAR )	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue. Otherwise, this argument is required on the OS/390 platform and optional on other platforms if the FTF_CONFIG_FILE environment variable is set.
cq (FTFCHAR)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.

## Related Functions

- FTFShutdown (page 192)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFShutdownInfo {
    FTFCHAR          lqm [MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCHAR          node [MQ_Q_MGR_NAME_LENGTH + 1];
    FTFCOMPONENT     component;
    FTFBOOL          immediate;
    FTFULONG         timeout;
    FTFCHAR          configFile[FTF_MAX_PATH + 1];
    FTFCHAR          cq [MQ_Q_NAME_LENGTH +1];
} FTFShutdownInfo;
```

# FTFShutdownReply

## Description

The FTFShutdownReply data structure contains the output information generated by the FTFShutdown API. This output information tracks the number of components that replied to the shutdown request and the number of components that should have replied.

## Data Elements

The following table lists and describes the FTFShutdownReply data structure's elements.

Name (Data Type)	Description
manager.replied (FTFBOOL)	Indicates whether the FTF Manager replied to the shutdown request. This data element is set to True if the FTF Manager has replied and False if it has not. <b>Valid values:</b> 1 (FTF Manager has replied), 0 (FTF Manager has not replied)
sender.cSender (FTFLONG)	Indicates the number of FTF Senders that should respond to the shutdown request. This value is determined by the number of instances specified in the FTF configuration file. <b>Valid values:</b> 0 to ULONGMAX.
sender.cReply (FTFLONG)	Indicates the number of FTF Senders that have replied to the shutdown request.
receiver.cReceiver (FTFLONG)	Indicates the number of FTF Receivers that should respond to the shutdown request. This value is determined by the number of instances specified in the FTF configuration file. <b>Valid values:</b> 0 to ULONGMAX.
receiver.cReply (FTFLONG)	Indicates the number of FTF Receivers that have replied to the shutdown request.
statd.cStatd (FTFLONG)	Indicates the number of status offload daemons that should respond to the shutdown request. This value is determined by the number of instances specified in the FTF configuration file. <b>Valid values:</b> 0 to ULONGMAX.
statd.cReply (FTFLONG)	Indicates the number of status offload daemons that have replied to the shutdown request.

## C Data Structures

### *FTFShutdownReply*

Name (Data Type)	Description
logger.cLogger (FTFLONG)	Indicated the number of FTFLOG processes that should respond to the shutdown request. <b>Valid values:</b> 0 to ULONGMAX
logger.cReply (FTFLONG)	Indicates the number of FTFLOG processes that have replied to the shutdown request.

## Related Functions

- FTFShutdown (page 192)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFShutdownReply {
    struct {
        FTFBUILD      replied;
    } manager;
    struct {
        FTFLONG      cSender;
        FTFLONG      cReply;
    } sender;
    struct {
        FTFLONG      cReceiver;
        FTFLONG      cReply;
    } receiver;
    struct {
        FTFLONG      cStatd;
        FTFLONG      cReply;
    } statd;
    struct {
        FTFLONG      cLogger;
        FTFLONG      cReply;
    } logger;
} FTFShutdownReply;
```

# FTFSourceFileInfo

## Description

The FTFSourceFileInfo data structure contains the information required to create and allocate space for the source file.

## Data Elements

The following table lists and describes the FTFSourceFileInfo data structure's elements.

Name (Data Type)	Description
pFilename (FTFCHAR*)	Pointer to the variable that contains the source file's name and fully qualified path. This path must be syntactically correct for the source platform that is accessing the file.
isStaged (FTFBOOL)	Determines whether the file is to be placed in the staging queue. After the source file has been read, it can be archived in a staging queue until the file needs to be transmitted to each of its destinations. <b>Valid values:</b> 1 (staging required), 0 (staging not required).
isStagePersistent (FTFBOOL)	Determines whether the data in the stage area is persistent or nonpersistent. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the staged data is not present, you must reaccess it before you can send it again. <b>Valid values:</b> 1 (staging queue data is persistent), 0 (staging queue data is not persistent).
isDataPersistent (FTFBOOL)	Determines whether the data can be recovered after a system restart. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the data is not present, you must reaccess the source before you can send it again. <b>Valid values:</b> 1 (transmission is persistent), 0 (transmission not persistent).
isCompressed (FTFBOOL)	Determines whether the file being sent is compressed using the FTF internal compression algorithm. <b>Valid values:</b> 1 (compression is used) 0 (compression is not used)
isDelete (FTFBOOL)	Used internally by FTF.

## C Data Structures

### *FTFSourceFileInfo*

Name (Data Type)	Description
fileType (FTFFileTypeInfo)	Should reflect the format of the data that is being transmitted. If you select a binary format, no conversion or formatting takes place. If you select a text format, data-type translation occurs as is necessary for the target platform. <b>Valid values:</b> FTF_BINARY (binary), FTF_TEXT (text).
filePad (FTFRecordPadInfo)	Enumerated data type that determines whether padding takes place. <b>Valid values:</b> FTFREQ_PAD, FTFREQ_NOPAD
padChar (FTFBYTE)	Determines the character to be used for record padding if record padding is enabled. If you specify a single character for this value, you must enclose it in single quotes. <b>Valid values:</b> A-Z, a-z, 0-9, space, 0x00-0xFF (where the last two positions are a hexadecimal value mapped to a character). <b>Default value:</b> space
sType (FTFCHAR)	Determines the sender file type for a transfer in which data not stored in files. This value is required if you want to send data that is not stored in a file. The valid values for this data element are based on the values specified in the FTF configuration file.
cstypeData (FTFLONG)	Determines the length in bytes of the *pstypeData data element.
pstypeData (FTFCHAR*)	Contains the source data that is not contained in a file.
bufNo (FTFLONG)	Determines the number of buffers used during a data-transfer request on the OS/390 platform.

## Related Functions

None

## Related Data Structures

- FTFExitSourceFileInfo (page 236)
- FTFRequestMsgInfo (page 256)



## Data Structure

```
typedef struct _FTFSourceFileInfo {
    FTFCHAR      *pFileName;
    FTFBOOL      isStaged;
    FTFBOOL      isStagePersistent;
    FTFBOOL      isDataPersistent;
    FTFBOOL      isCompressed;
    FTFBOOL      isDelete;
    FTFFileTypeInfo fileType;
    FTFRecordPadInfo filePad;
    FTFBYTE      padChar;
    FTFCHAR      stype [FTF_FILETYPE_SIZE];
    FTFULONG     cstypeData;
    FTFCHAR      *pstypeData;
    FTFULONG     bufNo;
} FTFSourceFileInfo;
```

# FTFStagedListMsgInfo

## Description

The FTFStagedListMsgInfo data structure contains the number of staging records returned from the FTFStage API on a query. It also contains another data structure that contains the staging records.

## Data Elements

The following table lists and describes the FTFStagedListMsgInfo data structure’s elements.

Name (Data Type)	Description
cRows (FTFLONG)	Contains the number of staged items in the query return list.
pRows (*FTFStagedTransaction MsgInfo*)	Contains the address of the variable that contains the first record returned from the query request.

## Related Functions

- FTFStage (page 194)

## Related Data Structures

- FTFStagedTransactionMsgInfo (page 267)

## Data Structure

```
typedef struct _FTFStagedListMsgInfo
{
    FTFLONG                cRows;
    FTFStagedTransactionMsgInfo *pRows;
} FTFStagedListMsgInfo;
```

# FTFStagedTransactionMsgInfo

## Description

The FTFStagedTransactionMsgInfo data structure contains the detail information output from an FTFStage query. Its data elements relate to the characteristics of the data-transfer request that is being processed and file that is being sent.

## Data Elements

The following table lists and describes the FTFStagedTransactionMsgInfo data structure's elements.

Name (Data Type)	Description
ftfid[FTF_ID_SIZE] (FTFCHAR)	Contains the staged transaction's FTFID.
sqm[FTF_NAME_SIZE] (FTFCHAR)	Contains the name of the staged transaction's source queue manager.
pSourceNameFile (FTFCHAR *)	Contains the name of the staged file.
stagedTimeStamp (FTFSTS)	Contains the time and date the file was staged.
MQMsgSize (FTFLONG)	Contains the size of the data messages that compose the staged file.
cDataBlocks (FTFLONG)	Contains the number of data messages that compose the staged file.
isCompressed (FTFBOOL)	Determines whether the staged messages are compressed. <b>Valid values:</b> True (compressed), False (not compressed)

## Related Functions

- FTFStage (page 194)

## Related Data Structures

- FTFStagedListMsgInfo (page 266)

## Data Structure

```
typedef struct _FTFStagedTransactionMsgInfo {
    FTFCHAR          ftfid[FTF_ID_SIZE];
    FTFCHAR          sqm[FTF_NAME_SIZE];
    FTFCHAR          *pSourceFileName;
    FTFTS            stagedTimeStamp;
    FTFLONG          MQMsgSize;
    FTFLONG          cDataBlocks;
    FTFBOOL          isCompressed;
} FTFStagedTransactionMsgInfo;
```

# FTFStageMsgInfo

## Description

The FTFStageMsgInfo data structure contains information required as input by the FTFStage API. It contains the following data:

- A list of affected queue managers
- The FTFID of any staged items that are to be purged
- A determination of whether a reply is required
- Time to wait for the reply
- An action code that determines the action taken by the FTFStage function.

You must populate this data structure before you call the FTFStage function.

## Data Elements

The following table lists and describes the FTFStageMsgInfo data structure's elements.

Name (Data Type)	Description
qmgrs (FTFQMgrsInfo)	Contains the data structure that contains the local, originating, source, and destination queue managers used by the FTFStage API.
pConfigFile (FTFCHAR*)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue. Otherwise, this argument is required on the OS/390 platform and optional on other platforms if the FTF_CONFIG_FILE environment variable is set.
pFile (FTFCHAR*)	Pointer to the variable that contains the FTFID or the filename of the item to be purged from the staging area.
isReply (FTFCHAR)	Determines whether a reply will be generated after the purge or query operation takes place.
replyWaitTime (FTFLONG)	Determines the time to wait for a reply from a query request. This value does not apply to purge requests.

Name (Data Type)	Description
action (FTFStageReq)	<p>Determines the type of action to be carried out.</p> <p><b>Valid values:</b></p> <ul style="list-style-type: none"> <li>FTFSTAGE_QUERY – If value is 1, the action is to query the staging queue.</li> <li>FTFSTAGE_PURGE_FILE – If the value is 2, the action is to purge a specific file from the staging queue.</li> <li>FTFSTAGE_PURGE_FTFID – If the value is 3, the action is to purge entries for a specific FTFID.</li> <li>FTFSTAGE_PURGE_ALL – If the value is 4, the action is to purge all files from the staging queues.</li> </ul>
cq (FTFCHAR *)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.

## Related Functions

- FTFStage (page 194)

## Related Data Structures

- FTFQMgrsInfo (page 254)

## Data Structure

```
typedef struct _FTFStageMsgInfo {
    FTFQMgrsInfo      qmgrs;
    FTFCHAR            *pConfigFile;
    FTFCHAR            *pFile;
    FTFCHAR            isReply;
    FTFULONG           replyWaitTime;
    FTFStageReq        action;
    FTFCHAR*           cq;
} FTFStageMsgInfo;
```

# FTFStatDetail

## Description

The FTFStatDetail data structure contains the detailed status records for a data-transfer request. It lists the component that generated the status message, the status, date, and time of the status record, and any error codes and text associated with the status.

## Data Elements

The following table lists and describes the FTFStatDetail data structure's elements.

Name (Data Type)	Description
ftfId (FTFCHAR)	Pointer to a variable that contains the unique FTFID that matches the data-transfer request for which the detailed status was retrieved.
timestamp (FTFSTS)	Contains the date and time at which the data-transfer request was initiated. The value returned is of type time_t returned from the time() function.
localQmgr (FTFCHAR)	Pointer to a variable that contains the name of the queue manager to which the reporting component has a connection handle.
component (FTFComponent)	Contains the type of the FTF component that issued the detailed status message.
status (FTFStatus)	Contains the status associated with the FTFcomponent reporting the current detailed message. FTFStatus is an enumerated data type that is defined in the ftfc.h header file and contains all possible statuses of the FTF subsystem.
ftfcode (FTFCODE)	Contains the primary processing or return code. The last three digits correspond to the FTF return codes outlined in this guide and defined in the ftfc.h header file.
src (FTFCODE)	Contains the secondary processing or return code. The last four digits are an external code. The code may be an MQSeries return code, but if an internal FTF error occurs it may be an FTF return code.
detailText (FTFCHAR)	Pointer to the variable that contains message text associated with an error.

Name (Data Type)	Description
curMsgNumber (FTFLONG)	Contains the message number of the messages created during the data-transfer request. This field is populated for messages with a status of PROCESSING that originated from the FTF Sender and FTF Receiver.
totalMsgCount (FTFLONG)	Contains the number of messages created during the data-transfer request. This field is populated for messages with a status of PROCESSING that originated from the FTF Sender and FTF Receiver.
CustomComponent (FTFCHAR)	Can contain user-specified custom component information submitted to the FTF status subsystem.
CustomStatusType (FTFCHAR)	Can contain user-specified custom status information submitted to the FTF status subsystem.

## Related Functions

- [FTFStatusFreeDetailList](#) (page 198)
- [FTFStatusGetDetailList](#) (page 202)

## Related Data Structures

- [FTFStatDetailList](#) (page 281)

## Data Structure

```
typedef struct _FTFStatDetail {
    FTFCHAR    ftfId[FTF_ID_SIZE];
    FTFTS      timestamp;
    FTFCHAR    localQmgr[FTF_NAME_SIZE];
    FTFComponent component;
    FTFStatus   status;
    FTFCODE    ftfcode;
    FTFCODE    src;
    FTFCHAR    detailText[FTF_MAX_ERR_TXT_SIZE];
    FTFLONG    curMsgNumber;
    FTFLONG    totalMsgCount;
    FTFCHAR    CustomComponent
               [FTF_MAX_CUSTOM_CPT_TXT_SIZE];
}
```



```
FTFCHAR CustomStatusType
[FTF_MAX_CUSTOM_ST_TYPE_TXT_SIZE];
} FTFStatDetail;
```

## **FTFSTAT Usage Notes**

The FTFStat API functions allow you to perform the following tasks:

- Retrieve real time status updates on data-transfer requests
- Run reports associated with data-transfer requests
- Purge all status messages associated with a selected data-transfer request.

Using the FTFStat API, you can generate reports for the following:

- All data-transfer requests, whether successful or failed
- All data-transfer requests where the source file resides on a specific queue or the destination file resides on a specific queue
- All data-transfer requests within a date range

You can choose to generate a report using all options or only one option. For example, you can run a date-specific report for all failed data-transfer requests.

The FTFStat APIs are most useful when more they are used in combination with each other. For example, you could use the following sequence to purge certain status information:

- Use the FTFStatusGetSummary API to retrieve all data-transfer requests that are more a week old and return the FTFID for each.
- Pass the FTFID values to the FTFStatusDelete API to remove all the status messages associated with the data-transfer requests. By calling the FTFStatusDelete function with each FTFID, you purge all status messages that are more a week old from the system.

You can also use the FTFStatusGetSummary API to retrieve all data-transfer requests that failed during the previous seven days. This API can return the FTFID for each of these data-transfer requests. You can then FTFID values to the FTFStatusGetDetail API to obtain all the status messages for the data-transfer request. By calling the FTFStatusGetDetail API with each FTFID, you can determine which components are failing the data-transfer requests and the most common reason for the failure.

## Examples of FTFStat API Implementation

The following example shows how to use the FTFStatusGetSummaryList API to retrieve the most recent status update associated with a data-transfer request and how to use the FTFStatusFreeSummaryList data structure. Because this example uses the current date as the filter, only data-transfer requests for the current date are returned in the FTFStatSummaryList structure.

---

### Note:

FTFStatus and FTFCComponent are enumerated data types and defined in the header file (ftfc.h). These data types are returned in the FTFStatSummary and FTFStatDetail data structures (also defined in the header file). User-written functions can be used to print out text representations of the FTFStatus and FTFCComponent.

---

```

/*****
/* Define variables and temp ptr.      */
/*****/
FTFStatusSummaryFilter  summaryFilter;
FTFStatSummaryList      summaryList;
FTFStatSummary          *pTempRow;
FTFCA                   ftfCa;
time_t                  tTime;

/*****/
/* Must initialize filter to NULLS      */
/*****/
memset(&summaryFilter, '\0', sizeof(summaryFilter));

/*****/
/* Connect to the local queue manager */
/*****/
MQCONN((char *)argv[1], &hQM, &compCode, &reason);

/*****/
/* originatingStartTimeStamp is mandatory */
/* populating with current date.          */
/* Format: YYYYMMDDhhmmss, partial dates  */
/* are accepted.                          */
/*****/
tTime = time(0);
strftime(summaryFilter.originatingStartTimestamp,
          sizeof(summaryFilter.originatingStartTimestamp),
          "%Y%m%d", localtime(&tTime));

```

```

/*****
/* Call the FTFStatusGetSummaryList API */
*****/
FTFStatusGetSummaryList(hQM, &configFile, &LocalQM, &summaryFilter,
    &summaryList, &ftfCa);

/*****
/* Check the return code from the API */
*****/
if (ftfCa.rc1 != 0) {
    printf("FTF Summary List Request failed (%d,%d)", ftfCa.rc1, ftfCa.rc2);
    if (ftfCa.FTFErrMsg[0]) {
        printf(". Reason: %s", ftfCa.FTFErrMsg);
    }
    FTFStatusFreeSummaryList(&summaryList, &ftfCa);
    exit(EXIT_FAILURE);
}

/*****
/* assign temp pointer. This is done */
/* so the original pointer can be used */
/* in the FTFStatusFreeSummaryList API. */
*****/
pTempRow = summaryList.pRow;

/*****
/* Check to see if any rows (FTF/MQ Transactions) were */
/* returned. */
*****/
if (summaryList.cRows) {

    /*****
    /* Loop through each row printing out the latest */
    /* status information for the current transaction */
    /* we are processing. */
    *****/
    for (currentRow = 0; currentRow < summaryList.cRows; currentRow++) {

        /*****
        /* print id, nodes, and files for the FTF/MQ Transaction */
        *****/
        printf("FTFID: %.*s\n", (int)sizeof(pTempRow->ftfId), pTempRow->ftfId);
        printf("Source Queue Manager: %.24s\n", pTempRow->sourceQmgr);
        printf("Source File Name: %.50s\n", pTempRow->sourceFilename);
        printf("Target Queue Manager: %.24s\n", pTempRow->targetQmgr);
        printf("Target File Name: %.50s\n", pTempRow->targetFilename);
    }
}

```

```

/*****
/* Printing the most recent status update for the FTF/MQ */
/* transaction. The order of status replies from the FTF */
/* components are Requestor -> Manager -> Sender -> Receiver */
/* -> Manager where the Manager's last updates are COMPLETE or */
/* FAILED. In order to display the most recent update we */
/* check the components in the reverse order. If there is */
/* a Manger status of COMPLETE or FAILED this will be the */
/* final update. If this status doesn't exist for the Manger */
/* we will print the Receiver's status if one exists if not we */
/* will check the Sender, then the Manger and finally the */
/* Requestor status. */
/*
/* Note: The Statuses being printed are numbers that correlate */
/* to a status that is defined in the supplied ftfc.h file. */
*****/

if ((pTempRow->originatingQmgrStatus == FTFSTAT_REQUEST_COMPLETE) ||
    (pTempRow->originatingQmgrStatus == FTFSTAT_REQUEST_FAILED)){
    printf("FTF MANAGER: ");
***    printf("STATUS: %d\n", pTempRow->originatingQmgrStatus);
    printf("%5s - ", PrintTimeStamp(pTempRow->timestamp));
    printf("%5s\n", PrintTimeStamp(pTempRow->originatingQmgrTimestamp));
}
else if (pTempRow->targetQmgrStatus != 0) {
    printf("FTF RECEIVER: ");
***    printf("STATUS %d\n", pTempRow->targetQmgrStatus);
    printf("%5s - ", PrintTimeStamp(pTempRow->timestamp));
    printf("%s\n", PrintTimeStamp(pTempRow->targetQmgrTimestamp));
}
else if (pTempRow->sourceQmgrStatus != 0) {
    printf("FTF SENDER: ");
***    printf("STATUS %d\n", pTempRow->sourceQmgrStatus);
    printf("%5s - ", PrintTimeStamp(pTempRow->timestamp));
    printf("%s\n", PrintTimeStamp(pTempRow->sourceQmgrTimestamp));
}

else if (pTempRow->originatingQmgrStatus != 0) {
    printf("FTF MANAGER: ");
***    printf("%STATUS: d\n", pTempRow->originatingQmgrStatus);
    printf("%5s - ", PrintTimeStamp(pTempRow->timestamp));
    printf("%5s\n", PrintTimeStamp(pTempRow->originatingQmgrTimestamp));
}
else {
    printf("FTF REQUEST: ");
***    printf("STATUS %d\n", pTempRow->requestingQmgrStatus);
    printf("%5s - ", PrintTimeStamp(pTempRow->timestamp));
    printf("%5s\n", PrintTimeStamp(pTempRow->requestingQmgrTimestamp));
}

```

```

/*****
/* Check for error information from that status messages */
*****/
if (pTempRow->ftfcode) {
    printf("Error: %ld ", pTempRow->ftfcode);
    printf("Error: %ld ", pTempRow->ftfcode);
    if (pTempRow->src)
        printf("(%ld) ", pTempRow->src);
}

/*****
/* increment Temp pointer */
*****/
pTempRow++;
}
else {
    printf("No transfer events found.\n");
}

/*****
/* Free the data allocated in the */
/* previous API call. */
*****/
FTFStatusFreeSummaryList(&summaryList, &ftfCa);

```

The following example shows how you can use the FTFStatusGetDetailList API to retrieve all the status updates associated with a supplied data-transfer request and how to use the FTFStatusFreeDetailList data structure. This data structure prints each status message associated with the data-transfer request.

```

/*****
/* Define variables and temp ptr.      */
/*****/
FTFStatDetailList    detailList;
FTFCA                ftfCa;
FTFStatDetail        *pRow;

/*****/
/* Connect to the local queue manager */
/*****/
MQCONN((char *)argv[1], &hQM, &compCode, &reason);

/*****/
/* set detailList buffer to NULLs      */
/*****/
memset(&detailList, '\0', sizeof(FTFStatDetailList));

/*****/
/* Call the FTFStatusGetDetailList API */
/*****/
FTFStatusGetDetailList(hQM, &configFile, &localQM, &ftfid, &detailList, &ftfCa);

/*****/
/* Check the return code from the API   */
/*****/
if (ftfCa.rc1 != 0) {
    printf("FTF Detail List Request failed (%d,%d)", ftfCa.rc1, ftfCa.rc2);
    if (ftfCa.FTFErrMsg[0]) {
        printf(". Reason: %s", ftfCa.FTFErrMsg);
    }
    FTFStatusFreeDetailList(&detailList, &ftfCa);
}

/*****/
/* Check to see if any rows (FTF/MQ Transactions) were */
/* returned.                                           */
/*****/
if (detailList.cRows) {

    /*****/
    /* assign temp pointer.      */
    /*****/
    pRow = detailList.pRow;

    /*****/
    /* Loop through each row printing out each status */
    /* record for the FTFID we are processing.        */
    /*****/

```

```

for (currentRow = 0; pRow && currentRow < detailList.cRows; currentRow++)
{
    printf("\n");
    printf("FTFID                : %.37s\n", pRow->ftfId);
    printf("QMgr                : %.*s\n", (int)sizeof(pRow->localQmgr),
                                                pRow->localQmgr);
    printf("Component            : %d\n", pRow->component);
    printf("Status                : %d\n", pRow->status);

    /******
    /* Check for errors reported in */
    /* the status messages.          */
    /******
    if (pRow->ftfcode || pRow->src || pRow->errorText[0])
    {
        printf("Error                : %ld ", pRow->ftfcode);
        if (pRow->src)
            printf("(%ld) ", pRow->src);

        printf("%.*s", (int)sizeof(pRow->errorText), pRow->errorText);
        printf("\n");
    }
    pRow++;
} else {
    printf("No transfer events found.\n");
}

/******
/* Free the data allocated in the */
/* previous API call.              */
/******
FTFStatusFreeDetailList(&detailList, &ftfCa);

```

The following sample shows how to use the FTFStatusDelete API to purge all status messages associated with an FTFID. This API removes old status messages from the Tivoli Data Exchange subsystem.

```

/*****/
/* Define variables. */
/*****/
FTFCA          ftfCa;

/*****/
/* Connect to the local queue manager */
/*****/
MQCONN((char *)argv[1], &hQM, &compCode, &reason);

/*****/
/* Call the FTFStatusDelete API */
/*****/
FTFStatusDelete(hQM, &configFile, &localQM, &ftfid, &ftfCa);

/*****/
/* Check the return code from the API */
/*****/
if (ftfCa.rc1 != 0) {
    printf("FTF Status Delete Request failed (%d,%d)", ftfCa.rc1, ftfCa.rc2);
    if (ftfCa.FTFErrMsg[0]) {
        printf(". Reason: %s", ftfCa.FTFErrMsg);
    }
}
```



# FTFStatDetailList

## Description

The FTFStatDetailList data structure contains the detail status information that describes the complete status history for the specified FTFID. It contains all status messages associated with the data-transfer request.

## Data Elements

The following table lists and describes the FTFStatDetailList data structure's elements.

Name	Description
cRows (FTFLONG)	Contains the number of detailed records that have been returned.
pRows (FTFStatDetail*)	Pointer a variable that contains to the FTFStatDetail data structure which contains detail status records operated during a data-transfer request.

## Related Functions

- FTFStatusFreeDetailList (page 198)
- FTFStatusGetDetailList (page 202)

## Related Data Structures

- FTFStatDetail (page 271)

## Data Structure

```
typedef struct _FTFStatDetailList
{
    FTFLONG          cRows;
    FTFStatDetail    *pRows;
} FTFStatDetailList;
```

## FTFStatSummary

### Description

The FTFStatSummary data structure contains the summary status record for a specific data-transfer request. It contains the most recent status updates for each FTF component used during the associated data-transfer request. It contains the unique FTFID and a section for each components: the FTF Manager, the FTF Sender, and the FTF Receiver.

### Data Elements

The following table lists and describes the FTFStatusSummary data structure's elements.

Name	Description
ftfId (FTFCHAR)	Pointer to a variable that contains the unique FTFID for the data-transfer request for which summary status information was retrieved.
timestamp (FTFSTS)	Contains the date and time at which the data-transfer request was initiated. The value returned is of type <code>time_t</code> returned from the <code>time()</code> function.
ftfcode (FTFCODE)	Contains the primary processing or return code. The last three digits of this code correspond to the FTF return codes outlined in the <i>Tivoli Data Exchange Messages and Codes Guide</i> .
src (FTFCODE)	Contains the secondary processing or return code. The last four digits of this code represent an external code. The code may be an MQSeries return code, but if an internal FTF error occurs it may be an FTF return code.
localQmgr (FTFCHAR)	Pointer to a variable that contains the name of the queue manager to which the reporting component has a connection handle.
originatingQmgr (FTFCHAR)	Pointer to a variable that contains the name of the queue manager to which the data-transfer request's FTF Manager is connected.
originatingQmgrStatus (FTFStatus)	Contains the most recent status update from the FTF Manager for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the FTF subsystem. It is an enumerated data type that is defined in the <code>ftfc.h</code> header file.

<b>Name</b>	<b>Description</b>
originatingQmgrTimestamp (FTFSTS)	Contains the date and time of the most recent status update from the FTF Manager. The FTFSTS data type is defined in the <code>ftfc.h</code> header file. The value returned is of type <code>time_t</code> returned from the <code>time()</code> function.
sourceQmgr (FTFCHAR)	Pointer to a variable that contains the name of the queue manager where the source file was processed by the FTF Sender for the data-transfer request associated with the FTFID.
sourceQmgrStatus (FTFStatus)	Contains the most recent status update from the FTF Sender for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the FTF subsystem. It is an enumerated data type that is defined in the <code>ftfc.h</code> header file.
sourceQmgrTimestamp (FTFSTS)	Contains the date and time of the most recent status update from the FTF Sender. The FTFSTS data type is defined in the <code>ftfc.h</code> header file. The value returned is of type <code>time_t</code> returned from the <code>time()</code> function.
sourceFilename (FTFCHAR)	Pointer to a variable that contains the fully qualified path and filename of the source file used in the data-transfer request.
targetQmgr (FTFCHAR)	Pointer to a variable that contains the name of the queue manager to which data is sent. The FTF Receiver is attached to this queue manager.
targetQmgrStatus (FTFStatus)	Contains the most recent status update from the FTF Receiver for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the FTF subsystem. It is an enumerated data type that is defined in the <code>ftfc.h</code> header file.
targetQmgrTimestamp (FTFSTS)	Contains the date and time of the most recent status update from the FTF Receiver. The FTFSTS data type is defined in the <code>ftfc.h</code> header file. The value returned is of type <code>time_t</code> returned from the <code>time()</code> function.
targetFilename (FTFCHAR)	Pointer to a variable that contains the fully qualified path and filename of the target file used in the data-transfer request.
requestingQmgrStatus (FTFStatus)	Contains the most recent status update from the component that requested the data-transfer. The FTFStatus data type represents all possible statuses in the FTF subsystem. It is an enumerated data type that is defined in the <code>ftfc.h</code> header file.

<b>Name</b>	<b>Description</b>
requestingQmgrTimestamp (FTFSTS)	Contains the date and time of the status update from the FTF transaction requesting component. The FTFSTS data type is defined in the ftfc.h header file. The value returned is of type time_t returned from the time() function.
groupName (FTFCHAR)	Reserved.
label (FTFCHAR)	Contains the user-defined label that is associated with the data-transfer request.
Ids (FTFIdentifiersInfo)	Contains the three user-defined fields which are associated with the data-transfer request.
priority (FTFLONG)	Contains the priority code associated with the data-transfer request.
persistent (FTFBOOL)	Contains the persistence code associated with the data-transfer request
bytesSent(FTFLONG)	Contains a count of the number bytes of data in the data-transfer request.
numberOfMessages (FTFLONG)	Contains a count of the number of message that make up the data-transfer request.
transferTime (FTFLONG)	Contains the time, in seconds, that has elapsed since the data-transfer request has begun.

## Related Functions

- FTFStatusFreeSummaryList (page 200)
- FTFStatusGetSummaryList (page 204)

## Related Data Structures

- FTFStatusSummaryFilter (page 294)

## Data Structure

```
typedef struct _FTFStatSummary {
    FTFCHAR    ftfId[FTF_ID_SIZE];
    FTFSTS     timestamp;
    FTFCODE    ftfcode;
```

```
FTFCODE      src;
FTFCHAR      localQmgr[FTF_NAME_SIZE];

FTFCHAR      originatingQmgr[FTF_NAME_SIZE];
FTFStatus    originatingQmgrStatus;
FTFTS        originatingQmgrTimestamp;

FTFCHAR      sourceQmgr[FTF_NAME_SIZE];
FTFStatus    sourceQmgrStatus;
FTFTS        sourceQmgrTimestamp;
FTFCHAR      sourceFilename[FTF_MAX_STRING_SIZE];

FTFCHAR      targetQmgr[FTF_NAME_SIZE];
FTFStatus    targetQmgrStatus;
FTFTS        targetQmgrTimestamp;
FTFCHAR      targetFilename[FTF_MAX_STRING_SIZE];

FTFStatus    requestingQmgrStatus;
FTFTS        requestingQmgrTimestamp;

FTFCHAR      groupName[FTF_NAME_SIZE];
FTFCHAR      label[FTF_NAME_SIZE];

FTFIdentifiersInfo Ids;
FTFLONG      priority;
FTFBOOL      persistent;
FTFLONG      bytesSent;
FTFLONG      numberOfMessages;
FTFLONG      transferTime;

} FTFStatSummary;
```

# FTFStatSummaryList

## Description

The FTFStatSummaryList data structure contains summary status information for data-transfer requests. It contains a summary record for each data-transfer request that meets the filter criteria. Each summary record contains the most recent status update from each FTF component.

## Data Elements

The following table lists and describes the FTFStatSummaryList data structure's elements.

Name	Description
cRows (FTFLONG)	Contains the number of returned data-transfer request summary records.
pRows (FTFStatSummary*)	Pointer to a variable that contains the FTFStatSummary data type. The FTFStatSummary data structure contains data-transfer request summary status records.

## Related Functions

- FTFStatusFreeSummaryList (page 200)
- FTFStatusGetSummaryList (page 204)

## Related Data Structures

- FTFStatSummary (page 282)
- FTFStatusSummaryFilter (page 294)

## Data Structure

```
typedef struct _FTFStatSummaryList
{
    FTFLONG          cRows;
    FTFStatSummary   *pRows;
} FTFStatSummaryList;
```

# FTFStatusDeleteInfo

## Description

The FTFStatusDeleteInfo data structure contains the information that determines which status records are deleted by the FTFStatusDelete API.

## Data Elements

The following table lists and describes the FTFStatusDeleteInfo data structure’s elements.

Name	Description
lqm (FTFCHAR)	Contains the name of the lqm from which the delete operation occurs and for which status records are deleted.
cFile (FTFCHAR)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue.
cq (FTFCHAR)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.
ftfid (FTFCHAR)	Contains the FTFID for which status information is deleted.

## Related Functions

- FTFStatusDelete (page 196)

## Related Data Structures

None



## Data Structure

```
typedef struct _FTFStatusDeleteInfo {  
    FTFCHAR lqm      [MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR cFile    [FTF_MAX_PATH        + 1];  
    FTFCHAR cq       [MQ_Q_NAME_LENGTH    + 1];  
    FTFCHAR ftfid    [FTF_ID_SIZE         ];  
} FTFStatusDeleteInfo;
```

# FTFStatusGetDetailListInfo

## Description

The FTFStatusGetDetailListInfo data structure contains the information that determines what detailed status information is returned by the FTFStatusGetDetailList API.

## Data Elements

The FTFStatusGetDetailListInfo data structure contains the following data elements:

Name	Description
lqm (FTFCHAR)	Contains the name of the lqm from which the set status operations occurs and for which detailed status information is retrieved.
cFile (FTFCHAR)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue.
cq (FTFCHAR)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.
ftfid (FTFCHAR)	Contains the FTFID for which detail status information is retrieved.

## Related Functions

- [FTFStatusGetDetailList](#) (page 202)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFStatusGetDetailListInfo {  
    FTFCHAR lqm      [MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR cFile    [FTF_MAX_PATH          + 1];  
    FTFCHAR cq       [MQ_Q_NAME_LENGTH      + 1];  
    FTFCHAR ftfid    [FTF_ID_SIZE           ];  
} FTFStatusGetDetailListInfo;
```

# FTFStatusGetSummaryListInfo

## Description

The FTFStatusGetSummaryListInfo data structure contains the information that determines what summary status information is returned by the FTFStatusGetSummaryList API.

## Data Elements

The following table lists and describes the FTFStatusGetSummaryListInfo data structure’s elements.

Name	Description
lqm (FTFCHAR)	Contains the name of the lqm from which the set status operations occurs and for which detailed status information is retrieved.
cFile (FTFCHAR)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue.
cq (FTFCHAR)	Contains the name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.

## Related Functions

- FTFStatusGetSummaryList (page 204)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFStatusGetSummaryListInfo {  
    FTFCHAR lqm      [MQ_Q_MGR_NAME_LENGTH + 1];  
    FTFCHAR cFile    [FTF_MAX_PATH          + 1];  
    FTFCHAR cq       [MQ_Q_NAME_LENGTH      + 1];  
} FTFStatusGetSummaryListInfo;
```

# FTFStatusSummaryFilter

## Description

The FTFStatusSummaryFilter data structure contains the filter information required for the FTFStatusGetSummaryList API. The FTFStatusGetSummaryList API retrieves the most recent status of the FTF components for all data-transfer requests that meet the filter requirements defined by this data structure.

Before you populate FTFStatusSummaryFilter, initialize its elements to null values.

## Data Elements

The FTFStatusSummaryFilter data structure contains the following data elements.

Elements	Description
ftfId (FTFCHAR[ ])	The unique FTFID that matches the data-transfer request for which status information is retrieved.
statusIgnoreActive (FTFLONG)	A flag that tells the API whether to ignore data-transfer request statuses with a type of <i>Active</i> . <b>Valid values:</b> 1 (ignore this status type), 0 (include this status type)
statusIgnoreComplete (FTFLONG)	A flag that tells the API to ignore data-transfer request statuses with a type of <i>Complete</i> . <b>Valid values:</b> 1 (ignore this status type), 0 (include this status type)
statusIgnoreFailed (FTFLONG)	A flag that tells the API to ignore data-transfer request statuses with a type of <i>Failed</i> . <b>Valid values:</b> 1 (ignore this status type), 0 (include this status type)
statusIgnoreCancelled (FTFLONG)	A flag that tells the API to ignore data-transfer request statuses with a type of <i>Canceled</i> . <b>Valid values:</b> 1 (ignore this status type), 0 (include this status type)

Elements	Description
statusIgnoreExpired (FTFLONG)	A flag that tells the API to ignore data-transfer request statuses with a type of <i>Expired</i> . I <b>Valid values:</b> 1 (ignore this status type), 0 (include this status type)
localQmgr (FTFCHAR[ ])	Reserved.
caseSensitive (FTFBOOL)	A flag that tells the API to ignore the case of the sourceFilename and the targetFilename data elements.
originatingQmgr (FTFCHAR[ ])	Pointer to a variable that contains the name of the oqm. (The FTF Manager is connected to the oqm.) Only records with the specified oqm are included in the returned status information.
originatingStartTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the earliest data-transfer start date and time included in the returned status information. The FTFStatusGetSummaryList API accepts partial date and time input. <b>Note:</b> This field is mandatory. <b>Format:</b> YYYYMMDDhhmmss
originatingEndTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the latest data-transfer start date and time included in the returned status information. The FTFStatusGetSummaryList API accepts partial date and time input. <b>Format:</b> YYYYMMDDhhmmss
sourceQmgr (FTFCHAR[ ])	Pointer to a variable that contains the name of the sqm. (The FTF Sender is connected to the sqm.) Only records with the specified sqm are included in the returned status information.
sourceQmgrStartTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the earliest time and date at which the data-transfer request started at the FTF Sender. Only records with a value greater than the value stored in this data element are included in the returned status information. <b>Format:</b> YYYYMMDDhhmmss

## C Data Structures

### *FTFStatusSummaryFilter*

Elements	Description
sourceQmgrEndTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the latest time and date at which the data-transfer request started at the FTF Sender. Only records with a value less than the value stored in this data element are included in the returned status information. <b>Format:</b> YYYYMMDDhhmmss
sourceFilename (FTFCHAR[ ])	Pointer a variable that contains the fully qualified path and filename of the source file used in the data-transfer request. Only records that match the source file value are included in the returned status information.
targetQmgr (FTFCHAR[ ])	Pointer to a variable that contains the name of the dqm. (The FTF Receiver is connected to the dqm.) Only records with the specified dqm are included in the returned status information.
targetQmgrStartTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the earliest time and date at which the data-transfer request started at the FTF Receiver. Only records with a value greater than the value stored in this data element are included in the returned status information. <b>Format:</b> YYYYMMDDhhmmss
targetQmgrEndTimestamp (FTFCHAR[ ])	Pointer to a variable that contains the latest time and date at which the data-transfer request started at the FTF Receiver. Only records with a value less than the value stored in this data element are included in the returned status information. <b>Format:</b> YYYYMMDDhhmmss
targetFilename (FTFCHAR[ ])	Pointer to a variable that contains the fully qualified path and filename of the target file used in the data-transfer request. Only records that match the target file value are included in the returned status information.
groupName (FTFCHAR[ ])	Reserved.
label (FTFCHAR[ ])	Pointer to a user-defined label that is associated with the data-transfer requests. Only records that match the label value are included in the returned status information.



## Related Functions

- FTFStatusGetSummaryList (page 204)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFStatusSummaryFilter {
    FTFCHAR    ftfId[FTF_ID_SIZE];
    FTFLONG    statusIgnoreActive;
    FTFLONG    statusIgnoreComplete;
    FTFLONG    statusIgnoreFailed;
    FTFLONG    statusIgnoreCancelled;
    FTFLONG    statusIgnoreExpired;
    FTFCHAR    localQmgr[FTF_NAME_SIZE];
    FTFBOOL    caseSensitive;

    FTFCHAR    originatingQmgr[FTF_NAME_SIZE];
    FTFCHAR    originatingStartTimestamp[FTF_NAME_SIZE];
    FTFCHAR    originatingEndTimestamp[FTF_NAME_SIZE];

    FTFCHAR    sourceQmgr[FTF_NAME_SIZE];
    FTFCHAR    sourceQmgrStartTimestamp[FTF_NAME_SIZE];
    FTFCHAR    sourceQmgrEndTimestamp[FTF_NAME_SIZE];
    FTFCHAR    sourceFilename[FTF_MAX_STRING_SIZE];

    FTFCHAR    targetQmgr[FTF_NAME_SIZE];
    FTFCHAR    targetQmgrStartTimestamp[FTF_NAME_SIZE];
    FTFCHAR    targetQmgrEndTimestamp[FTF_NAME_SIZE];
    FTFCHAR    targetFilename[FTF_MAX_STRING_SIZE];

    FTFCHAR    groupName[FTF_NAME_SIZE];
    FTFCHAR    label[FTF_NAME_SIZE];
} FTFStatusSummaryFilter;
```

## FTFSubscribeMsgInfo

FTFSubscribeMsgInfo data structure is associated with the FTF publish/subscribe function. This is the input into the FTFSUB API.

### Description

The FTFSubscribeMsgInfo data structure contains the subscription criteria and target. It also can specify subscription administration information such as query and purge.

### Data Elements

The FTFSubscribeMsgInfo data structure contains the following data elements:

Name	Description
qmgrs (FTFQMgrs)	Represents the queue managers involved (destination queue manager and broker queue manager).
targetFile (FTFTargetFile)	Represents the target file where the published data is stored.
job (FTFJobInfo)	Contains information about a job used by publish/subscribe.
userInfo (FTFUserInfo)	Contains information about the user that is used by publish/subscribe.
isReply (FTFBOOL)	Determines whether it is necessary to wait for a reply.
timeout (FTFULONG)	Contains the length of time to wait for a reply from the FTF Manager.
replyWaitTime (FTFULONG)	Determines the length of time (in seconds) to wait for a reply before control is returned to the calling application. If a response is received before the timeout is exhausted, the appropriate reply and control will be returned to the calling interface.
*configFile (FTFCHAR)	Pointer to a variable that contains the fully qualified path to the FTF configuration file. This element is not required if you specify a configuration queue. Otherwise, it is optional if the FTF_CONFIG_FILE environment variable is set.
cExitInfo (FTFLONG)	The number of exits to be invoked during the data-transfer request.
*exitInfo (FTFEXITINFO)	Pointer to a variable that contains a data structure governing how user exits are invoked and executed.

<b>Name</b>	<b>Description</b>
as400 (FTFAS400FileInfo)	A data structure that describes information about the target files to be written to the AS/400 platform.
*cq (FTFCHAR)	Name of the configuration queue used to store configuration file information. This data element must be populated if you are using a configuration queue.
action (FTFSubReq)	Contains a predefined action code.
subid (FTFCHAR[ ])	The generated subscription ID.
purgeid (FTFCHAR[ ])	The subscription ID that is to be purged.
*pFileName (FTFCHAR)	Name of the posted file.
subStringCount (FTFLONG)	The number of topic strings.
*pSubString (FTFCHAR)	A list of the topics.
dqmCount (FTFLONG)	The number of subscribers.
*dqmList (FTFCHAR)	A list of the subscribers that are to receive a posting.

## **Related Functions**

- [FTFStatusGetSummaryList](#) (page 204)

## **Related Data Structures**

None

**Data Structure**

```
typedef struct _FTFSubscribeMsgInfo {
    FTFQMgrsInfo      qmgrs;
    FTFTargetFileInfo targetFile;
    FTFJobInfo        job;
    FTFUserInfo       userInfo;
    FTFBOOL           isReply;
    FTFULONG          timeout;
    FTFULONG          replyWaitTime;
    FTFCHAR           *configFile;
    FTFULONG          cExitInfo;
    FTFEXITINFO       *exitInfo;
    FTFAS400FileInfo  as400;
    FTFCHAR           *cq;
    FTFSubReq         action;
    FTFCHAR           subid [FTF_ID_SIZE];
    FTFCHAR           purgeid [FTF_ID_SIZE];
    FTFCHAR           *pFileName;
    FTFULONG          subStringCount;
    FTFCHAR           *pSubString;
    FTFULONG          dqmCount;
    FTFCHAR           *dqmList;
} FTFSubscribeMsgInfo;
```

# FTFSubscriptionListMsgInfo

FTFSubscriptionListMsgInfo data structure is associated with the FTF publish/subscribe function.

## Description

The FTFSubscriptionListMsgInfo data structure contains a pointer to the list of subscriptions and is used during queries of existing subscriptions.

## Data Elements

The FTFSubscriptionListMsgInfo data structure contains the following data elements:

Name	Description
cRows (FTFLONG)	The number of subscriptions that were returned to the subscriber.
*pRows (FTFSubscriptionMsgInfo)	Contains the pointer to the list of subscriptions.

## Related Functions

- FTFSubscriptionMsgInfo (page 302)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFSubscriptionListMsgInfo {
    FTFLONG          cRows;
    FTFSubscriptionMsgInfo *pRows
} FTFSubscriptionListMsgInfo;
```

## FTFSubscriptionMsgInfo

FTFSubscriptionMsgInfo data structure is associated with the FTF publish/subscribe function.

### Description

The FTFSubscriptionMsgInfo data structure contains the detailed information about the subscription returned, including target data.

### Data Elements

The FTFSubscriptionMsgInfo data structure contains the following data elements:

Name	Description
subid (FTFCHAR[ ])	Contains the subscription ID.
targetFile (FTFTargetFileInfo)	Target file information of the subscription.
job (FTFJobInfo)	The job information.
userInfo (FTFUserInfo)	The user information.
cExitInfo (FTFLONG)	The number of exits.
*exitInfo (FTFEXITINFO)	A list of any exits associated with the subscription.
as400 (FTFAS400FileInfo)	A data structure that describes information about the target files to be written to the AS/400 platform.
*pSubString (FTFCHAR)	Topic of the subscription.
dqmCount (FTFLONG)	The number of subscribers.
*dqmList (FTFCHAR)	A pointer to the list of subscribers that are to receive a posting.
*dqmBrk (FTFCHAR)	Name of the broker queue manager that is handling the subscription.

### Related Functions

- FTFSubscriptionListMsgInfo (page 301)

## Related Data Structures

None

## Data Structure

```
typedef struct _FTFSubscriptionMsgInfo {
    FTFCHAR          subid [FTF_ID_SIZE];
    FTFTargetFileInfo targetFile;
    FTFJobInfo        job;
    FTFUserInfo        userInfo;
    FTFLONG           cExitInfo;
    FTFEXITINFO        *exitInfo;
    FTFAS400FileInfo   as400;
    FTFCHAR            *pSubString;
    FTFLONG            dqmCount;
    FTFCHAR            *dqmList;
    FTFCHAR            *dqmBrk;
} FTFSubscriptionMsgInfo;
```

# FTFTargetFileInfo

## Description

The FTFTargetFileInfo data structure contains the information required for describing the target file and its location, attributes, and allocation rules.

### Specifying an Esoteric Unit Name

To specify an esoteric name for the OS/390 UNIT value, follow these steps:

- Do not set a value in the MVSVOLUME stanza.
- Specify the *unit* value in the **unitName** data element or in the MVSUNITNAME stanza in the FTF configuration file on either the FTF Sender or the FTF Receiver.

## Data Elements

The following table lists and describes the FTFTargetFileInfo data structure’s elements:

Name	Description
pFileName (FTFCHAR*)	Pointer to a variable that contains the target file’s fully qualified path and name. This path must be syntactically correct for the target platform that will be depositing the file.
isCompressed (FTFBOOL)	Reflects whether inbound data is compressed or uncompressed. <b>Valid values:</b> 1 (compressed), 0 (uncompressed).
fileType (FTFFileTypeInfo)	Reflects the format of the data that is being transmitted. If you select a binary format, no conversion or formatting takes place. If you select a text format, data-type translation occurs as is necessary for the particular code page of the target platform. <b>Valid values:</b> FTF_BINARY (binary), FTF_TEXT (text).
fileMode (FTFFileModeInfo)	Determines how the data is processed at the destination. <b>Valid values:</b> <ul style="list-style-type: none"><li>• FTF_CREATE_FILE (create a new file),</li><li>• FTF_APPEND_FILE (append to an existing file),</li><li>• FTF_NOREPLACE_FILE (do not replace an existing file).</li></ul>



<b>Name</b>	<b>Description</b>
fileOrg (FTFFileOrgInfo)	Determines the target file's OS/390 file organization. <b>Valid values:</b> PS (Physical Sequential), PDS (Partitioned Data Set) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
cDirectoryBlocks (FTFLONG)	Determines the number of directory blocks allocated for the target file on OS/390.
recordFormat (FTFRecordFormatInfo)	Determines the target file's OS/390 record format. <b>Valid values:</b> F (fixed), V (variable), FB (fixed block), and VB (variable block) <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
lrecl (FTFLONG)	Determines the target file's OS/390 logical record length. <b>Valid values:</b> 1 - 32760. <b>Note:</b> See the following paragraph, <i>OS/390 File Allocation Items</i> .
blockSize (FTFLONG)	Determines the target file's OS/390 block size. Specifying a block size of 0 enables the system to choose the optimum block size for the dataset during allocation. If the record format is fixed block (FB), the block size must be a multiple of the logical record length value, (the lrecl data element). When the record format is variable length (VB), the blksize value must be at least four bytes greater than the lrecl value. <b>Valid values:</b> 0 - 32760
unitName (FTFCHAR)	Determines the target file's OS/390 unit name.
volser (FTFCHAR)	Determines the target file's volume serial number.
allocationUnit (FTFAllocUnitInfo)	Determines the target file's OS/390 allocation unit. <b>Valid values:</b> CYL (cylinder), BLK (block), TRK (track)
primaryAllocSize (FTFLONG)	Determines the target file's OS/390 primary allocation unit size.
secondaryAllocSize (FTFLONG)	Determines the target file's OS/390 secondary allocation unit size.
textWrapRecord (FTFTextFileWrapInfo)	Indicates how records are processed when they are longer than the specified target-file record length. <b>Valid values:</b> <ul style="list-style-type: none"> <li>• FTFREQ_WRAP (wraps records)</li> <li>• FTFREQ_FAIL (causes the data transfer to fail)</li> <li>• FTFREQ_TRUNC (truncates records at the specified record length)</li> </ul>

Name	Description
createDirectory (FTFBOOL)	Determines whether the specified target directory is created when it does not already exist. <b>Valid values:</b> 1 (create directory), 0 (do not create a directory).
isDataPersistent (FTFBOOL)	Determines whether the data is persistent or nonpersistent. Persistent data survives a system restart; nonpersistent data does not. If the data-transfer request requires recovery of the data and the data is not persistent, you must reaccess it before you can send it again. <b>Valid values:</b> 1 (transmission is persistent), 0 (transmission not persistent).
modelDataset (FTFCHAR)	Defines a dataset for GDG allocation. The data element contains definitions of items such as record length, block size, the maximum number of supported revisions form the attributes group These definitions form the basis for creating a GDG during processing <b>Valid values:</b> up to 44 characters
dType (FTFCHAR)	Determines the file type for a data-transfer request in which data is not stored in files. This value is required if you want to receive data that will not be stored in a file. The valid values for this data element are based on the values specified in the FTF configuration file.
cdtypeData (FTFLONG)	Determines the length in bytes of the *pdtypeData data element.
pdtypeData (FTFCHAR*)	Contains the receiver data that will not be stored in a file.
bufNo (FTFLONG)	Determines the number of buffers used during a data transfer request on the OS/390 platform.

## OS/390 File Allocation Items

The items listed in the FTFExitTargetFileInfo data structure represent the rules governing how files are dynamically file allocated on OS/390. These items are optional. Entries in the FTF configuration file fill in the missing entries. If items are specified in the FTF configuration file at the source FTF Sender's node or the target FTF Receiver's node, they are resolved at those specific points respectively. The items provided through the API take precedence. Next, items are resolved at the FTF Sender. Any outstanding items are attempted for resolution at the FTF Receiver.

## Related Functions

- FTFReq (page 187)

## Related Data Structures

- FTFCExitRequestInfo (page 234)
- FTFCRequestMsgInfo (page 256)

## Data Structure

```
typedef struct _FTFTargetFileInfo {
    FTFCCHAR          * pFileName;
    FTFCBOOL          isCompressed;
    FTFCFileTypeInfo  fileType;
    FTFCFileModeInfo  fileMode;
    FTFCFileOrgInfo   fileOrg;
    FTFCFLONG          cDirectoryBlocks;
    FTFCRecordFormatInfo recordFormat;
    FTFCFLONG          lrecl;
    FTFCFLONG          blockSize;
    FTFCCHAR          unitName [FTCFREQ_UNITNAME_SIZE];
    FTFCCHAR          volser   [FTCFREQ_VOLSER_SIZE];
    FTFCAllocUnitInfo allocationUnit;
    FTFCFLONG          primaryAllocSize;
    FTFCFLONG          secondaryAllocSize;
    FTFCTextFileWrapInfo textWrapRecord;
    FTFCBOOL          createDirectory;
    FTFCBOOL          isDataPersistent;
    FTFCCHAR          modelDataset[FTCFREQ_DATASETNAME_SIZE];
    FTFCCHAR          dtype [FTCF_FILETYPE_SIZE];
    FTFCFLONG          cdtypeData;
    FTFCCHAR          *pdtypeData;
    FTFCFLONG          bufNo;
} FTFTargetFileInfo;
```

## FTFUserInfo

### Description

The FTFUserInfo data structure allows you to specify group and label information about the current data-transfer request.

### Data Elements

The following table lists the FTFUserInfo data structure's elements.

Name	Description
groupName (FTFCHAR[ ])	Name of the group to which this data-transfer request belongs.
label (FTFCHAR[ ])	A user-specified label that can be used in FTF status queries. This value has no bearing on the internal processing of data-transfer request. It is used for output and query specifications.
replyQ (FTFCHAR[ ])	The queue to which reply messages are to be routed.
replyQmgr (FTFCHAR[ ])	The queue manager to which reply messages are to be routed.
notifyStatus (FTFNotifyStatus)	Defines when a notification message will be sent to the Notify Queue. If you specify this argument, you must also specify <i>Notify Data</i> and <i>Notify Type</i> arguments.
notifyType (FTFCHAR[ ])	Specifies the user-defined method used to deliver a notification message. If you specify this argument, you must also specify <i>Notify Data</i> and <i>Notify Status</i> arguments.
*pNotifyData (FTFCHAR)	Pointer to the user-defined data that specifies the information, such as e-mail, pager, or fax numbers, that is used to deliver a notification message. If you specify this argument, you must also specify <i>Notify Status</i> and <i>Notify Type</i> arguments.
*pSubString (FTFCHAR)	The topic string under which a document is published.

### Related Functions

- FTFReq (page 187)

## Related Data Structures

- FTFExitUserInfo (page 245)
- FTFRequestMsgInfo (page 256)

## Data Structure

```
typedef struct _FTFUserInfo {
    FTFCHAR      groupName[FTFREQ_GROUPNAME_SIZE];
    FTFCHAR      label[FTFREQ_LABEL_SIZE];
    FTFCHAR      replyQ[MQ_Q_NAME_LENGTH + 1];
    FTFCHAR      replyQmgr
                  [MQ_Q_MGR_NAME_LENGTH + 1];
    FTFNotifyStatus notifyStatus;
    FTFCHAR      notifyType
                  [FTF_MAX_NOTIFY_TYPE_SIZE];
    FTFCHAR      *pNotifyData;
    FTFCHAR      *pSubString;
} FTFUserInfo;
```



# **Tivoli Data Exchange and MQSeries Security**

This chapter describes the security used by Tivoli Data Exchange and MQSeries. It contains the following sections:

<b>Section</b>	<b>Page</b>
Introduction	312
MQSeries Security Enabling Interface	313
Enabling Security for Tivoli Data Exchange	315

## **Assumptions**

This chapter makes the following assumptions:

- You have a working knowledge of TDE queues, queue managers, and data transfers.
- You have an understanding of MQSeries security procedures.

## **Introduction**

This chapter discusses the security considerations required when running TDE in a multi-platform production environment. While the focus is on OS/390, the concepts presented can be easily adapted to any platform on which MQSeries is capable of running.

TDE is a powerful file transfer utility that enables enterprises to easily move large volumes of data across multiple platforms within the enterprise and with their trading partners. TDE utilizes MQSeries as the underlying transport mechanism to move the data. Care must be taken to ensure that files may only be accessed by properly authorized users.

TDE runs as a server in the context that it processes requests from unknown parties to move files from one location to another. These requests may come from users on the same platform as TDE or from remote platforms. As such, the TDE server must run at a sufficiently privileged state to access a broad spectrum of data while ensuring that access to the files is restricted to those individual users that are authorized. TDE provides facilities to ensure only authorized users may access data. This is accomplished by using MQSeries direct interface to the external security manager through the Security Enabling Interface and TDE's interface to the external security manager through the OS/390 Security Access Facility, also known as the SAF interface.

## **Modes of Security**

TDE may be run in one of three modes:

### **No Security**

No security enabled. This means that any user can access any file that TDE can access.

### **Level One Security**

Security is enabled either for the MQSeries objects that are used by TDE or at the file level. Tivoli Data Exchange recommends if level one security only is enabled that it is done at the file level whenever possible. An example where file level security cannot be accomplished is on a Windows/NT server utilizing the FAT file system. The Windows/NT security interface does not support this file system.



## **Level Two Security**

Level two security provides dual protection of resources accessed by TDE for users. It uses the access controls supported by MQSeries for the TDE objects and the access controls provided by the operating system for the file resources. OS/390 provides the SAF interface for this purpose. This interface enables MQSeries to validate whether the inbound request has access to the TDE objects. If so, then TDE validates the user authority to access the file resource. If the user is authorized to access both sets of resources, then the file transfer proceeds. If either authorization check fails, the request fails.

## **MQSeries Security Enabling Interface**

The MQSeries security enabling interface component within MQSeries provides security services for access to MQSeries objects. The security interface allows authentication and access at the object level (queues, channels) and command level. This interface allows you to enable access control of MQSeries objects through the access control facilities of a given platform. For example, access control on OS/390 might be done using RACF or a similar product. On Windows/NT, access control is accomplished with MQSeries commands (SETMQAUT), and access is controlled through an MQSeries facility known as the Object Authority Manager.

## **General MQSeries Security Concepts**

---

### **Note:**

The following discussion is general in nature. Implementing security for MQSeries is platform specific.

---

Because MQSeries handles the transfer of information that is potentially valuable, it needs the safeguard of a security system. The security levels ensure that the resources MQSeries owns and manages are protected from unauthorized access, which may lead to loss or disclosure of the information. It is essential that none of the following is accessed or changed by any unauthorized user or process:

- Connections to MQSeries
- MQSeries objects such as queues, processes, and namelists (OS/390 ONLY)

## **Tivoli Data Exchange and MQSeries Security**

### *MQSeries Security Enabling Interface*

- MQSeries transmission links
- MQSeries system control commands
- MQSeries messages
- Context information associated with messages

While the issues of concern are the same for all platforms supported by MQSeries, the implementation may differ between platforms. MQSeries for OS/390 uses the OS/390 system authorization facility. Since the length of the true name of objects can be greater than the length of object names that can be validated on many platforms, the method of protecting MQSeries objects can differ substantially.

## **MQSeries Alternate User Authority**

TDE uses the Alternate User Authority facility, when enabled, to ensure that a user can access the TDE objects. This feature enables MQSeries on behalf of TDE to validate whether a user (as defined in the MQMD of the control message) has access to TDE objects. If the userID of the inbound request to the TDE is authorized to the TDE objects, the request is allowed to continue. If not, the request fails with a TDE return code 720 (FTFRCE\_SECURITY\_ERROR).

## **Tivoli Data Exchange External Security Manager**

The Sender and Receiver TDE components are enabled via exits to validate a user's access to a file resource. An External Security Manager exit is provided for OS/390 platforms that validates access to a file resource through the System Authentication Facility (SAF) interface. When a request is received, the userID provided in the MQSeries Message Descriptor (MQMD) of the control message is used to validate whether access to the file resource is allowed. If the user is authorized to access the file resource, then the TDE component accesses the file on behalf of the user and the request proceeds. If not, then the request fails with a TDE return code 720 (FTFRCE\_SECURITY\_ERROR).

# Enabling Security for Tivoli Data Exchange

## Enabling Alternate User Authority

To allow MQSeries alternate user authority checking on behalf of TDE, the customer must enable this feature at the startup of TDE. This is done by setting the parameter OAMSecurity=On in the configuration file for TDE. By default, OAMSecurity is disabled. The customer must enable access to Alternate User Authority checking facilities to TDE using the appropriate facilities on the platform that TDE is running on. Please refer to the MQSeries *System Management Guide* for the platform MQSeries and TDE is running on.

## Enabling File Resource Security

See the *Tivoli Data Exchange Installation Guide*.

## Limiting User Access Using Queue Permissions

When OAMSecurity is disabled, the person requesting must have access to the following queues:

- FTFSTAT.CONTROL
- FTFSTAT.DETAIL
- FTFMGR.CONTROL

The person must have access to perform a put to the status queues as well as putting the initial request message to the manager's control queue. The userID that started the TDE components must have access to all of the queues that TDE uses.

When OAMSecurity is enabled, the Requesting User must have access to the following queues:

- FTFRCV.CONTROL
- FTFRCV.SYNC
- FTFRCV.STAGE
- FTFRCV.SYSTEM

## **Tivoli Data Exchange and MQSeries Security**

### *Enabling Security for Tivoli Data Exchange*

- FTFSDR.CONTROL
- FTFSDR.SYNC
- FTFSDR.STAGE
- FTFSDR.STAGE.CONTROL
- FTFSDR.SYSTEM
- FTFMGR.CONTROL
- FTFMGR.SYNC
- FTFSTAT.CONTROL
- FTFSTAT.DETAIL
- FTFDATA
- FTFICC
- FTFLOG
- FTFCFG

The userID that started the TDE components must still have access to all of these queues as well.

In order to limit a user from writing a file to a machine with TDE, you can take away the access to the FTFDATA queue on the queue manager of the machine. This can only be done when OAMSecurity is set to ON. In addition, the user must be defined on the machine, but not in a group that has access to the queue. For example, if the userID “User1” is in the “USERS” group, then the only way to limit User1’s access to a queue is to also limit the access of the USERS group.

Conversely, you cannot grant queue access to a user that is not defined on the machine.

To grant all access for queue FTFDATA to user “USER1,” enter the following MQSeries command:

```
setmqaut -m <QMGR> -n FTFDATA -t queue -p USER1 +all
```

## **Tivoli Data Exchange and MQSeries Security** *Enabling Security for Tivoli Data Exchange*

To display the access for user “USER1” on queue FTTFDATA, use the following MQSeries command:

```
dspmqaaut -m <QMGR> -n FTTFDATA -t queue -p USER1
```

### **OS/390**

For OS/390, the following must be performed for the userID under which the TDE address space executes:

```
permit QMGR.FTF.** class(MQQUEUE) id(FTF address  
space userid) access(update)  
permit QMGR.CONTEXT class(MQADMIN) id(FTF address  
space userid) access(control)  
permit QMGR.ALTERNATE.USER.* class(MQADMIN) id(FTF  
address space userid) access(update)
```

For each user who is granted access to perform file transfers, the following must be performed:

```
permit QMGR.FTFMGR.CONTROL class(MQQUEUE)  
id(user-id) access(update)  
permit QMGR.FTFSDR.CONTROL class(MQQUEUE)  
id(user-id) access(update)  
permit QMGR.FTFRCV.CONTROL class(MQQUEUE)  
id(user-id) access(update)  
permit QMGR.FTFSTAT.CONTROL class(MQQUEUE)  
id(user-id) access(update)  
permit QMGR.FTFSTAT.DETAIL class(MQQUEUE)  
id(user-id) access(update)  
permit QMGR.FTFDATA.** class(MQQUEUE) id(user-id)  
access(update)
```

This procedure may be required depending on the settings of the remote platform. If alternate user authority is not used, no TDE security is disabled and this statement should be used.

```
permit QMGR.FTFMGR.CONTROL class(MQQUEUE)  
id(user-id) access(update)
```

## **NT and Unix Systems**

This procedure may be required depending on the settings of the remote platform. If alternate user authority is not used, no TDE security is disabled and this statement should be used.

```
setmqaut -m QMGR -n FTFMGR.CONTROL -g anyusergroup  
+put +get
```

With altuser, the setting for the user changes as follows:

```
setmqaut -m QMGR -n FTFMGR.CONTROL -t queue -g  
anyusergroup +put  
+get +browse  
setmqaut -m QMGR -n FTFSDR.CONTROL -t queue -g  
anyusergroup +put  
+get +browse  
setmqaut -m QMGR -n FTFRCV.CONTROL -t queue -g  
anyusergroup +put  
+get +browse  
setmqaut -m QMGR -n FTFSTAT.CONTROL -t queue -g  
anyusergroup +get  
+browse  
setmqaut -m QMGR -n FTFSTAT.DETAILED -t queue -g  
anyusergroup +get  
+browse  
setmqaut -m QMGR -n FTFDATA -t queue -g  
anyusergroup +put +get +browse  
setmqaut -m QMGR -n FTFDATA.1.1 -t queue -g  
anyusergroup +put +get +browse
```

The above is all based on a TDE setup with only one Sender and one Receiver running. If additional instances of the sender and/or receiver are used, additional queues need to be defined and authorizations granted. Refer to the MQSeries *System Administration Guide* for more details about these commands.

---

---

# XML Integration

This chapter describes the characteristics of the XML files that can be submitted to the Tivoli Data Exchange Manager to initiate TDE functions.

The chapter includes the following sections.

Section	Page
Overview	320
Implementation Details	321
Return Objects	324
XML Element Names	325
Examples	337
Sample DTD	339

## Assumptions

This manual makes the following assumptions:

- You have a good working knowledge of IBM MQSeries or access to MQSeries documentation.
- You have a working knowledge of XML and XML tags.

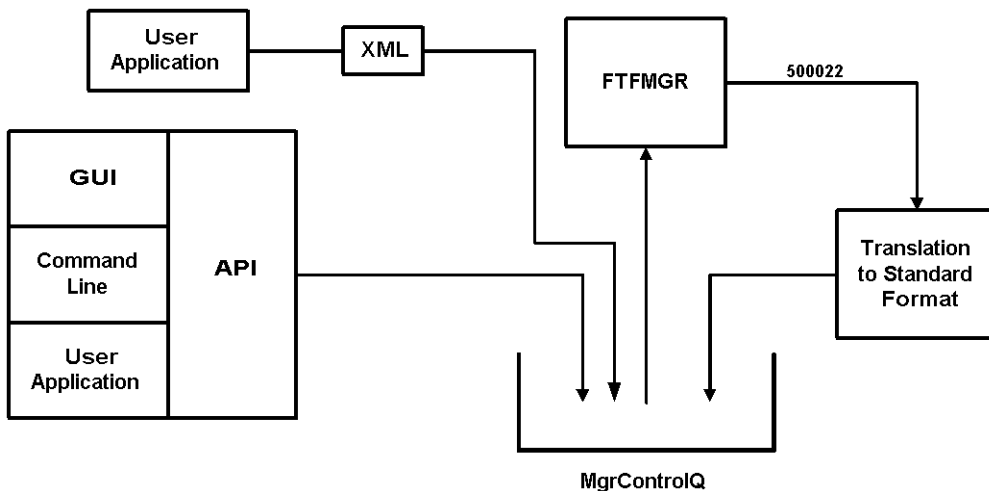
## Operating Requirements

XML integration is supported on the following operating system:

- Win 32

## Overview

The XML Integration allows a user/application to issue TDE commands using eXtensible Markup Language (XML) as the parameter input mechanism. Currently TDE commands can be invoked in two ways. First, via the command line, for interactive invocation; second, via the FTF API, for programmatic invocation. The XML interface is oriented as an alternative programmatic implementation. The current TDE API requires a series of data structures as the parameter input. These data structures are very complex and require the user to have significant understanding of the FTF data structures. Using XML as the input mechanism, we make the application interface much easier to use. The following diagram illustrates the general flow.



If you work with the GUI, command line, or the API, information is formatted into data structures and passed to the MgrControlQ. With the XML Integration invocation, your application passes the XML as the data portion of the MQSeries message and passes it to the MgrControlQ.

You must set the MsgType field in the MQMD (MQSeries Message Descriptor) to the value of 500022, then the FTFMGR recognizes the transaction as an XML document that needs to be processed in a special manner.

Three of the TDE commands are enabled: Request, Ping, and Cancel.



# Implementation Details

## Input

The input data mimics the current command line interface and the options are the XML elements.

### Sample Input

```
<?xml version='1.0'?>
<!DOCTYPE FTFInputData SYSTEM"D:\Ping.dtd" >
<FTFInputData>
  <RequestType>Ping</RequestType>
  <Dqm>QMGRA</Dqm>
  <SqM>QMGRA</SqM>
  <Oqm>QMGRA</Oqm>
  <Timeout>60</Timeout>
  <CFile>D:\TDE_120_DEV\FTFCONFIG.INI</CFile>
</FTFInputData>
```

### Required Fields

- **XML Message Identifier** – The MQSeries Message must assign the Message Descriptor MsgType field. The manager uses this field to uniquely identify the type of message. When using XML integration the mqmd.MsgType = 500022.
- **Request Type Tag** – The XML Element “RequestType” must be present in the XML Data and equal to one of the following values:
  1. Request
  2. Ping
  3. Cancel

## Output

### FTFId Object

The application requesting the transaction may need to know the FTFId of the transaction. If an application needs to know the FTFId of a transaction it should populate the ReplyToQ and ReplyToQMgr fields of the MQMD. If these fields are populated the manager will return the FTFId Object as well as the original XML input to this Queue.

Object Name	Description
FTFIdObject	Object Identifier
FTFId	FTF Identifier
FTFDataObject	Data Object containing input data

### FTFReturnObject

The success or failure of TDE to perform the requested action is indicated with an embedded FTFReturnObject. The format of the FTFReturnObject mimics the TDE FTFCA data structure but in XML format. The FTFReturnObject is embedded in the original request object. This allows the requested action and its return code to be encompassed in one object. The format of the ReturnCode object is listed below:

Field	Description
RC1	Level 1 return code or 0 if successful
RC2	Level 2 return code or 0 if successful
FTFId	FTF identifier for this transaction
FTFErrMsg	Error text if an error occurred, otherwise blank

## **FTFReplyObject**

This object is returned, with the original transaction input when the replyQ and replyQMgr options are specified with a request transaction.

<b>Field</b>	<b>Description</b>
FTFId	FTF identifier for this transaction
TransferStatus	Status of transfer

## **FTFNotifyObject**

This object is returned with the input data when the notify option is specified with a request transaction.

<b>Field</b>	<b>Description</b>
FTFId	FTF identifier for this transaction
TransferStatus	Status of transfer
NotifyStatus	Status that was matched
NotifyType	User-defined method
NotifyData	User-defined data

FTFPingInfo

Field	Description
Lqm	Local queue manager
Oqm	Originating queue manager
Sqm	Source queue manager
Dqm	Destination queue manager
Difftime	Time difference from start to finish
BytesWritten	Number of bytes written

Return Objects

1. If the –wait option is specified as an option on a Request Transaction, the following Object is returned to the Queue specified in the MQSeries Message Descriptor Field ReplyToQ on the Queue Manager specified in the ReplyToQMgr.

```
<FTFReturnObject>
  <FTFId>    </FTFId>
  <RC1>    </RC1>
  <RC2>    </RC2>
  <FTFErrMsg> </FTFErrMsg>
</FTFReturnObject>
```

2. If you populate the MQSeries Message Descriptor Fields ReplyToQ and ReplyToQMgr on any Request transaction, the following Object is returned.

```
<FTFIdObject>
  <FTFId> </FTFId>
  <FTFDataObject>
    .
    Input Data
    .
  </FTFDataObject>
</FTFIdObject>
```

3. If you populate the FTFInputData Object with the ReplyQ and ReplyQMgr elements, the following object is returned to this Queue Manager and Queue upon completion of the Request Transaction.

```
<FTFReplyObject>
  <FTFId> </FTFId>
  <TransferStatus> </TransferStatus>
</FTFReplyObject>
```

4. If you populate the FTFInputData Object with the NotifyStat element, the following object is returned to Queue and Queue Manager specified during the startup of the Manager.

```
<FTFNotifyObject>
  <FTFId> </FTFId>
  <TransferStatus> </TransferStatus>
  <NotifyStatus> </NotifyStatus>
  <NotifyType> </NotifyType>
  <NotifyData> </NotifyData>
</FTFNotifyObject>
```

5. If you request a Ping Transaction the following object is returned to the FTFICC queue upon completion of the Ping Transaction.

```
<FTFPingObject>
  <Lqm> </Lqm>
  <Sqm> </Sqm>
  <Oqm> </Oqm>
  <Dqm> </Dqm>
  <BytesWritten> </BytesWritten>
  <RoundTripTime> </RoundTripTime>
</FTFPingObject>
```

## XML Element Names

When using the XML Integration feature, you need to encode your data within an XML file. Each of the element names listed in the following section would be tags placed in the XML input stream submitted to TDE. All following tags are placed within the FTFInputData root element.

The tags can be used in two ways. If a value needs to be provided as part of using the tag, place the value between the tag and end-tag. An example is as follows:

```
<AlcUnit>CYL</AlcUnit>
<BufNo>5</BufNo>
```

In other cases, you need to use a tag that does not have value, but you want to use a tag to change how a file transfer is processed. The existence of the tag sets the condition on. An example might be as follows:

```
<FromStage></FromStage>
<StagePersist></StagePersist>
```

The sections that follow describe the elements that can be used for each of the transactions.

## Request Transaction Elements

The list that follows contains the elements that can be used with a TDE file transfer request. The RequestType element value must be set to “Request” for this type of transaction.

- **<AlcUnit>** – Determines the allocation unit used for the target on OS/390. **Valid values:** CYL (cylinder), BLK (block), and TRK (track).
- **<AS400FT>** – The value entered specifies the type of file. **Valid values:** \*DFLT (defaults to the value entered in the configuration table) \*SAVE (specifies an AS/400 Save file) \*SRCPF (specifies an AS/400 source physical file).
- **<BlkSize>** – Determines the block size for the target file on OS/390. Specifying a block size of 0 enables the system to choose the optimum block size for the data set during allocation. If the record format is Fixed Block (FB), the block size in the blksize argument must be a multiple of the logical record length, the lrecl parameter. When the record format is Variable Block (VB), the blksize value must be at least four bytes greater than the lrecl value. **Valid values:** 0 - 32760

- **<BufNo>** – Allows you to specify the number of internal buffers that are to be used when processing data transfers. The throughput of a TDE data transfer is governed by a combination of the block size of the data being transferred and the number of buffers that are allocated for transfer in the BufNo argument. **Valid Values:** 1 - 255
- **<CancelMode>** – Provides a command-line override to the preemptive cancel flag in the TDE configuration file. **Valid values:** On, Off
- **<CFile>** – Contains the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no CQ argument is specified, this value must be specified. You cannot specify both a CFile and a CQ argument in the same command.
- **<Compress>** – Specifies that the data being sent is compressed using the internal TDE compression algorithm.
- **<CQ>** – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a CFile and a CQ argument in the same command.
- **<CrtLib>** – Specifies that TDE is to create the specified library if it does not exist. **Valid Values:** Yes, No
- **<CCSid>** – The CCSid is used as the identifier for the data-transfer request. If CCSid is not specified, TDE uses the CCSid of the job. **Valid values:** 1-65535
- **<DData>** – Determines the data output for a transfer that receives data that is not stored in a file.
- **<Delsrc>** – Indicates that the source data is to be deleted once the data-transfer request is completed.
- **<DirBlks>** – Sets up the number directory blocks used to allocate the target PDS if it does not exist. If this argument is not specified, the value in the configuration file is used. If neither is specified, the PDS allocation will fail. **Valid values:** 1-32760
- **<Dpath>** – Determines the fully qualified path and filename of the destination file. This argument is required unless the transaction sends the file to the staging queues.

- **<Dqm>** – Designates the destination queue manager. It is the queue manager on which the TDE Receiver component runs. This argument cannot be specified if the transaction sends the data to the staging queues. Otherwise, it is required when sending data to the OS/390 platform. The argument is optional with other operating systems.
- **<DType>** – Determines the data type for the destination data. You should only use this argument to handle destination data that is not stored in a file. This value must match a data type specified in the TDE configuration file and it is case sensitive.

You can also reference a datamap name in the DType argument of the FTF command. Specify which datamap you want to use when issuing the data-transfer request.

- **<Exit>** – Determines the exit number to be invoked. **Valid values:** 1-8, 9-10 (connectors)
- **<ExitData>** – Contains the command-line argument to execute when you invoke a user exit that requires input parameters.
- **<ExitDll>** – Determines the DLL, shared object, or load module used to invoke the exit module.
- **<ExitEntry>** – Contains the name of the function in the DLL that contains the exit module.
- **<Expiry>** – Determines the time period, measured in minutes, after which the data-transfer request expires. If the expiration duration is exceeded, the request is terminated and the FTFRCI\_REQUEST\_EXPIRED message is returned. If the expiration occurs partway through a request, the request is marked as expired.
- **<FileASP>** – Specifies the Library Auxiliary Pool for a file that TDE creates for a data-transfer request. **Valid values:** 1-16
- **<FileTxt>** – Specifies the file description for a library that TDE creates for a data-transfer request.
- **<FromStage>** – Indicates that the source file specified in the Spath argument should be retrieved from the staging queue rather than from the actual source file.
- **<FTFId>** – Indicates the FTFId of the transaction that should be sent from the staging queue. You should not use this option unless you are sending a file from the staging queue using the fromstage option, and you have not specified a source filename.



- **<ID1>** – Designates the first user-defined field that is associated with a data transfer. The ID1 field can contain any value that you desire. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **<ID2>** – Designates the second user-defined field that is associated with a data transfer. The ID2 field can contain any value that you desire. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **<ID3>** – Designates the third user-defined field that is associated with a data transfer. The ID3 field can contain any value that you desire. If the identifier text value contains spaces, the value needs to be in quotes. The value placed in the field is carried with the data transfer and the values are available to you in exit routines. Additionally, the FTFSTAT command displays the values of the identifier fields when you request a status display.
- **<Immed>** – Gives you the ability to issue a TDE data-transfer request that is processed synchronously between the Sender and Receiver rather than the normal asynchronous mode. Using this argument differentiates an immediate data transfer from a regular TDE transfer. Using FTF immediate, the TDE Receiver begins to process the data transfer request as soon as it gets the request instead of waiting for all of the data to arrive. Since the receiver processes the data messages as they arrive, the queue storage required to transfer a large amount of data is significantly reduced. Normal TDE data transfers require that the Receiver have all related data messages before writing the data to the disk. For example, using TDE in normal mode processing very large data transfers such as 1 gigabyte, the receiving node needs to have 1 gigabyte for queue storage and 1 gigabyte to store the data. With FTF immediate, the queue storage space is not required.
- **<Label>** – Specifies the user-defined label. This value allows you to assign arbitrary labels to data transfers to allow for status queries. Each label can be up to 20 bytes in length.
- **<LibASP>** – Specifies the Library Auxiliary Pool for a library that TDE creates for a data-transfer request. **Valid values:** 1-16

- **<LibTxt>** – Specifies the library description for a library that TDE creates for a data-transfer request.
- **<Lqm>** – Determines the queue manager from which the FTF command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTF command connects to the default queue manager that is set in the MQSeries configuration.

Otherwise, whenever a command or interface starts up it tries to connect to the local queue manager (Lqm). If no Lqm value is specified, the command or interface attempts to connect to the specified default queue manager on platforms where MQSeries supports them.

- **<LRecl>** – Determines the logical record length for the target file on OS/390. If the value for recfmt is V or VB, then the value for LRecl should be 4 bytes greater than the longest data record. **Valid values:** 1-32760

RecFmt Value	LRecl and BlkSize Relationship
F	LRecl must be equal to BlkSize
FB	If BlkSize does not equal 0 then BlkSize must be a multiple of LRecl
V	Blksize must be equal to LRecl+4. This will allow for the block descriptor word.
VB	If BlkSize does not equal 0, then LRecl must be no more than BlkSize–4. This will allow for the block descriptor word

- **<MkDirs>** – Creates the directories required to support the Dpath value. If this argument is not specified and the specified directory does not exist, the data-transfer request fails with an FILE OPEN ERROR.
- **<Mode>** – Determines what occurs when the data is written to the target. Although the default value for this argument is *create*, the only values you can specify from the command line are *append* and *noreplace*. Use this argument only when you want to override the default *create* setting. **Valid values:** append, noreplace. **Default value:** create
- **<Model>** – Indicates a model data set for Generation Data Group (GDG) allocation. Consult your OS/390 Systems Administrator for the available model data sets.

- **<MsgSize>** – Allows you to set a message size value to override the MQSeries message size value. **Valid values:** 1-3906 KB (3.9 MB)  
**Default value:** 512
- **<NotifyData>** – Specifies user-defined data to aid in notification, such as e-mail, pager, or fax information, that will be used to deliver a notification message based on a transaction's status. If you specify this argument, you must also specify NotifyStatus and NotifyType arguments.
- **<NotifyStatus>** – Defines when a notification message will be sent to the NotifyQueue (see *Tivoli Data Exchange Installation Guide*, "Tivoli Data Exchange Configuration," Notification Message Property, for more information). The notification is sent if the transaction's status matches the status specified in this argument. If you specify this argument, you must also specify NotifyData and NotifyType arguments. **Valid values:** Success, Failure, Nonsuccess (includes failed, cancelled, expired)
- **<NotifyType>** – Specifies the user-defined method, such as Email, Pager, Fax, or WTO, that will be used to deliver a notification message based on a transaction's status. If you specify this argument, you must also specify NotifyData and NotifyStatus arguments.
- **<OFile>** – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTF command. In the options file, you can set any of the command-line arguments that can be set for the FTF command. Any values specified on the command line override the values in the options file.
- **<Oqm>** – Determines the queue manager where the TDE Manager operates. If this value is not specified, the Oqm is given the same value as the Lqm.
- **<Org>** – Determines the file organization of the target file on OS/390. This argument is not required for a preallocated data set. **Valid values:** Physical Sequential (PS), Partitioned Data Set (PDS)
- **<PadChar>** – If padding of target file records is elected by entering the argument RecPad=pad, the padCharacter indicates what hexadecimal character is to be used for padding. If PadChar is not specified in the command and padding is elected, blank is assumed to be the padding character.

- **<Pool>** – Name of the data pool used for transferring data from the TDE Sender to the TDE Receiver. If this value is not specified, the default pool specified in the TDE configuration file is used. For this option to function, the specified pool must be defined in the TDE configuration file.
- **<Primary>** – Determines the number of primary allocation units required on OS/390.
- **<Priority>** – Determines the priority applied to the data-transfer request. **Valid values:** 1 (highest) – 5 (lowest) **Default value:** 5
- **<RcdLen>** – Determines the record length for the target file on OS/390. **Valid values:** 13-3276
- **<RecFmt>** – Determines the record format for the target file on OS/390. **Valid values:** F (fixed), V (variable), FB (fixed block), VB (variable block)
- **<RecPad>** – Enables or disables the padding facility. TDE provides the following options:
  - **nopad** – Specifies that blanks are not to be inserted in each record to fill it out to the length of the other records in the file.
  - **pad** – Specifies that blanks (by default) are to be inserted in each record to fill it out to the length of the other records in the file (*see also* PadChar)
- **<RecWrap>** – Indicates how records will be processed when they reach the target file and the records are longer than the target record length. TDE provides the following options:
  - **Wrap** - wraps records that are of greater length than the target file record length.
  - **Truncate** - truncates records up to the record length of the target file.
  - **Fail** - fails the data-transfer request when the record length exceeds the maximum allowed for the target file.
- **<ReplyQ>** – Names the queue to which reply messages are to be routed.
- **<ReplyQMgr>** – Names the queue manager to which reply messages are to be routed.

- **<RequestType>** – Specifies the type of transaction that is being submitted to the FTFMGR. **Valid values:** Request, Ping, Cancel
- **<SData>** – Determines the data input for a transfer that sends data that is not stored in a file.
- **<Secondary>** – Determines the number of secondary allocation units required on OS/390.
- **<Spath>** – Determines the fully qualified path and filename of the source file. This argument is required unless the file is being sent from the staging queue [using the fromstage option] and an FTFId is specified.
- **<Sqm>** – Determines the queue manager on the TDE Sender component. This argument is required in OS/390.
- **<Stage>** – Enables the data messages that make up the data being transferred to be stored in a staging area and remain there after the data-transfer transaction has ended.
- **<StageOnly>** – Places the data messages on the staging queue, but does not send it. If you specify this value, you cannot specify a destination queue manager or destination file value.
- **<StagePersist>** – Specifies that the messages in the staging area are persistent. If you specify this argument, the messages still exist after a system or TDE reboot or shutdown. If you select this argument, you increase the recovery ability of TDE but reduce performance.
- **<SType>** – Determines the data type for the source data. You should only use this argument to handle source data that is not stored in a file. This value must match a data type specified in the TDE configuration file, and it is case sensitive.
- **<TranPersist>** – Specifies that the data being transferred is persistent. If you specify this argument, the data still exists after a system or TDE reboot or shutdown. If you select this argument, you increase the recovery ability of TDE but reduce performance.
- **<Trusted>** – Sacrifices the ability to recover in order to allow for greater performance. In a trusted transaction, no file recovery is possible. Specifying this argument invokes the TDE trusted option, not the MQSeries trusted option.
- **<Type>** – Determines whether the file is text or binary. **Valid values:** text, binary. **Default value:** binary

- **<Unit>** – Determines the unit name for the target file on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.
- **<VolSer>** – Determines the volume serial number for the target on OS/390. This argument's value is installation-dependent. Obtain it from your OS/390 administrator.
- **<Wait>** – Contains the amount of time – in seconds – to wait for a reply. This argument indicates that the FTF command blocks for a response from the TDE Manager to indicate whether a request has succeeded or failed. If the TDE Manager does not respond within the specified time period, the command times out and is unblocked.

## **Ping Transaction Elements**

The list that follows contains the elements that can be used with a TDEe ping. The RequestType element value must be set to "Ping" for this type of transaction.

- **<CFile>** – Can contain the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no cq argument is specified, this value must be specified. You cannot specify both a cfile and a cq argument in the same command.
- **<CQ>** – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a CFile and a CQ argument in the same command.
- **<Dqm>** – Designates the destination queue manager. It is the queue manager on which the TDE Receiver component runs. This argument cannot be specified if the transaction sends the data to the staging queues. Otherwise, it is required when sending data to the OS/390 platform. The argument is optional with other operating systems.
- **<Lqm>** – Determines the queue manager from which the FTF command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTF command connects to the default queue manager that is set in the MQSeries configuration.

Otherwise, whenever a command or interface starts up it tries to connect to the local queue manager (Lqm). If no Lqm value is specified, the command or interface attempts to connect to the specified default queue manager on platforms where MQSeries supports them.

- **<MsgSize>** – Allows you to set a message size value to override the MQSeries message size value. **Valid values:** 1-3906 KB (3.9 MB)  
**Default value:** 512
- **<OFile>** – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTF command. In the options file, you can set any of the command-line arguments that can be set for the FTF command. Any values specified on the command line override the values in the options file.
- **<Oqm>** – Determines the queue manager where the query or purge request will be accepted into the TDE subsystem by the processing TDE Manager. If not specified, it defaults to the same value as Lqm.
- **<Priority>** – Determines the priority applied to the data-transfer request. **Valid values:** 1 (highest) – 5 (lowest). **Default value:** 5
- **<RequestType>** – Specifies the type of transaction that is being submitted to the FTFMGR. **Valid values:** Request, Ping, Cancel
- **<Sqm>** – Determines the queue manager on the TDE Sender component. This argument is required in OS/390.
- **<Timeout>** – Determines the amount of time until the FTFPING command times out. If the time limit is exceeded, the ping operation is terminated and an error message is generated. This value is represented in seconds. **Valid values:** 1-32767

## **Cancel Transaction Elements**

The list that follows contains the elements that can be used with a TDE cancel. The RequestType element value must be set to “Cancel” for this type of transaction.

- **<CFile>** – Contains the fully qualified path and filename for the TDE configuration file. On OS/390 platforms, if no CQ argument is specified, this value must be specified. You cannot specify both a CFile and a CQ argument in the same command.

- **<CQ>** – Displays the queue from which the configuration information is to be retrieved for this TDE instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points TDE to the queue name rather than to the standard configuration file. You cannot specify both a CFile and a CQ argument in the same command.
- **<FTFId>** – Indicates the FTFId of the transaction that should be sent from the staging queue. You should not use this option unless you are sending a file from the staging queue using the *fromstage* option, and you have not specified a source filename.
- **<Lqm>** – Determines the queue manager from which the FTF command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTF command connects to the default queue manager that is set in the MQSeries configuration.

Otherwise, whenever a command or interface starts up it tries to connect to the local queue manager (lqm). If no lqm value is specified, the command or interface attempts to connect to the specified default queue manager on platforms where MQSeries supports them.

- **<OFile>** – Contains the fully qualified path and filename of a text file used to contain command-line arguments for the FTF command. In the options file, you can set any of the command-line arguments that can be set for the FTF command. Any values specified on the command line override the values in the options file.
- **<Oqm>** – Determines the queue manager where the TDE Manager operates. If this value is not specified, the Oqm is given the same value as the Lqm.
- **<RequestType>** – Specifies the type of transaction that is being submitted to the FTFMGR. **Valid values:** Request, Ping, Cancel



# Examples

## Request

```
<?xml version='1.0'?>
<FTFInputData>
  <RequestType>Request</RequestType>
  <Oqm>value</Oqm>
  <Dqm>value</Dqm>
  <Trusted/>
  <Tranpersist/>
  <StagePersist/>
  <MsgSize>value</MsgSize>
  <FTFId>value</FTFId>
  <CancelMode>value</CancelMode>
  <DelSrc/>
  <ID1>value</ID1>
  <ID2>value</ID2>
  <CFile>value</CFile>
  <CQ>value</CQ>
  <OFile>value</OFile>
  <ReplyQ>value</ReplyQ>
  <NotifyType>value</NotifyType>
  <NotifyData>value</NotifyData>
  <Exit>value</Exit>
  <RecWrap>value</RecWrap>
  <RecPad>value</RecPad>
  <MkDirs/>
  <Mode>value</Mode>
  <SType>value</SType>
  <DType>value</DType>
  <DData>value</DData>
  <AS400FT>value</AS400FT>
  <LibAsp>value</LibAsp>
  <FileAsp>value</FileAsp>
  <LibTxt>value</LibTxt>
  <FileTxt>value</FileTxt>
  <RcdLen>value</RcdLen>
  <LRecl>value</LRecl>
  <AlcUnit>value</AlcUnit>
  <Secondary>value</Secondary>
  <BufNo>value</BufNo>
</FTFInputData>
```

## **Ping**

```
<?xml version='1.0'?>
<FTFInputData>
  <RequestType>Ping</RequestType>
  <Dqm>value</Dqm>
  <Sqm>value</Sqm>
  <Oqm>value</Oqm>
  <Timeout>60</Timeout>
  <CFile>D:\FTFCONFIG.INI</CFile>
</FTFInputData>
```

## **Cancel**

```
<?xml version='1.0'?>
<FTFInputData>
  <RequestType>Cancel</RequestType>
  <FTFId>value</FTFId>
  <Lqm>value</Lqm>
  <Oqm>value</Oqm>
  <CFile>value</CFile>
</FTFInputData>
```

# Sample DTD

## Request

```

<!ELEMENT FTFInputData
  (RequestType?,LQM?,Oqm?,Sqm?,Dqm?,SPath?,DPath?,Label?,Wait?,Trusted?,T
  ranpersist?,StagePersist?,MsgSize?,Priority?,Stage?,StageOnly?,FromSta
  ge?,FTFId?,Expiry?,CancelMode?,DelSrc?,Immed?,ID1?,ID2?,ID3?,CFile?,CQ
  ?,OFile?,ReplyQ?,ReplyQMgr?,NotifyStat?,NotifyType?,NotifyData?,Exit?,
  ExitDll?,ExitData?,Type?,RecWrap?,RecPad?,PadChar?,MkDirs?,Mode?,Pool?
  ,Compress?,SType?,DType?,SData?,DData?,AS400FT?,CRTLib?,LibAsp?,FileAs
  p?,LibTxt?,FileTxt?,CCSID?,RcdLen?,Org?,DirBlks?,RecFmt?,LRecl?,BlkSiz
  e?,Unit?,VolSer?,AlcUnit?,Primary?,Secondary?,Model?,BufNo?)>
<!ATTLIST FTFInputData
  Version CDATA #IMPLIED>
<!ELEMENT RequestType (#PCDATA)>
<!ELEMENT LQM (#PCDATA)>
<!ELEMENT Oqm (#PCDATA)>
<!ELEMENT Sqm (#PCDATA)>
<!ELEMENT Dqm (#PCDATA)>
<!ELEMENT SPath (#PCDATA)>
<!ELEMENT DPath (#PCDATA)>
<!ELEMENT Label (#PCDATA)>
<!ELEMENT Wait (#PCDATA)>
<!ELEMENT Trusted EMPTY>
<!ELEMENT Tranpersist EMPTY>
<!ELEMENT StagePersist EMPTY>
<!ELEMENT MsgSize (#PCDATA)>
<!ELEMENT Priority (#PCDATA)>
<!ELEMENT Stage EMPTY>
<!ELEMENT StageOnly EMPTY>
<!ELEMENT FromStage EMPTY>
<!ELEMENT FTFId (#PCDATA)>
<!ELEMENT Expiry (#PCDATA)>
<!ELEMENT CancelMode (#PCDATA)>
<!ELEMENT DelSrc EMPTY>
<!ELEMENT Immed EMPTY>
<!ELEMENT ID1 (#PCDATA)>
<!ELEMENT ID2 (#PCDATA)>
<!ELEMENT ID3 (#PCDATA)>
<!ELEMENT CFile (#PCDATA)>
<!ELEMENT CQ (#PCDATA)>
<!ELEMENT OFile (#PCDATA)>

```

```
<!ELEMENT ReplyQ (#PCDATA)>
<!ELEMENT ReplyQMgr (#PCDATA)>
<!ELEMENT NotifyStat (#PCDATA)>
<!ELEMENT NotifyType (#PCDATA)>
<!ELEMENT NotifyData (#PCDATA)>
<!ELEMENT Exit (#PCDATA)>
<!ELEMENT ExitDll (#PCDATA)>
<!ELEMENT ExitEntry (#PCDATA)>
<!ELEMENT ExitData (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT RecWrap (#PCDATA)>
<!ELEMENT RecPad (#PCDATA)>
<!ELEMENT PadChar (#PCDATA)>
<!ELEMENT MkDirs EMPTY>
<!ELEMENT Mode (#PCDATA)>
<!ELEMENT Pool (#PCDATA)>
<!ELEMENT Compress EMPTY>
<!ELEMENT SType (#PCDATA)>
<!ELEMENT DType (#PCDATA)>
<!ELEMENT SData (#PCDATA)>
<!ELEMENT DData (#PCDATA)>
<!ELEMENT AS400FT (#PCDATA)>
<!ELEMENT CRTLib EMPTY>
<!ELEMENT LibAsp (#PCDATA)>
<!ELEMENT FileAsp (#PCDATA)>
<!ELEMENT LibTxt (#PCDATA)>
<!ELEMENT FileTxt (#PCDATA)>
<!ELEMENT CCSId (#PCDATA)>
<!ELEMENT RcdLen (#PCDATA)>
<!ELEMENT Org (#PCDATA)>
<!ELEMENT DirBlks (#PCDATA)>
<!ELEMENT RecFmt (#PCDATA)>
<!ELEMENT LRecl (#PCDATA)>
<!ELEMENT BlkSize (#PCDATA)>
<!ELEMENT Unit (#PCDATA)>
<!ELEMENT VolSer (#PCDATA)>
<!ELEMENT AlcUnit (#PCDATA)>
<!ELEMENT Primary (#PCDATA)>
<!ELEMENT Secondary (#PCDATA)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT BufNo (#PCDATA)>
```

## **Ping**

```
<!ELEMENT RequestType (#PCDATA)>
<!ELEMENT Lqm (#PCDATA)>
<!ELEMENT Dqm (#PCDATA)>
<!ELEMENT Sqm (#PCDATA)>
<!ELEMENT Oqm (#PCDATA)>
<!ELEMENT FTFInputData
  (RequestType,Lqm?,Dqm?,Sqm?,Oqm?,Timeout?,MsgSize?,CFile?,CQ?,OFile?)>
<!ELEMENT Timeout (#PCDATA)>
<!ELEMENT MsgSize (#PCDATA)>
<!ELEMENT CFile (#PCDATA)>
<!ELEMENT CQ (#PCDATA)>
<!ELEMENT OFile (#PCDATA)>
```

## **Cancel**

```
<!ELEMENT FTFInputData (RequestType,FTFId,Lqm?,Oqm?,CFile?,CQ?,OFile?)>
<!ELEMENT RequestType (#PCDATA)>
<!ELEMENT Lqm (#PCDATA)>
<!ELEMENT Oqm (#PCDATA)>
<!ELEMENT FTFId (#PCDATA)>
<!ELEMENT CFile (#PCDATA)>
<!ELEMENT OFile (#PCDATA)>
<!ELEMENT CQ (#PCDATA)>
```



---

---

# Multi-File Connector

The Multi-File Connector allows you to extract data from and update rows in OS/390 file structures. Additionally, the connector has the ability to transfer many files to a single target location. The purpose of this document is to define and describe the added functionality needed to provide this capability.

This chapter contains the following sections:

Section	Page
Introduction	343
Multi-File Connector Installation	344
Connector Entry Points	345
Invoking the Multi-File Connector	346
XML Configuration File	352
Error Conditions	354

## Assumptions

For the Tivoli Data Exchange (TDE) Multi-File Connector to function, TDE 1.1.0 or higher and MQSeries must be installed.

## Introduction

The Multi-File Connector allows TDE to transfer OS/390 VSAM data sets, transfer an entire OS/390 partitioned data set (PDS) or an Open Systems directory as a single object, and merge multiple files into a single TDE data-transfer request transmitted under a single FTF ID. The source files must exist before the request is initiated and can have different logical record lengths. Having accomplished this with a single TDE data-transfer request, any exit

## **Multi-File Connector**

### *Multi-File Connector Installation*

processing (e.g., receiver postprocess exit) occurs only when the entire process has completed successfully. All files must be sent as binary with no data-to-text conversion and no code page conversion. All OS/390 dynamic file allocation happens within the connector (except for VSAM files, which must be preallocated).

The Multi-File Connector supports the following operating systems:

- OS/390
- Win 32
- Solaris
- AIX
- HP-UX

The Multi-File Connector can transfer multiple files as a single transfer unit in two ways:

1. If the source or target specifies an Open Systems directory or a PDS, each object within the directory or PDS will be handled individually. An OS/390 PDS is converted to a directory of files on an Open Systems receiver.
2. A directory of files with a PDS target will create individual members within the output data set. If the files are not grouped into directories, the MULTLOAD entry point uses an XML document pointed to the `—spath` option to parse a list of otherwise unrelated source and target file pairs. All files must have the same origination and target host.

Transfers using the Multi-File Connector generate TDE status information like any other transfer request. The connector processes at both the sending and receiving nodes. The host supplements FTFSTAT messages with specific details of the transfer. In instances where multiple files are transferred in a single request, the names of individual files appear within the detailed TDE status messages. A multi-file transfer may result from specifying several files within an XML configuration document or by transferring a directory or library containing multiple file objects.

## **Multi-File Connector Installation**

The Multi-File Connector is distributed and installed with the base TDE product.



# Connector Entry Points

## Overview

The Multi-File Connector is implemented using several entry points to a dynamic linked library and is invoked using the connector exit numbers 9 (Sending) and 10 (Receiving). The TDE request must specify both the connector library name using the `exitdll` argument and the entry point using `exitentry`. Each platform should accept `FTFCMF` as the exit library. This causes TDE to search the `SYSPROC` or library path for a file with this name.

There are five distinct entry points in the library.

Entry Point	Description
<b>DATAEX</b>	This entry point is invoked at the source (exit 9) to extract the local data into TDE messages.
<b>DATALOAD</b>	This entry pairs with <b>DATAEX</b> at the target host, and is invoked in exit 10. It takes the messages from TDE and translates them into data stores on the local file system.
<b>MULTIEX</b>	This is the source exit (exit 9) used when specifying a configuration file containing a file list. The name of the configuration file appears in the <code>spath</code> . This must be a valid text file with appropriate syntax.
<b>MULTLOAD</b>	The <b>MULTLOAD</b> entry point couples with the <b>MULTIEX</b> entry for the target host and is specified in exit 10.
<b>USAGE</b>	This entry point displays release, version, and build from the <code>portal.h</code> file. In the process of displaying the information, this entry point issues a pseudo-transfer which fails. This failure is recorded in the transfer logs.

## Invoking the Multi-File Connector

All data transfers are binary and they must contain the `-type BINARY` option. The Multi-File Connector does not support text format. Transfers complete successfully only in binary format.

This presents special problems with the record-oriented data model on OS/390.

---

### Note:

Because of the complex nature of VSAM allocations, the Multi-File Connector is designed to work only with existing data sets. The following JCL samples illustrate a simple use of IDCAMS to allocate basic VSAM clusters. More complex allocations are typical, however, and are usually strictly controlled in production environments.

---

## Sample Transfer from Win 32 to OS/390

The following example shows a transfer of a binary file from a Win 32 file on the local host to a target OS/390 data set. This specific transfer is taking a picture image and writing it as a single record in a sequential data set.

```
../${FTFVER}/bin/ftf
-sqm $DFLTQM \
-type BINARY \
-dqm MQA1 \
-spath C:/FTF/TEST/LANE1.JPG \
-dpath DEVSXK.FTFCMF.TEST.NT.JPEG \
-org PS \
-recfmt VB \
-lrecl 32760 \
-blksize 32760 \
-volser USER01 \
-unit SYSDA \
-exit 9 -exitdll FTFCMF -exitentry DATAEX \
-exit 10 -exitdll FTFCMF -exitentry DATALOAD
```

On Win 32, the forward slash (/) may be interchanged for the backslash (\) whenever necessary. In this case, the transfer is being invoked within a UNIX script, so the use of the forward slash avoids the special meaning of the backslash to the shell. Both exit 9 (sending connector) and exit 10 (receiving connector) are specified in this transfer.

---

## **Warnings:**

- Any transfer using this connector overlays an existing data store if it has the same name as the target. There is no support in this connector for the -mode noreplace option. Even if the transfer fails, a preexisting data store may be deleted.
- Be aware when transferring files from a platform that supports mixed-case file naming to a platform that does not support it. If you have two files named New.txt and NEW.TXT on the UNIX platform and you transfer them to the Win 32 platform, the last file transferred overwrites the first file transferred on the Win 32 platform because NT sees both files as having the same name.

---

## **Notes:**

- If the source file does not exist before the connector is initiated, the connector terminates with an error.
- If you change environment variable values, you must restart a running TDE Manager, Sender, and Receiver component to effect the changes.

---

## **Output**

Transfer information is logged to the TDE status system. These can be viewed using the -format detail of the FTFSTAT command line or through the TDE GUI. The specific messages vary depending on the type of transfer; most notably, multi-file transfers generate considerably more information than others.

## Multi-File Connector

### *Invoking the Multi-File Connector*

## VSAM Data

The example below illustrates how to transfer an existing OS/390 VSAM Keyed Sequenced Data Set (KSDS), "DEVSEXK.FTFCMF.KSDS" stored on DASD (-unit SYSDA) on an OS/390 system running a Queue Manager called MQA1 to a UNIX destination system running a Queue Manager called DOLPHIN. The name of the file on the destination system is /home/skanner/openPortal/frommvs. Exit 9 on the OS/390 system calls the DATAEX entry point in DLL FTFCMF to read the data set. Exit 10 on the UNIX system calls the DATALOAD entry point in DLL libopenprtl.so to write the file.

No record format information is preserved once the receiver connector, DATALOAD, is executed. To preserve VSAM data, the records should be extracted from VSAM using the sender connector, DATAEX, and stored as a simple Open Systems file. The receiving component should not invoke the connector for the initial transfer. A subsequent transfer may then use this stored file as input and transfer it to an OS/390 target specifying the receiving connector DATALOAD. This scenario should restore data into the original or similar VSAM cluster.

Use this same format to transfer an exiting Entry Sequenced Data Set (ESDS) to an Open Systems directory.

```
ftf -CFILE ~/openPortal/ftfconfig.ini
-LQM DOLPHIN
-SQM MQA1
-SPATH DEVSEXK.FTFCMF.KSDS -unit SYSDA
-DQM DOLPHIN
-DPATH /home/skanner/openPortal/frommvs
-EXIT 9
-EXITDLL FTFCMF -EXITENTRY DATAEX
-EXIT 10
-EXITDLL libopenprtl.so -EXITENTRY DATALOAD
```

The following format illustrates how to transfer an existing KSDS VSAM file from one OS/390 system to another OS/390 system.

EDIT     DEVSXK.FTFCMF.JCLLIB(PKSDSE) - 01.00

```
000001 //PKSDSE   JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
000002 //FTFREQ   EXEC PGM=FTF,REGION=0M,TIME=NOLIMIT,
000003 //          PARM='TRAP(OFF) /-OFILE DD:SYSIN'
000004 //*
000005 //STEPLIB   DD DSN=FTFV2.EA3101.FTF.LOADLIB,DISP=SHR
000006 //          DD DSN=SYS1.V114.SCSQAUTH,DISP=SHR
000007 //          DD DSN=SYS1.V114.SCSQANLE,DISP=SHR
000008 //SYSPRINT  DD SYSOUT=*
000009 //SYSERR    DD SYSOUT=*
000010 //SYSOUT    DD SYSOUT=*
000011 //SYSUDUMP  DD SYSOUT=*
000012 //FTFLOG00  DD SYSOUT=*
000013 //SYSIN     DD *
000014 -CFILE      DEVSXK.FTFCMF.TEST.FTF.INI
000015 -LQM        MQA1
000016 -SQM        MQA1
000017 -SPATH      DEVSXK.FTFCMF.KSDS
000018 -DQM        MQA1
000019 -DPATH      DEVSXK.FTFCMF.KSDS.DEST
000020 -EXIT       9
000021 -EXITDLL    FTFCMF
000022 -EXITENTRY  DATAEX
000023 -EXIT       10
000024 -EXITDLL    FTFCMF
000025 -EXITENTRY  DATALOAD
000026 /*
```

## Multi-File Connector

### *Invoking the Multi-File Connector*

The following sample JCL can be used to create a KSDS data set on OS/390.

```
000001 //IDCAMSK  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID,RESTART=*
000002 //STEP1    EXEC    PGM=IDCAMSK
000003 //*
000004 //SYSPRINT DD      SYSOUT=*
000005 //SYSIN     DD      *
000006         DELETE -
000007             (DEVSXK.FTFECMF.KSDS) PURGE
000008
000009         IF LASTCC < 12 -
000010         THEN -
000011         DEFINE CLUSTER -
000012             (NAME (DEVSXK.FTFECMF.KSDS) -
000013             VOLUMES (TECH01) -
000014             RECORDS (50 25) ) -
000015             DATA -
000016             (NAME (DEVSXK.FTFECMF.KSDS.DATA) -
000017             KEYS (24 0) -
000018             RECORDSIZE (3883 3883) -
000019             FREESPACE (20 10) -
000020             BUFFERSPACE (250) ) -
000021             INDEX -
000022             (NAME (DEVSXK.FTFECMF.KSDS.INDX) IMBED)
000023 //
```

The following JCL can be used to create an ESDS data set on OS/390.

```
000001 //IDCAMS JOB CLASS=A,MSGCLASS=O,NOTIFY=&SYSUID
000002 //*
000003 //STEP2 EXEC PGM=IDCAMS
000004 //SYSPRINT DD SYSOUT=A
000005 //SYSIN DD *
000006 DELETE -
000007 DEVSXK. FTFCMF.ESDS
000008 //STEP1 EXEC PGM=IDCAMS
000009 //*
000010 //SYSPRINT DD SYSOUT=A
000011 //SYSIN DD *
000012 DEFINE CLUSTER -
000013 (NAME (DEVSXK. FTFCMF.ESDS) -
000014 VOLUMES (TECH00) -
000015 RECORDS (100 100) -
000016 RECORDSIZE (80 120) -
000017 NONINDEXED -
000018 REUSE)
000019 /
```

## Directory / PDS Transfers

The example below illustrates how to transfer files from an Open Systems directory to an existing (the destination directory must exist) Open Systems directory. All of the files in the directory are transferred.

```
#ftf -CFILE ~/openPortal/ftfconfig.ini
-LQM DOLPHIN
-SQM DOLPHIN
-SPATH /home/skanner/openPortal/orig/src
-DQM DOLPHIN
-DPATH /home/skanner/openPortal/testDir
-EXIT 9 -EXITDLL ftfcmf -EXITENTRY DATAEX
-EXIT 10 -EXITDLL ftfcmf -EXITENTRY DATALOAD
```

## XML Configuration File

Files that are otherwise unrelated may be transferred together using the `spath` option to point to an XML configuration file. The `MULTIEX` and `MULTLOAD` entry points facilitate this functionality. This file has the following basic data type definition (DTD).

```
<!ELEMENT filelist (file)+>

<!ELEMENT file (dcb-info)?>
<!ATTLIST file
  filename CDATA #REQUIRED
  destfile CDATA #IMPLIED
  >

<!ELEMENT dcb-info EMPTY>
<!ATTLIST dcb-info
  devicetype CDATA #IMPLIED
  volser CDATA #IMPLIED
  unittype CDATA #IMPLIED
  unit CDATA #IMPLIED
  recfm CDATA #IMPLIED
  primaryspace CDATA #IMPLIED
  secondaryspace CDATA #IMPLIED
  lrecl CDATA #IMPLIED
  blocksize CDATA #IMPLIED
  >
```



The following sample script file transfers a text file and a directory of files from Win 32 to OS/390:

```
<?xml version="1.0" encoding="UTF-8"?>
<filelist>

  <!--Transfer a PS specifying some DCB information.-->
  <file filename="C:/FTF/TEST/small.txt">
    <dcbl-info secondaryspace="9" lrecl="3200"
      blksize="3200"
primaryspace="7"
      VOLSER="USER04" DEVICE="SYSDA"/>
  <destfile="FTF.FTFCMF.TEST.TEXT.NT.XMLMULTI.PS">
  </file>

  <!--Transfer a PDS with a new target name.-->
  <file filename="c:/ftf/test/pds">
    <destfile="FTF.FTFCMF.TEST.TEXT.XMLMULTI.PDS">
    <dcbl-info secondaryspace="9" lrecl="3200"
      blksize="3200"
primaryspace="7"
      DSORG="PO" VOLSER="USER04" DEVICE="SYSDA"/>
  </file>
</filelist>
```

The following sample script file invokes the transfer:

```
#!/bin/bash

# FTF transfer of a file and a PDS within the same host
# based on XML

ftf -sqm $DFLTQM \
  -type BINARY \
  -dqm MQA1 \
  -spath $PWD/mvsMulti.xml \
  -exit 9 -exitdll FTFCMF -exitentry MULTIEX \
  -exit 10 -exitdll FTFCMF -exitentry MULTLOAD
```

## Error Conditions

Error messages displayed when using the Multi-File Connector usually come from one of two sources:

1. IBM MQSeries - the underlying transport mechanism for TDE
2. TDE itself

Information on error conditions for MQSeries and TDE can be found in the documentation for those products. Here are the common error messages from the Multi-File Connector. The messages without comments are self-explanatory.

<b>Numeric Code</b>	<b>Error Message Text</b>	<b>Comment</b>
346	ReadRecordFTF: FTFPortalReceive error <i>errorCode1 errorCode2</i>	FTFPortalReceive() call failed.
1001	In dataload of ftfcmf, exit invoked with bad rc <i>returncodeNumber</i>	DATALOAD() must be invoked with pExit->rc == 0.
1029	Unable to open file <i>filename</i>	
1030	DATAEX(): error from getStatus on file <i>filename</i>	
1035	Unable to open file <i>filename</i>	
1050	FTFCMF::DATAEX():Unexpected return from PutPortalHeader(),error= <i>errorCode</i>	
1052	DATAEX():Unexpected return from CreateFileNameList(), error= <i>errorCode</i>	
1053	FTFCMF::DATAEX():Unexpected return from PutFileHeader(), error= <i>errorCode</i>	
1055	Unable to complete request, error= <i>errorCode</i>	If fileType==PDS then PutFileTrailer(). Else MVSToFTF() failed.
1060	Unexpected return from PutPortalHeader(), error= <i>errorCode</i>	Expected CONNECTOR__SUCCESS.
1065	Unable to complete request, error= <i>errorCode</i>	If fileType==PDS then PutFileTrailer(). Else MVSToFTF() failed
1066	Unexpected status from PutFileHeader(), error= <i>errorCode</i>	Expected CONNECTOR__SUCCESS

<b>Numeric Code</b>	<b>Error Message Text</b>	<b>Comment</b>
1075	Unexpected status from GetPortalHeader(), <i>error=errorCode</i>	Issued from Dataload entry point
1076	Unexpected return from FTFToMVS, <i>error=errorCode</i>	
1080	Unexpected status from GetPortalHeader(), <i>error=errorCode</i>	Issued from the Multiload entry point
1081	Unable to read a record from the local file, <i>fileName</i>	
1082	Unable to read a record from MVS	
1083	Unable to read a record from PS	
1085	Unexpected status from FTFToMVS(), <i>error=errorCode</i>	
1086	Unexpected status from GetFileHeader(), <i>error=errorCode</i>	Issued from the Multiload entry point
1175	Unexpected status from GetFileHeader(), <i>error=errorCode</i>	Issued from the Dataload entry point
1220	DATAEX(): code passed to exit entry ( <i>errorCode</i> )	DATAEX() must be invoked with pExit->rc == 0
1221	FTFToMVS(): Unexpected result from WriteRecord()	
1231	Unable to allocate memory	
1293	DATAEX(): failure returned from PutPortalHeader()	
1294	DATAEX(): unable to determine file type for ( <i>fileName</i> )	
1346	ReadRecordFTF: FTFTPportalReceive error ( <i>errorCode1</i> ) ( <i>errorCode2</i> )	
1349	GetFileHeader(): FTFTPportalReceive error ( <i>errorCode1</i> ) ( <i>errorCode2</i> )	Call to FTFTPportalReceive() in function GetFileHeader() failed
1449	GetPortalHeader(): FTFTPportalReceive error ( <i>errorCode1</i> ) ( <i>errorCode2</i> )	Call to FTFTPportalReceive() in function GetPortalHeader() failed
1500	Unable to perform FTFTPportalSend ( <i>errorCode</i> )	

## Multi-File Connector

### Error Conditions

Numeric Code	Error Message Text	Comment
1504	Unable to perform FTFPortalSend for RDW. rc1:( <i>errorCode</i> )	
1510	Tried to write ( <i>number</i> ) bytes, actually wrote ( <i>number</i> ) bytes	
1511	WriteRecordFTF(): called with short recordLength	
1514	Tried to write RDW for ( <i>number</i> ) bytes, actually wrote ( <i>number</i> ) bytes	
1530	PutPortalHeader(): Error from FTFPortalSend. rc1:( <i>number</i> )	
1531	PutPortalHeader(): Tried to write ( <i>number</i> ) bytes, actually wrote ( <i>number</i> ) bytes	
1533	PutFileHeader(): Error from FTFPortalSend. rc1:( <i>errorCode</i> )	
1534	PutFileHeader(): Tried to write ( <i>number</i> ) bytes, actually wrote ( <i>number</i> ) bytes	
2010	FTFCMF::FTFToMVS(): Unexpected status from ReadRecordFTF, error <i>errorCode</i>	
2012	Unable to open file ( <i>fileName</i> )	
2120	MULTIEX() called with exit ( <i>exitNumber</i> ) rather than (9)	
2131	DATAEX(): empty file list returned	Empty file list returned by CreateFileNameList()
3020	FTFCMF::GetFileHeader(): Read file header length <i>numberBytes</i> not same as expected <i>numberBytes</i>	
3021	PMVS00::GetFileHeader(): found invalid file header type: <i>headerType</i>	
3050	Line is too big	Line is too big for the length specified in the call to getLine()
3109	GetPortalHeader(): header read ( <i>headerType</i> ) expected ( <i>headerType</i> )	
3111	GetPortalHeader(): invalid portal header type ( <i>headerType</i> )	Expect type CONNECTOR_HDR_REC

<b>Numeric Code</b>	<b>Error Message Text</b>	<b>Comment</b>
4000	stat() failed for fileName. rc = <i>returnCode</i>	Stat() call for this filename failed
4002	malloc failed for <i>numberBytes</i> bytes\n	
4200	Error opening file for output ( <i>fileName</i> ) type ( <i>fileType</i> ): <i>fileName</i>	
4211	Unexpected status from PL_CreateDirectory for ( <i>fileName</i> )	Status returned by PL_CreateDirectory() was CONNECTOR__FAILED
4213	error opening File ( <i>fileName</i> ); <i>fileName</i>	An Open Systems file
4219	error opening File ( <i>fileName</i> ); <i>fileName</i>	General file open failues
4811	Failed to open file ( <i>fileName</i> ): <i>fileName</i>	An Open Systems file or a PDS member
4812	Failed to open file ( <i>fileName</i> ): <i>fileName</i>	Type of file unknown
4813	Unhandled file type for file ( <i>fileName</i> ) type: <i>fileType</i>	
4814	Translated unknown file type	
5000	Tried to read <i>numberBytes</i> bytes, only read <i>numberBytes</i> bytes	
6020	GetAttribute(): probable parsing error with ( <i>errorCode</i> )	Check to prevent continuous looping
6030	ParseFileList(): no valid filename tag found	No valid filename was found in the file list
6422	malloc failed for <i>numberBytes</i> bytes\n	
9000	ParseFileList(): Unable to open file ( <i>fileName</i> ), errno: <i>errorNumber</i> , msg: <i>messageNumber</i>	
9243	ParseFileList(): unable to determine file type for ( <i>fileType</i> )	



---

---

# The File-to-Message Connector

The File-to-Message connector provides the ability for Tivoli Data Exchange to send text to and from MQSeries queues. There are two types of processing: DataExtract and DataLoad.

This chapter contains the following sections:

Section	Page
Introduction	396
File-to-Message Connector Installation	396
Arguments	397
File-to-Message Modifiers	399
Delimiter Processing	403
Record Processing	405
Logging and Status	406
Examples	407
Error Messages	413

## Assumptions

For the File-to-Message connector to function, TDE1.0.0 or higher and MQSeries must be installed.

## Introduction

The File-to Message connector allows you to send text to and from MQSeries queues using two types of processing: DataExtract and DataLoad. This connector is invoked through standard TDE (FTF) exits.

The File-to-Message connector supports the following operating systems:

- OS/390
- Win 32
- Solaris
- AIX
- HP-UX

## DataExtract Processing

The DataExtract process allows you to specify a source queue. Text messages are read from the source queue in the FTF Sender and sent through FTF to the FTF Receiver. The FTF Receiver processes the data as if it came from a standard source file. If no connectors or exits are specified in the FTF Receiver, then a file is written at the target node, by the Receiver, containing the text found in the source queue by the FTF Sender.

## DataLoad Processing

The DataLoad process provides the ability for the FTF Receiver to write the text it receives from the FTF Sender to an MQSeries queue. For example, text from a standard text file could be sent to an MQSeries queue at the target node. A user-written MQSeries application could then read the text from the queue using MQGet calls.

## File-to-Message Connector Installation

The File-to-Message connector is distributed and installed with the base FTF product.

See the “User Exits” chapter in the *Tivoli Data Exchange User's Guide* for details on where exits are located and how to configure the environment.



## Arguments

The File-to-Message connector is invoked using the standard FTF conventions. You can enter requests using the command line or one of the other interfaces. In either case, you need to consider the values that are described below.

In general, FTF commands using the File-to-Message connector have the following basic structure:

```
ftf    -lqm      QM1
        -sqm      QM1
        -dqm      QM1
        -spath    QUEUE_NAME
        -dpath    C:\TEMP\TEST.TXT
        -stype    MESSAGE
        -sdata    modifiers (see below)
```

—or—

```
ftf    -lqm      QM1
        -sqm      QM1
        -dqm      QM1
        -dpath    QUEUE_NAME
        -spath    C:\TEMP\TEST.TXT
        -dtype    MESSAGE
        -ddata    modifiers (see below)
```

### **-stype**

The source type, -stype, tells FTF what type of text is being transferred to the FTF Sender. For the File-to-Message connector, always set this value to MESSAGE. When the source data resides on an MQSeries queue, this value is required. The stype argument is used in conjunction with -sdata and -spath.

### **-sdata**

The source data, -sdata, specifies the File-to-Message modifiers (see below). It is recommended that you use quotes around this argument, due to the handling of special characters. The sdata argument is used in conjunction with -stype and -spath.

#### **-spath**

The source path, -spath, specifies the source file in a standard FTF transfer. If you are using the File-to-Message connector for DataExtract, the spath argument is used to specify the source queue. The source queue contains the source data, which will be sent to the FTF Receiver.

---

#### **Note:**

The queue must reside on the same queue manager that the FTF Sender is connected to.

---

#### **-dtype**

The destination type, -dtype, tells FTF what type of text is being processed by the FTF Receiver. For the File-to-Message connector, always set this value to MESSAGE. When the target text is destined for an MQSeries queue, this value is required. The dtype argument is used in conjunction with -ddata and -dpath.

#### **-ddata**

The destination data, -ddata, specifies the File-to-Message modifiers (see below). It is recommended that you use quotes around this argument, due to the handling of special characters. The ddata argument is used in conjunction with -dtype and -dpath.

#### **-dpath**

The destination path, -dpath, specifies the destination file in a standard FTF transfer. If you are using the File-to-Message connector for DataLoad, the dpath argument is used to specify the destination queue. The FTF Receiver puts all data messages on the destination queue.

---

#### **Note:**

The queue must reside on the same queue manager that the FTF Receiver is connected to.

---

## **-delsrc**

The `delsrc` argument is normally used to tell FTF to delete the source file when the transfer has completed. When you use the `delsrc` argument with `DataExtract`, the FTF Sender removes messages from the source queue after the transfer has completed. This is the only way FTF will remove messages from the source queue.

---

### **Note:**

Review the `MSGID` option in the `exitdata` modifiers section below. If the `MSGID` option is not specified, then all messages existing on the source queue will be removed. If the `MSGID` is specified, then only messages matching the Message ID will be removed. Please keep this in mind when designing systems.

---

## **File-to-Message Modifiers**

The `sdata` and `ddata` arguments can be populated with modifiers, which change the behavior of the connector. When you specify modifiers, separate them by commas with no spaces. It is recommended that you use quotes around this argument, due to the handling of special characters.

## **DataExtract Modifiers**

### **BATCH=xxx**

This modifier specifies the number of messages read from the `input_message_queue` before an `MQCommit` is performed. The default value is 100. You can change this value for performance tuning.

### **LOGFILE**

This modifier specifies the log file to be used by the connector. You must specify the fully qualified path and filename. If the file does not exist, then it is created at runtime. Normally, you should specify this modifier so you can monitor log messages.

## The File-to-Message Connector

### *File-to-Message Modifiers*

#### **MSGID=msgid**

This modifier specifies all 24 characters of the Message ID of the messages that are to be read from the `input_message_queue` specified by the `spath` argument. If you do not specify this argument, then all messages in the queue will be processed.

Note that the Message ID field is padded with NULL values.

#### **SINGLE**

This modifier specifies that only the first message is to be read from the `input_message_queue` for this transfer. If there are multiple messages on the `input_message_queue` and you do not specify **SINGLE**, then all messages will be read. If you specify **SINGLE**, then only the first message will be read.

#### **WAIT**

This modifier specifies that FTF should wait for messages to arrive on the `input_message_queue`, if no messages already exist. Use this option in conjunction with the **WAITTIME** option.

---

#### **Note:**

No other transfers can be processed by the FTF Sender component involved in this transfer while the File-to-Message connector is waiting.

---

#### **WAITTIME=xxx**

This modifier specifies the number of seconds to wait for messages to arrive on the `input_message_queue`. Use this option in conjunction with the **WAIT** option.

## DataLoad Modifiers

### **BATCH=xxx**

This modifier specifies the number of messages that are put to the output\_message\_queue before an MQCommit is performed. The default value is 100. You can change this value for performance tuning.

### **LOGFILE**

This modifier specifies the log file to be used by the connector. You must specify the fully qualified path and filename. If the file does not exist, then it is created at runtime. Normally, you should specify this modifier so you can monitor log messages.

### **CR – CRLF – LF – NOCR**

This modifier specifies the line terminators used to format the data stream. MQ messages are put on the stream separated by the hexadecimal value for one of the following line terminators:

- **CR** – carriage return
- **LF** – line feed
- **CRLF** – carriage return and line feed
- **NOCR** – no carriage return

**Default value:** LF

### **DELDLM**

This modifier specifies that the delimiters described in the STARTDLM and ENDDLM parameters are to be deleted from the data when it is stored in the target queue.

For example, if you specify STARTDLM=!!! and ENDDLM=@@@ and the DELDLM modifier, the source file contains the following data:

!!!This is line number one.@@@

The resulting message placed on the output\_message\_queue is:

*Message 1*

This is line number one.

## **ENDDL=xxx**

This modifier specifies the ending delimiter for the data being transferred. Data is copied from and including the STARTDLM, up to and including the ENDDL delimiter. You must specify this modifier with the STARTDLM. The STARTDLM and ENDDL values must be different. The comma and backslash are not valid delimiters.

The delimiters can be from one to many characters in length; they cannot, however, be broken across lines of data. The data can span any number of lines.

You cannot use delimiters with FIXED or SINGLE modifiers.

## **FIXED**

This modifier specifies that the connector should break up the input file stream based on the value in the RECSIZE modifier.

For example, if a file contains ten 100-byte records and the RECSIZE is set to 100, then 10 messages are placed on the output\_message\_queue. Each message contains 10 bytes of data.

## **NOT\_PERSISTENT**

This modifier specifies that messages put into the output\_message\_queue are not persistent, which means that they do not survive a system reboot or FTF shutdown. **Default value:** persistent

## **PERSISTENT**

This modifier specifies that messages put to the output\_message\_queue are persistent, which means that they do survive a system reboot or FTF shutdown. **Default value:** persistent

### **RECSIZE=x**

If you are operating in FIXED mode, this modifier specifies the record length. Any data in an input stream that occurs after this value is truncated. If you are operating in SINGLE mode, the RECSIZE value is ignored and the entire input file stream is processed.

### **SINGLE**

This modifier places the entire input file stream in a single message (not valid in FIXED mode).

### **STARTDLM=xxx**

This modifier specifies the starting delimiter for the data being transferred. Data is copied from and including the STARTDLM, up to and including the ENDDLM delimiter. You must specify this modifier with the ENDDLM. The STARTDLM and ENDDLM values must be different. The comma and backslash are not valid delimiters.

The delimiters can be from one to many characters in length; they cannot, however, be broken across lines of data. The data can span any number of lines.

You cannot use delimiters with FIXED or SINGLE modifiers.

### **SYNCQ=sync\_Queue**

This modifier specifies the name of the sync queue. The queue manager is assumed to be the same as the queue manager of the output\_message\_queue. The sync queue is used internally for recovery purposes. **Default value:** FTF.CONNECTOR.SYNC

## **Delimiter Processing**

This section describes delimiter processing and the DataLoad modifiers used to control it. The modifiers are: DELDLM, STARTDLM, and ENDDLM. Delimiters cannot be used with the FIXED or SINGLE modifiers. You must specify both STARTDLM and ENDDLM modifiers.

## The File-to-Message Connector

### *Delimiter Processing*

Delimiter processing allows you to control how source data is broken up into messages destined for the `output_message_queue`. For example, consider the following text file:

```
This is line number one.  
This is line number two.  
This is the last line.
```

If you transfer this file without delimiters or other modifiers, all three lines are placed into a single message on the `output_message_queue`.

If you specify start and end delimiters, you can separate out blocks of data to be placed in individual target messages. For example, if you modify the sample file as follows:

```
!!!This is line number one.@@@  
!!!This is line number two.@@@  
!!!This is the last line.@@@
```

Set the start delimiter modifier to “!!!” (`STARTDLM=!!!`) and the end delimiter modifier to “@@@” (`ENDDLM=@@@`) and three messages are placed on the `output_message_queue`:

```
Message 1  
!!!This is line number one.@@@  
Message 2  
!!!This is line number two.@@@  
Message 3  
!!!This is the last line.@@@
```

If you want to remove the start and end delimiters from the source data that is placed on the `output_message_queue`, you can specify the `DELDLM` modifier. The `DELDLM` modifier tells FTF to remove the delimiters before the messages are put on the `output_message_queue`.

If you specify `DELDLM` for the above example, the message on the `output_message_queue` is as follows:

```
Message 1  
This is line number one.  
Message 2  
This is line number two.  
Message 3  
This is the last line.
```

Data can span multiple lines. For example, if the source data looked as follows:



```
!!!This is line  
number one.@@@  
!!!This is line  
number two.@@@  
!!!This is the last line.@@@
```

The resulting messages placed on the output\_message\_queue are the same as for the previous example where each message's data was on a single line:

```
Message 1  
This is line number one.  
Message 2  
This is line number two.  
Message 3  
This is the last line.
```

Multiple blocks of data can reside on the same line. For example, if the source data looked as follows:

```
!!!ONE@@@!!!TWO@@@  
!!!THREE@@@
```

The resulting messages placed on the output\_message\_queue are as follows:

```
Message 1  
ONE  
Message 2  
TWO  
Message 3  
THREE
```

## Record Processing

Record processing gives you the ability to specify the maximum length of the input data. Use RECSIZE in conjunction with FIXED to specify that the source data should be treated as fixed-length records. Any data found after the RECSIZE is then truncated. For example, if you send the following file with FIXED and RECSIZE=10:

```
This is line number one.  
This is line number two.  
This is the last line.
```

The results are:

## The File-to-Message Connector

### *Logging and Status*

This is li  
This is li  
This is th

---

#### Note:

- If SINGLE has been specified, then the RECSIZE option is ignored.
  - You cannot specify STARTDLM and ENDDLM options when using the FIXED option.
- 

If you specify RECSIZE without FIXED, then this controls the internal buffer sizes used for IO. You can use this value for performance tuning.

## Logging and Status

The File-to-Message connector logs all activity and errors to the log file you specify in the LOGFILE parameter. In addition, status messages are entered into the status system.

---

#### Note:

- You need to maintain the log files.
  - See “Error Messages” on page -413 for information on the File-to-Message connector error conditions. For MQSeries errors, see the IBM documentation.
- 

Each log entry contains a date/timestamp followed by the associated FTFID. Warnings and information messages contain LOG(number), where the number corresponds to an information message. Error messages contain ERROR(number), where the number corresponds to an error number. Both entries end with a text message. The following sample log file illustrates typical log entries:

```
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) DataLoad PROCESSING STARTED
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) Using queue (F2M_QUEUE)
```

```
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) Using connector sync queue
(FTF.CONNECTOR.SYNC)
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) StartDlm = #####, EndDlm = %%
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) 6 Messages Written to Queue
2001/06/21 09:50:13 2bb9dda1-6583-11d5-9ef1-bb961abac4cd LOG(5200) Finished processing RC(0)
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef LOG(5200) DataLoad PROCESSING STARTED
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef LOG(5200) Using queue (F2M_QUEUE)
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef LOG(5200) Using connector sync queue
(FTF.CONNECTOR.SYNC)
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef ERROR(2053) MQPUT ERROR putting line message
to queue: MQ Reason = 2053
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef LOG(5200) 2 Messages Written to Queue
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef ERROR(2053) Unable to complete request, error=2053
2001/06/21 14:06:50 053ff8c1-65a7-11d5-9af8-e9e719ee29ef ERROR(2053) PROCESSING COMPLETE RC(2053)
```

You can query the FTF status system to obtain more insight into the status of a transfer.

## Examples

All of the following examples specify the LOGFILE option to enable logging. Some examples require that there be messages on a queue.

The source queue is a local queue named F2M\_QUEUE\_SOURCE.

The destination queue is a local queue named F2M\_QUEUE\_TARGET.

## Sending a Single Message to a File

In this example, you send text from a single message on a queue to a file. Note that you must place the message on the queue using an application. The transfer takes one message from the queue, regardless of the number of messages on the queue, so the SINGLE option is used. The following command line is used:

```
ftf -lqm QM1
    -sqm QM1
    -dqm QM1
    -spath F2M_QUEUE_SOURCE
    -dpath C:\TEMP\TEST.FILE
    -type TEXT
    -stype MESSAGE
    -sdata "LOGFILE=C:\TEMP\F2M.LOG, SINGLE"
```

The result of this transfer is that the target file contains data from the first message read from the queue.

---

**Note:**

In this example, the messages that were read from the queue still remain on the queue.

---

## **Sending Multiple Messages to a File**

In this example, you send text from a single message on a queue to a file. Note that you must place multiple messages on the queue using an application. The transfer takes all messages from the queue, regardless of the number of messages there, so the **SINGLE** option is *not* used. The following command line is used:

```
ftf -lqm QM1
    -sqm QM1
    -dqm QM1
    -spath F2M_QUEUE_SOURCE
    -dpath C:\TEMP\TEST.FILE
    -type TEXT
    -stype MESSAGE
    -sdata "LOGFILE=C:\TEMP\F2M.LOG"
```

The result of this transfer is that the target file contains text from all of the messages read from the queue.

---

**Note:**

In this example, the messages that were read from the queue still remain on the queue.

---

## **Sending a Specific Message to a File**

In this example, you send text from one particular message on a queue to a file. You know the message IDs of the messages on the queue. Note that you must place messages on the queue using an application. The transfer takes all messages from the queue that match an ID that you specify, regardless of the number of messages on the queue. The **SINGLE** option is *not* used. The message ID used is "TEST." The following command line is used:

```
ftf -lqm QM1
    -sqm QM1
    -dqm QM1
    -spath F2M_QUEUE_SOURCE
    -dpath C:\TEMP\TEST.FILE
    -type TEXT
    -stype MESSAGE
    -sdata "LOGFILE=C:\TEMP\F2M.LOG,MSGID=TEST"
```

The result of this transfer is that the target file contains text from all of the messages read from the queue with the message ID of “TEST.”

If you want to change this example to read only the first message with a matching message ID, add the **SINGLE** option to the **sdata** argument.

---

### **Note:**

In this example, the messages that were read from the queue still remain on the queue.

---

## **Sending a Message to a File Using the -DELSRC Option**

In this example, you send data from a single message on a queue to a file. After the transfer is completed, all messages that were read are removed from the queue. You know the message ID of the messages on the queue. Note that you must place messages on the queue using an application. The transfer takes all messages from the queue that match a specific message ID, regardless of the number of messages on the queue. The **SINGLE** option is *not* used. The message ID is “TEST.” The following command line is used:

```
ftf -lqm QM1
    -sqm QM1
    -dqm QM1
    -spath F2M_QUEUE_SOURCE
    -dpath C:\TEMP\TEST.FILE
    -type TEXT
    -delsrc
    -stype MESSAGE
    -sdata "LOGFILE=C:\TEMP\F2M.LOG,MSGID=TEST"
```

The result of this transfer is that the target file contains text from all of the messages read from the queue with the message ID of “TEST.” After all messages with MSGID=TEST are transferred, the messages that were read are removed from the queue.

## Waiting for a Single Message and Transferring to a File

In this example, you send text from a single message on a queue to a file. The transfer takes one message from the queue, regardless of the number of messages on the queue, so the SINGLE option is used. Also, you specify WAIT and WAITTIME options. The following command line is used:

```
ftf    -lqm          QM1
        -sqm          QM1
        -dqm          QM1
        -spath        F2M_QUEUE_SOURCE
        -dpath        C:\TEMP\TEST.FILE
        -type          TEXT
        -stype        MESSAGE
        -sdata         "LOGFILE=C:\TEMP\F2M.LOG, SINGLE,
                        WAIT, WAITTIME=60"
```

The result of this transfer is that the FTF Sender waits for up to 60 seconds for a message to arrive on the queue. Once a message is received, the text is sent as usual. Note that you must place a message on the queue during the 60-second time period. The target file contains data from the first message read from the queue.

---

### Note:

In this example, the messages that were read from the queue still remain on the queue.

---

## Sending a File to a Single Message

In this example, you send text from a standard text file to an output queue. All of the text is placed into a single message. The following command line is used:

```
ftf    -lqm          QM1
        -sqm          QM1
        -dqm          QM1
```

```
-spath      C:\TEMP\TEST.FILE
-dpath      F2M_QUEUE_SOURCE
-type       TEXT
-dtype      MESSAGE
-ddata      "LOGFILE=C:\TEMP\F2M.LOG,SINGLE"
```

The result of this transfer is that all text from the source file is placed into a single message on the F2M\_QUEUE\_TARGET.

## **Sending a File Using and Keeping Delimiters**

In this example, you send text from a standard text file to an output queue. Text is placed into messages based on the delimiters in the file. The delimiters are kept. The following command line is used:

```
ftf      -lqm      QM1
          -sqm      QM1
          -dqm      QM1
          -spath    C:\TEMP\TEST.FILE
          -dpath    F2M_QUEUE_SOURCE
          -type     TEXT
          -dtype    MESSAGE
          -ddata    "LOGFILE=C:\TEMP\F2M.LOG,
STARTDLM=!!!,ENDDLM=@@@"
```

The following is the sample source file:

```
!!!Item 1@@@!!!Item 2@@@!!!This is
data that spans multiple lines.@@@
```

The result of this transfer is that the delimited data from the source file is placed into messages on the F2M\_QUEUE\_TARGET.

```
Message 1
!!!Item 1@@@
Message 2
!!!Item 2@@@!!!
Message 3
!!!This is data that spans multiple lines.@@@
```

## **Sending a File Using and Removing Delimiters**

In this example, you send text from a standard text file to an output queue. Text is placed into messages based on the delimiters in the file. The delimiters are removed from the resulting data messages. The following command line is used:

```
ftf      -lqm           QM1
          -sqm           QM1
          -dqm           QM1
          -spath         C:\TEMP\TEST.FILE
          -dpath         F2M_QUEUE_SOURCE
          -type           TEXT
          -dtype         MESSAGE
          -ddata          "LOGFILE=C:\TEMP\F2M.LOG,
                          STARTDLM=!!!, ENDDLM=@@@, DELDLM"
```

The following is the sample source file:

```
!!!Item 1@@@!!!Item 2@@@!!!This is
data that spans multiple lines.@@@
```

The result of this transfer is that the delimited text from the source file is placed into messages on the F2M\_QUEUE\_TARGET.

*Message 1*

Item 1

*Message 2*

Item 2

*Message 3*

This is data that spans multiple lines.



## Error Messages

The following error and message codes are defined for the File-to-Message connector. Any other error messages are from MQSeries or FTF. See *Tivoli Data Exchange Messages and Codes* for information on FTF error conditions. For MQSeries errors, see the IBM documentation.

<b>Numeric Code</b>	<b>Error Message Text</b>	<b>Comment/Corrective Action</b>
5200	FTFCF2M_INFORMATION	An informative message was issued. Check the log file for more details.
5201	FTFCF2M_CONFIG_FILE_ERROR	Internal use only.
5202	FTFCF2M_NO_QMGR_INFO	Internal use only.
5204	FTFCF2M_MEMORY_ALLOCATION_ERROR	A request for memory failed. Examine system resources and free up more memory.
5205	FTFCF2M_INVALID_EXITDATA_VALUES	While processing user-supplied modifiers, an error was found. Check the DataExtract and DataLoad modifiers.
5206	FTFCF2M_OPEN_QUEUE_ERROR	An error occurred while trying to open a queue. Ensure that the queue name was specified correctly and that the queue exists. Check that the queue can be accessed for the type of processing required.
5207	FTFCF2M_PORTAL_SEND_ERROR	An error occurred while sending data to the FTF Sender component. Contact CommerceQuest Technical Support.
5208	FTFCF2M_MQ_ERROR	An MQSeries error was encountered. Check the log file for more details.
5209	FTFCF2M_MQ_COMMIT_ERROR	An error occurred while trying to perform an MQCommit. Check the log file for more details.

## The File-to-Message Connector

### Error Messages

<b>Numeric Code</b>	<b>Error Message Text</b>	<b>Comment/Corrective Action</b>
5210	FTFCF2M_CONFIG_ARG_ERROR	While processing user-supplied modifiers, an error was found. Check the DataExtract and DataLoad modifiers.
5211	FTFCF2M_NO_MESSAGE_FOUND	While performing an MQGet, no message was found. Ensure that the MSGID was specified correctly and that there are messages on the source queue.
5212	FTFCF2M_NO_RECORD_SIZE	No RECSIZE was specified, but was required.
5213	FTFCF2M_PORTAL_RECEIVE_ERROR	An error occurred while receiving data from the FTF Sender component. Contact CommerceQuest Technical Support.
5214	FTFCF2M_QMGR_NAME_ERROR	Internal use only.
5215	FTFCF2M_QUEUE_NAME_ERROR	The specified MQSeries queue name was incorrect. Ensure that the specified queue name is valid.
5216	FTFCF2M_OPEN_FILE_ERROR	An error occurred while trying to open a file. See the log file for more details.
5217	FTFCF2M_CLOSE_QUEUE_ERROR	An MQSeries error occurred while trying to close a queue. See the log file for more details.
5218	FTFCF2M_CONFIG_LINE_TOO_BIG	Internal use only.

# FTFBPI Wrapper

The FTFBPI chapter provides details of Tivoli Data Exchange interfacing with enableNet's Business Process Integrator (BPI) process.

It contains the following sections:

Section	Page
FTFBPI Wrapper	416

## Assumptions

This chapter makes the following assumptions:

- You have a working knowledge of Tivoli Data Exchange.
- You understand the concept and working of FTF, FTFPING, and FTSTAT requests.

## FTFBPI Wrapper

FTFBPI is a connection for CommerceQuest's enableNet Business Process Integrator (BPI) platform to interface into Tivoli Data Exchange as part of a business process executed under the control of BPI's business process coordinator.

FTFBPI is a DLL which can be loaded from a BPI script. Once the DLL is loaded, calls can be made to Tivoli Data Exchange to issue FTF, FTFPING, and FTFSTAT requests. The functions defined in the FTFBPI DLL are FTFRequestXMS, FTFPingXMS, and FTFStatXMS. Each of these functions requires two arguments.

### Argument #1

- Defines how to manage the MQSeries handle. If no handle is currently in use, set this argument to a non-zero value. FTFBPI obtains a handle to the queue manager and then releases it before returning control back to the script. If the queue manager handle is available within the script then set Argument #1 to 0 (zero). FTFBPI does not disconnect the handle when finished.

### Argument #2

- Is the name of the object that contains the FTF arguments.

For example, the following lines create an FTF request object to define and submit a simple transfer request.

---

```
var Request+XMOBJECT::create("FTFReqObject");
Request->cfile = "d:\\Tivoli Data
      Exchange\\bin\\ftfconfig.ini";
Request->lqm = "TPADEV002";
Request->oqm = "TPADEV002";
Request->sqm = "TPADEV002";
Request->dqm = "TPADEV002";
Request->wait = 10;
Request->spath = "d:\\spathe\\smltxt";
Request->dpath = "d:\\dpathe\\smalltxt";
```

---

All regular FTF command-line options (refer to the FTF -help option for the list of command-line options) can be specified in the request object by using the following convention.

OBJECTNAME->FtfOption -- where OBJECTNAME is the name of the BPI XMOBJect. FtfOption is the FTF argument name with the same case and argument name as an FTF command-line argument. -cfile is cfile, -spath is spathe, etc.

For more details on writing BPI scripts to utilize FTFBPI calls, examine the samples included here.

## **FTF.XMS Sample**

```
=====
FTF.XMS Sample
=====
/*****
**
** Source      : FTF.XMS
** Date       : 09/25/2001
** Description: This is a sample script, which shows how to submit an
**              FTF and FTFStat request using BPI.
** Required Changes:
**      Change all references to the following as appropriate:
**      ++CFILE++   FTF configuration file to use
**      ++LQM++     Queue Manager to connect to
**      ++DQM++     Destination Queue Manager to use
**      ++SQM++     Source Queue Manager to use
**      ++OQM++     Originating Queue Manager to use
**      ++WAIT++    Wait time in seconds (-wait)
**      ++SPATH++   Source filename and path
**      ++DPATH++   Destination filename and path
*****/
main ()
{
    //////////////////////////////////////
    // Load DLL and import the functions.
    //////////////////////////////////////
    var mh = loadModule("FTFBPI.dll"); //DLL Location
    import long FTFRequestXMS(long,xmoptr) from mh;
    import long FTFPingXMS(long,xmoptr) from mh;
    import long FTFStatXMS(long,xmoptr) from mh;
```

```

////////////////////////////////////
// Define FTFReqObject: You can basically use all of the command
// line options, by adding the fields of the same name to the
// FTFReqObject. For example, -cfile becomes FTFReqObject.cfile
////////////////////////////////////

var Request=XObject::create("FTFReqObject");
Request->cfile = "++CFILE++";
Request->lqm = "++LQM++";
Request->oqm = "++OQM++";
Request->sqm = "++SQM++";
Request->dqm = "++DQM++";
Request->wait = ++WAIT++;
Request->spath = "++SPATH++";
Request->dpath = "++DPATH++";
Request->idl = "test ID";

////////////////////////////////////
// Add an Exit:
// In order to add an exit to the request object, define an
// FTFReqExits object and then copy it into the FTFReqObject.
////////////////////////////////////

var Exit=XObject::create("FTFReqExits");
Exit->exit =5;
Exit->exitdll ="FTFEX56";
Exit->exitentry ="FTFSdrPre5";
Exit->exitdata ="notepad";
Request->insert(Exit->deepCopy());

Exit->empty();
Exit->exit =6;
Exit->exitdll ="FTFEX56";
Exit->exitentry ="FTFSdrPost6";
Exit->exitdata ="calc";
Request->insert(Exit->deepCopy());

////////////////////////////////////
// Call the FTFRequestXMS function to submit the FTF transfer.
// The function will insert an FTFReturnObject into the
// FTFReqObject. This can be used to tell if the request was
// submitted correctly. This is an example of the return object:
// <FTFReturnObject>
// <RC1>634</RC1>
// <RC2>0</RC2>
// <ErrorMsg>c:\spath\smltxt</ErrorMsg>
// <ftfid>d13533d1-ef85-11d3-b36d-bcf89f5a8e56</ftfid>
// </FTFReturnObject>
////////////////////////////////////

```

```

FTFRequestXMS(0,Request);
print(Request->asString());
if(Request->FTFReturnObject->RC1!=0)
{
    return(1);
}

////////////////////////////////////
// Now get the status:
// Submitting a status request works like submitting a regular
// request. In addition to the FTFReturnObject, the function will
// also return an FTFSummaryObject containing the status of the
// request. Here is an example of the summary object:
//<FTFSummaryObject>
//    <ftfId>dl3533d1-ef85-11d3-b36d-bcf89f5a8e56</ftfId>
//    <timeStamp>03/02/2000 10:26:54</timeStamp>
//    <ftfcode>628</ftfcode>
//    <lqm>QMGR1</lqm>
//    <oqm>QMGR1</oqm>
//    <oqmStatus>REQUEST_FAILED          </oqmStatus>
//    <oqmTimeStamp>03/02/2000 10:26:54</oqmTimeStamp>
//    <sqm>QMGR1</sqm>
//    <sqmStatus>REQUEST_FAILED          </sqmStatus>
//    <sqmTimeStamp>03/02/2000 10:26:54</sqmTimeStamp>
//    <spath>c:\spath\smltxt</spath>
//    <dqm>QMGR1</dqm>
//    <dqmStatus>UNKNOWN_STATUS          : 0</dqmStatus>
//    <dqmTimeStamp>N/A</dqmTimeStamp>
//    <dpath>d:\dpath\smalltxt</dpath>
//    <rqmStatus>REQUEST_SUBMITTED       </rqmStatus>
//    <rqmTimeStamp>03/02/2000 10:26:54</rqmTimeStamp>
//    <groupName/>
//    <label/>
//</FTFSummaryObject>
//
// Note: If no filters are specified in the FTFStatObject, then
//        it is possible for multiple SummaryObjects to be
//        returned!
////////////////////////////////////

// Notice in the sample below that we are setting
FTFStatObject.purge = 1,
// so that it purges the status. If you don't want to remove the
// status record, then set FTFStatObject.purge = 0.

```

## **FTFBPI Wrapper**

### *FTFBPI Wrapper*

```
var Stat=XMObject::create("FTFStatObject");
Stat->cfile = "++CFILE++";
Stat->lqm = Request->lqm;
Stat->purge = 1;
Stat->ftfid = Request->FTFReturnObject->ftfid;

FTFStatXMS(0,Stat);
if(Stat->FTFReturnObject->RC1!=0)
{
    print("There was a problem processing the FTFStat request");
    print(Stat->asString());
    return(1);
}
else
{
    print("The status request was succesful!");
    print(Stat->asString());
}
return(0);
}
```



## **FTFPING.XMS Sample**

```
=====
FTFPING.XMS Sample
=====
/*****
** Source      : FTFPING.XMS
** Date       : 09/25/2001
** Description: This is a sample script, which shows how to submit an
**              FTFping request using BPI.
** Required Changes:
**      Change all references to the following as appropriate:
**      ++CFILE++   FTF configuration file to use
**      ++LQM++     Queue Manager to connect to
**      ++DQM++     Destination Queue Manager to use
**      ++SQM++     Source Queue Manager to use
**      ++OQM++     Originating Queue Manager to use
*****/
main ()
{
////////////////////////////////////////////////////
// The first step is to create an FTFpingObject.  This object will hold
// all information needed by FTF to create and process the ping
// request. The FTFpingObject elements are based off of the FTF
// command-line arguments. For example -cfile, -lqm, -dqm, and
// -timeout become element names as shown below.
////////////////////////////////////////////////////

    object FTFpingObject;
    var Ping = XMOBJECT::create("FTFPingObject");
    Ping->cfile = "++CFILE++";
    Ping->lqm = "++LQM++";
    Ping->sqm = "++SQM++";
    Ping->dqm = "++DQM++";
    Ping->oqm = "++OQM++";
    Ping->timeout = 30;

////////////////////////////////////////////////////
// Load the DLL and import the function.
////////////////////////////////////////////////////
    var mh = loadModule("FTFBPI.dll");
    import long FTFpingXMS(long,xmoptr) from mh;

    print("Trying to ping: " + Ping->dqm);

////////////////////////////////////////////////////
// Submit the FTFping request.
////////////////////////////////////////////////////
    FTFpingXMS(0,Ping);
}
```

## FTFBPI Wrapper

### FTFBPI Wrapper

```
////////////////////////////////////
// FTF inserts an FTFReturnObject into the original FTFPingObject.
// This object contains information about the FTFPing result. In this
// case we check the RC1 element to determine if the ping was
// successful. This is an example of an FTFPingObject after an
// FTFPingXMS call:
//<FTFPingObject>
//   <cfile>c:\FTF\bin\ftfconfig.ini</cfile>
//   <lqm>QMGR1</lqm>
//   <dqm>QMGR2</dqm>
//   <timeout>30</timeout>
//   <FTFReturnObject>
//     <RC1>0</RC1>
//     <RC2>0</RC2>
//     <ErrorMsg />
//     <KBytes>510</KBytes>
//     <Time>0</Time>
//     <FTFVersion>4.1.0</FTFVersion>
//   </FTFReturnObject>
//</FTFPingObject>
////////////////////////////////////
    if (Ping->FTFReturnObject->RC1!=0)
    {
        print("FTFPing FAILED " + Ping->dqm);
        return(1);
    }
    else
    {
        print("FTFPing SUCCESSFUL " + Ping->dqm);
    }

    print(Ping->asString());

    return(0);
}
```

\*\*\*\*\*

NOTE: This release is designed to run with enableNet Business Process Integrator (BPI) version 3.1.1. The platforms that currently support this functionality are AIX 4.3, HP10.20, HP11.00, Win32, and OS/390.

\*\*\*\*\*

---

---

# Appendices

## Contents

- Appendix A, “Security AuthorizationExit JCL”
- Appendix B, “Tivoli Data Exchange GUI Environment Variables”



# Security Authorization Exit JCL

This appendix contains two sets of sample JCL. The first example relinks FTFSTART as APF authorized. The second starts FTFSTART and specifies an options file.

## **FTFSTART Relink JCL**

The following JCL file relinks the FTFSTART command as APF authorized.

```
//LINKFTF JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*****
//* (C) COPYRIGHT MESSAGEQUEST, INC. V2.3.0 *
//*****
//*****
//*
//* THIS Job will relink FTFSTART as APF authorized
//* Change the SYSLIB and SYSLMOD to your current FTF loadlib
//*
//* Change ++FTFHLQ++ to the high-level-qualifier for your FTF
//*loadlib
//*
//*****
//LINK0017 EXEC PGM=IEWL,
//          PARM=('SIZE=(800K,100K),NCAL,LIST,LET,XREF,AMODE=31',
//          'RMODE=ANY,COMPAT=LKED,STORENX,CALL')
//*****
//*
//*****
//SYSLIB DD DSN=++FTFHLQ++.FTF.LOADLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLMOD DD DSN=++FTFHLQ++.FTF.LOADLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSALLDA,
//          SPACE=(3120,(760,760))
//SYSLIN DD *
//          INCLUDE SYSLIB(FTFSTART)
//          ENTRY CEESTART
//          SETCODE AC(1)
//          NAME FTFSTART(R)
```

## FTFSTART Execution JCL

This section contains JCL that starts the FTFSTART module and uses an options file to specify command-line arguments. Before you use this JCL, replace the high-level qualifiers listed in the first column of the following table with the replacement values described in the second column.

High-Level Qualifier	Replacement Value
++QMGR++	The name of the queue manager to which TDEconnects (for instance, MQA1)
++FTFHLQ++	The name of the TDE data set high-level qualifier (for instance, FTFV2)
++MQHLQ++	The name of the MQSeries data set high-level qualifier (for instance, SYS1.V114)
++FTFINI++	The fully qualified path and name of the TDE configuration file (for instance, FTFV2.INI)
++FTFOFILE++	The fully qualified path and name of the options file (for instance, FTFV2.OPTIONS)

The following symbolic changes are optional:

- MEM='0M'
- OUTCLASS='\*'

Use the following JCL to start the FTFSTART module.

```
//FTFJOB JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*****
//* (C) COPYRIGHT MESSAGEQUEST, INC. V2.2.0 *
//*****
//* START THE FTF/MQ FRAMEWORK *
//*****
//* REPLACE THE JOB CARD ABOVE WITH A VALID JOB CARD *
//* *
//* REQUIRED SYMBOLIC CHANGES : *
//* *
//* ++QMGR++ QUEUE MANAGER THAT FTF/MQ WILL CONNECT *
//* TO (E.G. MQA1) *
//* *
//* ++FTFHLQ++ HIGH LEVEL QUALIFIER OF THE FTF/MQ *
//* DATASETS (E.G. FTFV2) *
```

## Security Authorization Exit JCL

### FTFSTART Execution JCL

```

//*
//*      ++MQHLQ++      HIGH LEVEL QUALIFIER OF THE MQSERIES      *
//*                      DATASETS (E.G. SYS1.V114)                *
//*                                                                *
//*      ++FTFINI++     FULL NAME OF THE FTF/MQM DEFAULT          *
//*                      INITIALIZATION FILE (E.G. FTFV2.INI)      *
//*                                                                *
//*      ++FTFOFILE++   FULL NAME OF THE FTF/MQM DEFAULT          *
//*                      OPTIONS FILE (E.G. FTFV2.INI)             *
//*                                                                *
//* OPTIONAL SYMBOLIC CHANGES :                                   *
//*      MEM='OM'                                                *
//*      OUTCLASS='*'                                           *
//*****
//FTFSTART PROC QMGR=++QMGR++,
//      FTFHLQ='++FTFHLQ++',
//      MQMHLQ='++MQHLQ++',
//      INIFILE='++FTFINI++',
//      MEM='OM',
//      OUTCLASS='*'
//STEP1   EXEC PGM=FTFSTART,
//      REGION=&MEM,PARM='-LQM &QMGR -CFILE DD:FTFINI -OFILE
DD:FTFOFILE'
//STEPLIB DD DSN=&FTFHLQ..FTF.LOADLIB,DISP=SHR
//      DD DSN=&MQMHLQ..SCSQAUTH,DISP=SHR
//      DD DSN=&MQMHLQ..SCSQANLE,DISP=SHR
//FTFINI  DD DSN=&INIFILE,DISP=SHR
//FTFOFILE DD DSN=++FTFOFILE++,DISP=SHR
//SYSOUT  DD SYSOUT=&OUTCLASS
//SYSPRINT DD SYSOUT=&OUTCLASS
//SYSERR  DD SYSOUT=&OUTCLASS
//SYSUDUMP DD SYSOUT=&OUTCLASS
//FTFSTART PEND
//*
//FTFSTART EXEC FTFSTART
//

```



# Tivoli Data Exchange GUI Environment Variables

This appendix lists the environment variables set in the command file used to start Tivoli Data Exchange (TDE) GUI. It also describes the environment variable setting necessary if you want to run the Tivoli Data Exchange (TDE) GUI without using the command file.

## Command-File Environment Variables

The following table lists and describes the environment variables used in the FTFGUI command file. This command file is named *ftfgui.cmd* and it resides in the Tivoli Data Exchange (TDE) installation directory.

Variable	Description
FTF_CLASS_PATH	This variable is used to populate the CLASSPATH variable with required TDE GUI classes. If you use this variable to set the environment variables permanently, you must include %FTF_CLASSPATH% in your CLASSPATH environment variable. <b>Example:</b> %FTFGUI_HOME%\ftfgui.jar  AIX must include references to the Swing classes, the TDE GUI classes, and the JDK classes. <b>AIX Example:</b> %SWING_HOME%\swing.jar;%FTFGUI_HOME%\ftfgui.jar;%JDK_HOME%\lib\classes.zip
FTFGUI_HOME	This variable must contain the location of the TDE GUI's Java classes. Example: c:\jdk1.2
SWING_HOME (AIX only)	This variable must contain the location of the Java Foundation Classes' (JFC's) Swing classes. <b>Example:</b> /opt/swing_1.1
JDK_HOME (AIX only)	This variable must contain the location of the Java Development Kit (JDK) or the Java Runtime Environment (JRE). <b>Example:</b> /usr/jdk_base

**Tivoli Data Exchange GUI Environment Variables**  
*Variable Settings Outside the Command File*

Variable	Description
PATH (UNIX)	\$FTFGUI_HOME must be appended to the PATH variable. <b>Example:</b> \$PATH:\$FTFGUI_HOME
FTFGUI_LIBRARY_PATH (UNIX)	This variable must contain the location of the ftfgui library. <b>Example:</b> \$HOME/FTFGUI_HOME/lib
LD_LIBRARY_PATH (UNIX)	\$FTFGUI_LIBRARY_PATH must be appended to the LD_LIBRARY_PATH variable. <b>Example:</b> \$LD_LIBRARY_PATH:\$FTFGUI_LIBRARY_PATH

**Variable Settings Outside the Command File**

To run the TDE GUI without using the command file, you must include the following directories in your CLASSPATH environment variable:

- the directory containing the TDE GUI's Java classes (for instance, *c:\jtf\ftfgui*)

If you do not use Java Platform 2 (JDK 1.2), include the following directories in your CLASSPATH environment variable to run the TDE GUI:

- the directory containing the JFC's Swing classes (for instance, */opt/swing\_base*)
- the directory containing the JDK (for instance, */usr/jdk\_base*)
- LD\_LIBRARY\_PATH must include the location of the ftfgui library (for instance, *\$HOME/FTFGUI\_240/lib*) (UNIX only)

# Index

## Numerics

1001 error	354	1346 error	355
1029 error	354	1349 error	355
1030 error	354	1449 error	355
1035 error	354	1500 error	355
1050 error	354	1504 error	356
1052 error	354	1510 error	356
1053 error	354	1511 error	356
1055 error	354	1514 error	356
1060 error	354	1530 error	356
1065 error	354	1531 error	356
1066 error	354	1533 error	356
1075 error	355	1534 error	356
1076 error	355	2010 error	356
1080 error	355	2012 error	356
1081 error	355	2120 error	356
1082 error	355	2131 error	356
1085 error	355	3020 error	356
1086 error	355	3021 error	356
1175 error	355	3050 error	356
1220 error	355	3109 error	356
1221 error	355	3111 error	356
1231 error	355	346 error	354
1293 error	355	4000 error	357
1294 error	355	4002 error	357
		4200 error	357
		4211 error	357
		4213 error	357

## B

- 4219 error 357
- 4811 error 357
- 4812 error 357
- 4813 error 357
- 4814 error 357
- 5000 error 357
- 5020 error 357
- 5030 error 357
- 6422 error 357
- 9000 error 357
- 9243 error 357

## A

- alcunit argument
  - FTF 46
  - FTFLOG 129
  - FTFSUB 111
- AlcUnit element 326
- allocation unit 46, 111, 129
- alternate user authority 314
- append mode 40
- appending data to an existing file 40
- architectural overview 17
- Argument #1 416
- Argument #2 416
- Arguments
  - FTFSUB 110
- arguments
  - File-to-Message connector 397
  - FTF 26–28, 39, 45, 49, 99, 113
  - FTFBRK 87–88
  - FTFCFG 118–119, 122
  - FTFCNCL 50–52
  - FTFEND 123–126
  - FTFLOG 128–129, 131, 133
  - FTFMGR 135–138
  - FTFPING 54–57, 65–66, 335
  - FTFPSP 88–89
  - FTFPUB 92–99, 102, 105
  - FTFRCV 142–146, 150
  - FTFSDR 149–153

- FTFSTAGE 62–65
- FTFSTART 155–157, 159
- FTFSTAT 73–78
- FTFSUB 101, 103–104, 106–108, 110–111
- AS/400 arguments 108
  - FTF 44–45
  - FTFSUB 108
- as400ft
  - FTF 44
  - FTFSUB 108
- as400ft argument
  - FTF 44
  - FTFSUB 108
- AS400FTelevent 326
- assigning arbitrary labels
  - FTF control arguments 35
- authorization arguments
  - FTFPUB 96
- authorizations arguments
  - FTF 32

## B

- binary file 346
- blksize argument
  - FTF 46
  - FTFLOG 129
  - FTFSUB 111
- BlkSize element 326
- block size 46, 111, 129
- BPI script 416
- BPI XMObject 417
- bqm argument
  - FTFBRK 87
  - FTFPUB 92
  - FTFSUB 101
- broker queue manager
  - FTFBRK 87
  - FTFPUB 92
  - FTFSUB 101
- broker stream

FTFPUB 93  
 FTFSUB 103  
 bufno argument  
   FTF 46  
 BufNo element 327

## C

C APIs 209  
 cache size  
   FTFBRK 88  
 cacheSize argument  
   FTFBRK 88  
 cancel transaction elements 335  
 canceling a file-transfer request  
   from the API 183  
 cancelling a transaction  
   FTFPUB 93  
 cancelmode agument  
   FTF 33  
 cancelmode argument  
   FTFPUB 93  
 CancelMode element 327  
 case sensitivity  
   ignore 75  
 ccsid argument  
   FTF 44  
 CCSid element 327  
 cfile argument  
   FTF 31  
   FTFCFG 118  
   FTFCNCL 50  
   FTFEND 124  
   FTFLOG 131  
   FTFMGR 136  
   FTFPING 56  
   FTFPSP 88  
   FTFPUB 95, 102  
   FTFRCV 144  
   FTFSDR 151  
   FTFSTAGE 64  
   FTFSTART 157

FTFSTAT 76  
 CFile element 327, 334–335  
 clientname  
   example 138  
 coded character set identification  
   FTF 44  
   FTFSUB 108  
 coded character set identification argument  
   FTFSUB 108  
 coffiguration file  
   FTFPUB 95, 102  
 command  
   FTF 25  
   FTFCFG 117  
   FTFCNCL 50  
   FTFEND 121  
   FTFLOG 127  
   FTFMGR 134  
   FTFPING 53  
   FTFRCV 141  
   FTFSDR 148  
   FTFSTAGE 62  
   FTFSTART 155  
   FTFSTAT 72  
 command line interface 23  
 command-line options 417  
 commands  
   FTFSUB 100  
 component 123  
 components 17  
   shutting down 192  
 compress argument  
   FTF 39  
   FTFPUB 98  
 Compress element 327  
 configuration file  
   FTF 31  
   FTFCFG 118  
   FTFCNCL 50  
   FTFEND 124  
   FTFLOG 131  
   FTFMGR 136  
   FTFNodeAlias 139  
   FTFPING 56

## D

- FTFPSP 88
- FTFRCV 144
- FTFSDR 151
- FTFSTAGE 64
- FTFSTART 157
- FTFSTAT 76
- configuration queue
  - FTFPUB 95, 102
- configuration queue name
  - FTF 31
  - FTFCNCL 51
  - FTFEND 124
  - FTFLOG 131
  - FTFMGR 136
  - FTFPING 56
  - FTFPSP 88
  - FTFRCV 144
  - FTFSDR 151
  - FTFSTAGE 65
  - FTFSTART 157
  - FTFSTAT 77
- control arguments
  - FTFSUB 104
- cpt argument
  - FTFEND 123
- cq argument
  - FTF 31
  - FTFCNCL 51
  - FTFEND 124
  - FTFLOG 131
  - FTFMGR 136
  - FTFPING 56
  - FTFPSP 88
  - FTFPUB 95, 102
  - FTFRCV 144
  - FTFSDR 151
  - FTFSTAGE 65
  - FTFSTART 157
  - FTFSTAT 77
- CQ element 327, 334, 336
- create library
  - FTF 44
  - FTFSUB 108
- create library argument
  - FTF 44
  - FTFSUB 108
  - create mode 40
  - creating a destination file 40
  - CrtLib element 327
  - custom status information
    - submitting to status subsystem 207

## D

- Data Specification Arguments
  - FTFPUB 97
- data specification arguments
  - FTF 39–43
  - FTFSUB 106
- data transfer
  - appending data to an existing file 40
  - creating a destination file 40
- data wrapping 42
- DATAEX 345
- DataExtract
  - File-to-Message connector 396
- DATALOAD 345
- DataLoad
  - File-to-Message connector 396
- dataset
  - Generation Data Group 28, 106
  - PDS member 28, 106
- dataset naming conventions 106
- date argument
  - FTFSTAT 75
- date range
  - beginning 75
  - ending 75
- DBCS argument
  - FTF 39
- ddata argument
  - File-to-Message connector 398
  - FTF 39–40
- DData element 327
- default client queue naming convention 139
- delete source file 33

- deleting status information 196
- delimiter processing
  - File-to-Message connector 403
- delsrc argument
  - File-to-Message connector 399
- Delsrc element 327
- destination file
  - creating 40
- destination path
  - FTF 28
  - FTFSTAT 75
  - FTFSUB 106
- destination queue manager
  - FTF 27
  - FTFPING 54
  - FTFSTAT 75
- detail argument 104
- detailed subscription information 104
- dirblks argument 47, 110
- DirBlks element 327
- directory blocks 47, 110
- double byte character set
  - FTF 39
- dpath argument
  - File-to-Message connector 398
  - FTF 28
  - FTFSTAT 75
  - FTFSUB 106
- Dpath element 327
- dqm
  - and stageonly 36
- dqm argument
  - FTF 27
  - FTFPING 54
  - FTFSTAT 75
- Dqm element 328, 334
- dtype argument
  - File-to-Message connector 398
  - FTF 39–40
- DType element 328

## E

- edate argument
  - FTFSTAT 75
- elements
  - cancel transactions 335
  - ping transaction 334
  - request transaction 326
- ENDDLM 404, 406
- entry point of DLL 38, 96, 105
- entry points
  - DATAEX 345
  - DATALOAD 345
  - MULTIEX 345
  - MULTLOAD 345
  - USAGE 345
- Environment Arguments 88
  - FTFPUB 95
- environment arguments
  - FTF 31–32
  - FTFSUB 102
- error codes
  - error code 1001 354
  - error code 1029 354
  - error code 1030 354
  - error code 1035 354
  - error code 1050 354
  - error code 1052 354
  - error code 1053 354
  - error code 1055 354
  - error code 1060 354
  - error code 1065 354
  - error code 1066 354
  - error code 1075 355
  - error code 1076 355
  - error code 1080 355
  - error code 1081 355
  - error code 1082 355
  - error code 1085 355
  - error code 1086 355
  - error code 1175 355
  - error code 1220 355
  - error code 1221 355

- error code 1231 355
- error code 1293 355
- error code 1294 355
- error code 1346 355
- error code 1349 355
- error code 1449 355
- error code 1500 355
- error code 1504 356
- error code 1510 356
- error code 1511 356
- error code 1514 356
- error code 1530 356
- error code 1531 356
- error code 1533 356
- error code 1534 356
- error code 2010 356
- error code 2012 356
- error code 2120 356
- error code 2131 356
- error code 3020 356
- error code 3021 356
- error code 3050 356
- error code 3109 356
- error code 3111 356
- error code 346 354
- error code 4000 357
- error code 4002 357
- error code 4200 357
- error code 4211 357
- error code 4213 357
- error code 4219 357
- error code 4811 357
- error code 4812 357
- error code 4813 357
- error code 4814 357
- error code 5000 357
- error code 5020 357
- error code 5030 357
- error code 6422 357
- error code 9000 357
- error code 9243 357
- error conditions
  - Multi-File Connector 354
- error messages
  - File-to-Message connector 413
  - esoteric unit name, setting 46, 110, 128, 304
  - examples
    - File-to-Message connector 407–410, 412
    - FTFCFG 118–119
    - FTFCNCL 52
    - FTFPING 54–55, 57
    - FTFSTAGE 64–65
    - FTFSTAT 73–74, 76–77
  - exit argument
    - FTF 38
    - FTFPUB 96, 105
  - exit DLL
    - FTFSTART 155–156
  - Exit element 328
  - exit entry point
    - FTFSTART 156
  - exit number 38, 96, 105
  - exitdata argument
    - FTF 38
    - FTFPUB 97, 105
  - ExitData element 328
  - exitdll argument
    - DLL name 38, 96, 105
    - FTF 38
    - FTFPUB 96, 105
    - FTFRCV 143
    - FTFSDR 150
    - FTFSTART 155–156
  - ExitDll element 328
  - exitentry argument
    - FTF 38
    - FTFPUB 96, 105
    - FTFRCV 143
    - FTFSDR 150
    - FTFSTART 156
  - ExitEntry element 328
  - expire time
    - FTFPUB 93
  - expiry
    - expiration duration 33
  - expiry argument
    - FTFPUB 93
  - Expiry element 328



**F**

- FAIL mode 42
- FFTId Object 322
- file
  - binary 346
- file argument
  - FTFSTAGE 63
- file auxiliary storage pool
  - FTF 44
  - FTFSUB 108
- file auxiliary storage pool argument
  - FTF 44
  - FTFSUB 108
- file for staging queue 63
- file id
  - FTFSTAGE 63
  - FTFSTAT 75
- file id for staging queue 63
- file label
  - FTFSTAT 75
- file organization 129
- file organization 129
- file resource security
  - enabling 315
- file text
  - FTF 44
  - FTFSUB 108
- file text argument
  - FTF 44
  - FTFSUB 108
- file type 39, 43, 97
- FileASP element 328
- File-to-Message connector
  - arguments 397
  - DataExtract 396
  - DataLoad 396
  - ddata argument 398
  - delimiter processing 403
  - delsrc argument 399
  - dpath argument 398
  - dtype argument 398
  - error messages 413
  - examples 407–410, 412
  - logging 406
  - modifiers 399
  - operating systems supported 396
  - sdata argument 397
  - spath argument 398
  - stype argument 397
- File-to-Message connector installation 396
- File-to-Message modifiers
  - BATCH 399, 401
  - CR 401
  - DataExtract 399
  - DataLoad 401
  - DELDLM 401
  - ENDDLM 402
  - FIXED 402
  - LOGFILE 399, 401
  - MSGID 400
  - NOT\_PERSISTENT 402
  - PERSISTENT 402
  - RECSIZE 403
  - SINGLE 400, 403
  - STARTDLM 403
  - SYNCQ 403
  - WAIT 400
  - WAITTIME 400
- FileTxt element 328
- FIXED 405
- format argument
  - FTFSTAT 73
- formatting status records
  - FTFSTAT 73
- freeing status memory 198, 200
- fromstage argument
  - FTF 36
- FromStage element 328
- FTF 415
  - alcunit 46
  - as400ft 44
  - authorization arguments 32
  - blksize 46
  - bufno 46
  - cancelmode 33
  - ccsid 44

- cfile 31
- command 25
- compress 39
- cq 31
- crtlib 44
- DBCS 39
- ddata 39–40
- delsrc 33
- dirblks 47
- double byte character set 39
- dpath 28
- dqm 27
- dtype 39–40
- environment arguments 31
- exit 38
- exitdata 38
- exitdll 38
- exitentry 38
- expiry argument 33
- fileasp 44
- filetxt 44
- fromstage argument 36
- ftfid 36
- help argument 49, 99, 113
- i2l 34
- idl 34
- id3 34
- immed 34
- label 35
- libasp 45
- libtxt 45
- lqm 26
- lrecl 47
- makedirs 40
- mode argument 39–40
- mode of file processing 39–40
- model 47
- msgsize 35
- ofile 32
- oqm 26
- org 47
- OS/390 arguments 46, 48
- padchar argument 41
- padding character 41

- pool 39, 41
- primary 48
- priority 36
- process arguments 33
- queue manager arguments 26
- rcdlen 45
- recfmt 48
- recpad 39, 42
- recwrap 39, 42
- sdata 39, 43
- secondary allocation unit 48
- secondary argument 48
- source and target file arguments 27
- spath 27
- specifying a destination data type 39–40, 43
- specifying a source data type 39, 43
- sqm 27
- stage argument 36
- stageonly 36
- stageonly control argument 36
- stagepersist 36
- standard arguments 49
- stype 39, 43
- syntax 26
- transfer and performance arguments 39
- transpersist 36
- trusted 37
- type 39, 43
- unit name 48
- user exit arguments 38–39
- version 32
- volser 48
- wait 37
- wait control argument 37
- FTF command
  - append mode 40
  - create mode 40
  - examples 27, 37, 39, 43, 48–49, 87–88, 99
  - noreplace mode 40
- FTF -help option 417
- FTF\_CLASSPATH environment variable 429
- FTFAS400FileInfo data structure
  - data elements 213

- described 213
- ftfauth argument
  - FTFRCV 144, 150
- FTFBPI 415
- FTFBPI calls 417
- FTFBPI DLL 416
- FTFBRK 86
  - bqm 87
  - cacheSize 88
  - Environment Arguments 88
  - Help Arguments 89
  - lfile 88
  - lqm 87
  - Process Arguments 87
  - Queue Manager Arguments 87
- FTFBRK command
  - examples 89
- FTFCA data structure 213, 215
  - data elements 215
  - described 215
- FTFCancel function 183
  - description 183
  - parameters 184
  - prototype 183
- FTFCancelInfo data structure
  - data elements 217
  - described 217
- FTFCF2M error messages 413–414
- FTFCFG
  - arguments 118, 122
  - cfile 118
  - command 117
  - configuration information 117
  - examples 118–119
  - help 119
  - lqm 118, 122
  - node 118, 122
  - ofile 119
  - syntax 117–119, 122
  - version 119
- FTFCNCL
  - arguments 51
  - cfile 50
  - command 50
  - cq 51
  - examples 52
  - ftfid 51
  - help 52
  - lqm 51
  - notes 52
  - ofile 51
  - oqm 51
  - syntax 50
  - version 51
- FTFEND
  - cfile 124
  - command 121
  - cpt 123
  - cq 124
  - example 125
  - help 126
  - immediate 123
  - notes 126
  - ofile 125
  - process-control arguments 123
  - queue manager node arguments 122–123
  - quiesce 123
  - standard arguments 124, 126
  - timeout 124
  - version 125
- FTFExitAuthInfo data structure
  - data elements 219
  - description 219
- FTFExitFileInfo 221
- FTFExitFileInfo data structure
  - described 221
- FTFExitInfo data structure 225
  - data elements 221, 225
  - described 225
- FTFExitJobInfo data structure
  - data elements 229
  - described 229
- FTFExitQMgsInfo data structure
  - data elements 232
  - definition 232
- FTFExitRequestInfo data structure
  - data elements 234
  - description 234

- FTFExitSourceFileInfo data structure
  - data elements 236
  - described 236
- FTFExitStatusOffload data structure
  - data elements 239
  - described 239
- FTFExitTargetFileInfo data structure
  - data elements 241
  - described 241
- FTFExitUserInfo data structure
  - data elements 245
  - described 245
- FTFGUI\_HOME environment variable 429
- ftfid argument
  - FTF 36
  - FTFCNCL 51
  - FTFSTAT 75
- FTFId element 328, 336
- FTFIdentifiersInfo data structure 247
- FTFJobInfo data structure 248
  - data elements 247–248
  - described 247–248
- FTFLOG
  - alcunit 129
  - arguments 129
  - blksize 129
  - cfile 131
  - command 127
  - cq 131
  - description 127
  - examples 130
  - help argument 133
  - ldir 128
  - lfile 128
  - lqm 128
  - lrecl 129
  - ofile 131
  - org 129
  - OS/390 arguments 128
  - primary 129
  - q argument 128
  - recfmt 129
  - secondary allocation unit 129
  - secondary argument 129
  - standard arguments 127, 130, 133
  - syntax 127
  - unit name 129
  - version 131
  - volser 129
- FTFMGR
  - cfile argument 136
  - command 134
  - cq 136
  - example 135, 137
  - help 138
  - jsdataset 137
  - lfile 135
  - lqm 135
  - nodename 135
  - ofile 137
  - standard arguments 136, 138
  - startup arguments 134
  - syntax 134
  - version 137
- FTFMGR - Standard Environment Arguments 136
- FTFNodeAlias 139
- FTFNotifyObject 323
- FTFPING 53, 415
  - cfile 56
  - command 53
  - cq 56
  - dqm 54
  - examples 54–55, 57
  - help argument 57, 66
  - lqm 54
  - msgsize 55
  - notes 58
  - ofile argument 56, 65
  - oqm 54
  - priority 55
  - process control arguments 55
  - queue manager node arguments 54
  - sqm 54
  - standard arguments 56–57
  - syntax 53
  - timeout 55, 335
  - uses 53, 185

- version 56, 65
- FTFPing function 185
  - description 185
  - parameters 186
  - prototype 185
- FTFPingInfo 324
- FTFPingInfo data structure
  - data elements 251
  - described 251
- FTFPingXMS
  - FTFBPI Wrapper 416
- FTFPSP
  - cfile 88
  - cq 88
  - help argument 89
  - ofile 89
  - version 89
- FTFPSP command
  - examples 89
- FTFPUB
  - authorizations arguments 96
  - bqm 92
  - cancelmode 93
  - cfile 95, 102
  - compress 98
  - cq 95, 102
  - data specification arguments 97
  - environment arguments 95
  - exit 96, 105
  - exitdata 97, 105
  - exitdll 96, 105
  - exitentry 96, 105
  - expiry 93
  - label 94
  - lqm 92
  - msgsize 99
  - ofile 95, 102
  - oqm 92
  - padchar 97
  - priority 93
  - process arguments 93
  - recpad 98
  - spath 93
  - sqm 92
  - stream 93
  - syntax 90
  - type 97
  - user exit arguments 96
  - version 95, 102
  - wait 93
- FTFPUB command
  - examples 92, 94–95, 97, 99
- FTFQMgsInfo data structure
  - data elements 254
  - described 254
- FTFRCV
  - cfile 144
  - command 141
  - connector arguments 147
  - cq 144
  - examples 143, 146, 150
  - exitdll 143
  - exitentry 143
  - ftfauth 144, 150
  - help 146
  - invoking security authorization exits 143, 150
  - jsdataset 145
  - lfile 142
  - lqm 142
  - nodename 142
  - ofile 145
  - standard arguments 144, 146, 151
  - su,su argument
    - FTFRCV 145
  - sync 142
  - syntax 141, 148
  - version 145
- FTFReplyObject 323
- FTFReq API 190
- FTFReq function
  - description 187
  - parameters 188, 191
  - prototype 188, 190
- FTFRequestMsgInfo data structure 256
  - data elements 256
  - described 256
- FTFRequestXMS

- FTFBPI Wrapper 416
- FTFReturnObject 322
- FTFSDR
  - cfile 151
  - command 148
  - connector arguments 153
  - cq 151
  - examples 152
  - exitdll 150
  - exitentry 150
  - help 153
  - jsdataset 152
  - lfile 149
  - lqm 149
  - nodename 149
  - ofile 152
  - standard arguments 153
  - sync 149
  - version 152
- FTFShutdown function
  - description 192
  - parameters 193
  - prototype 192
- FTFShutdownInfo data structure
  - data elements 259
  - described 259
- FTFShutdownReply data structure
  - data elements 261
  - described 261
- FTFSourceFileInfo data structure
  - data elements 263
  - described 263
- FTFSTAGE
  - all argument 63
  - cfile 64
  - command 62
  - cq 65
  - examples 64– 65
  - file 63
  - ftfid 63
  - lqm 62
  - oqm 63
  - purge 63
  - purging all from staging queue 63
  - query 63
  - queue manager node arguments 62– 63
  - sqm 63
  - standard arguments 64, 66
  - syntax 62
  - wait argument 63
- FTFSTAGE - Process-Control Arguments 63
- FTFStage function
  - description 194
  - parameters 195
  - prototype 194
- FTFStagedListMsgInfo data structure
  - data elements 266
  - described 266
- FTFStagedTransactionMsgInfo data structure
  - data elements 267
  - described 267
- FTFStageMsgInfo data structure
  - data elements 269
  - described 269
- FTFSTART
  - cfile 157
  - command 155
  - cq 157
  - examples 156– 159
  - exitdll 155– 156
  - exitentry 156
  - help 159
  - lqm 157
  - ofile 159
  - rcvauth 156
  - scrauth 155
  - security authorization arguments 155
  - startup arguments 157
  - syntax 155
  - SYSOUTCLASS argument 156
  - version argument 159
- FTFSTAT 75, 415
  - additional arguments 74
  - cfile 76
  - command 72
  - cq 77
  - date 75
  - dpath 75

- dqm 75
- examples 73– 74, 76– 77
- format 73
- ftfid 75
- help 78
- i argument 75
- ignore case sensitivity 75
- label 75
- lqm 73
- manager node arguments 73
- notes 78
- ofile 77
- oqm 75
- orphans 74
- process control arguments 73
- purge 73
- rqm 75
- samples 78
- spath 76
- sqm 76
- standard arguments 76, 78
- status type argument 76
- syntax 72
- version 77
- FTFSTAT - Standard Environment Arguments 76
- FTFStatDetail data structure 271
  - data elements 271
  - described 271
- FTFStatDetailList data structure
  - data elements 281
  - described 281
- FTFStatSummary data structure 282
  - data elements 282
  - description 282
- FTFStatSummaryList data structure 286
  - data elements 286
  - description 286
- FTFStatusDelete function
  - description 196
  - parameters 196
  - prototype 196
- FTFStatusDeleteInfo data structure
  - data elements 288
  - described 288
- FTFStatusFreeDetailList function 198
  - description 198
  - parameters 198
  - prototype 198
- FTFStatusFreeSummaryList function
  - description 200
  - parameters 201
  - prototype 200
- FTFStatusGetDetailList function
  - description 202
  - parameters 203
  - prototype 202
  - usage 278
- FTFStatusGetDetailListInfo data structure
  - data elements 290
  - described 290
- FTFStatusGetSummaryList function
  - parameters 205
  - prototype 204
- FTFStatusGetSummaryListInfo data structure
  - data elements 292
  - described 292
- FTFStatusSummaryFilter data structure 294
  - data elements 294
  - described 294
  - usage notes 294
- FTFStatXMS
  - FTFBPI Wrapper 416
- FTFSUB 100, 108
  - 110
  - alcunit 111
  - Arguments 110
  - arguments 111
  - AS/400 arguments 108
  - as400ft 108
  - blksize 111
  - bqm 101
  - ccsid 108
  - crtlib 108
  - data specification arguments 106
  - detail 104
  - dirblks 110
  - dpath 106

## H

- environment arguments 102
- fileasp 108
- filetxt 108
- help arguments 112
- libasp 108
- libtxt 108
- lqm 101
- lrecl 110
- makedirs 107
- mode argument 107
- mode of file processing 107
- model 111
- node 101
- org 110
- pool 107
- primary 111
- process arguments 103
- purge 104
- purgenode 104
- query 104
- queue manager arguments 101
- queue manager node arguments 92, 101
- rcdlen 108
- recfmt 110
- recwrap 107
- secondary allocation unit 111
- secondary argument 111
- stream 103
- subscription criteria arguments 103
- syntax 100
- topic 103
- unit name 111
- user exit arguments 104– 105
- volser 111
- wait 104
- wait control argument 104
- FTFSUB command
  - examples 102– 103, 106– 107, 109, 111, 113
- FTFSUB environment arguments 102
- FTFSubmitStatusMessage function
  - description 207
  - parameters 208
  - prototype 207

- FTFSubscribeMsgInfo data structure
  - data elements 298
  - described 298
- FTFSubscriptionListMsgInfo data structure
  - data elements 301
  - described 301
- FTFSubscriptionMsgInfo data structure
  - data elements 302
  - described 302
- FTFTargetFileInfo data structure
  - data elements 304
  - described 304
- FTFUserInfo data structure 308
  - data elements 308
  - described 308

## G

- Generation Data Group 106
  - dataset 106
  - Generation Data Group
    - dataset 28
- getting status information 202
- gradual shutdown 123

## H

- help argument
  - FTF 49, 99, 113
  - FTFCFG 119
  - FTFCNCL 52
  - FTFEND 126
  - FTFLOG 133
  - FTFMGR 138
  - FTFPING 57, 66
  - FTFPSP 89
  - FTFRCV 146
  - FTFSDR 153
  - FTFSTART 159
  - FTFSTAT 78



Help Arguments 89  
     FTFPUB 99  
 help arguments  
     FTFSUB 112

## I

i argument  
     FTFSTAT 75  
 ID 400  
 id1 argument  
     FTF 34  
 ID1element 329  
 id2 argument  
     FTF 34  
 ID2 element 329  
 id3 argument  
     FTF 34  
 ID3 element 329  
 identifier 1  
     FTF 34  
 identifier 2  
     FTF 34  
 identifier 3  
     FTF 34  
 identifier for files  
     FTFCNCL 51  
 immed argument  
     FTF 34  
 Immed element 329  
 immediate argument  
     FTFEND 123  
 immediate shutdown 123  
 installation  
     File-to-Message connector 396

## J

JCL data set  
     FTFMGR 137

    FTFRCV 145  
     FTFSDR 152  
 JDK\_HOME environment variable 429  
 jsdataset argument  
     FTFMGR 137  
     FTFRCV 145  
     FTFSDR 152

## L

label argument  
     FTF 35  
     FTFPUB 94  
     FTFSTAT 75  
 Label element 329  
 label value  
     FTFPUB 94  
 ldir argument 128  
 level one security 312  
 level two security 313  
 lfile argument  
     FTFBRK 88  
     FTFLOG 128  
     FTFMGR 135  
     FTFRCV 142  
     FTFSDR 149  
 Libasp element 329  
 library auxiliary storage pool  
     FTF 45  
     FTFSUB 108  
 library auxiliary storage pool argument  
     FTF 45  
     FTFSUB 108  
 library text  
     FTF 45  
     FTFSUB 108  
 library text argument  
     FTF 45  
     FTFSUB 108  
 LibTxt element 330  
 limiting queue permissions 315  
 local queue manager

## M

- FTF 26
- FTFBRK 87
- FTFCFG 118, 122
- FTFCNCL 51
- FTFMGR 135
- FTFPING 54
- FTFPUB 92
- FTFRCV 142
- FTFSDR 149
- FTFSTAGE 62
- FTFSTART 157
- FTFSTAT 73
- FTFSUB 101
- log directory 128
- log file
  - FTFBRK 88
  - FTFLOG 128
  - FTFMGR 135
  - FTFRCV 142
  - FTFSDR 149
- log information
  - eliminating from standard output 136, 143, 150
- log queue 128
- logging
  - File-to-Message connector 406
- logical record length 47, 110, 129
- lqm argument 128
  - FTF 26
  - FTFBRK 87
  - FTFCFG 118, 122
  - FTFCNCL 51
  - FTFMGR 135
  - FTFPING 54
  - FTFPUB 92
  - FTFRCV 142
  - FTFSDR 149
  - FTFSTAGE 62, 73
  - FTFSTART 157
  - FTFSUB 101
- Lqm element 330, 334, 336
- lrecl argument
  - FTF 47
  - FTFLOG 129

- FTFSUB 110
- LRecl element 330

## M

- make directory 40, 107
- Manager
  - description 19
  - starting as background process 136
- message size 35
  - FTFPING 55
- mkdirs argument
  - FTF 40
  - FTFSUB 107
- MkDirs element 330
- mode argument
  - FTF 39–40
  - FTFSUB 107
- Mode element 330
- model 47, 111
- model argument
  - FTF 47
  - FTFSUB 111
- Model element 330
- modes of security 312
  - level one security 312
  - level two security 313
  - no security 312
- modifiers
  - BATCH 399, 401
  - CR 401
  - DataExtract 399
  - DataLoad 401
  - DELDLM 401
  - ENDDLM 402
  - File-to-Message connector 399
  - FIXED 402
  - LOGFILE 399, 401
  - MSGID 400
  - NOT\_PERSISTENT 402
  - PERSISTENT 402
  - RECSIZE 403

SINGLE 400, 403  
 STARTDLM 403  
 SYNCQ 403  
 WAIT 400  
 WAITTIME 400  
 ModifyType element 331  
 msgsize argument  
     FTF 35  
     FTFPING 55  
     FTFPUB 99  
 MsgSize element 331, 335  
 MULTIEX 345  
 Multi-File Connector  
     error conditions 354  
     installation 344  
     operating systems supported 344  
     transferring multiple files 344  
     XML configuration 352  
 MULTLOAD 345

## N

no security 312  
 node argument  
     FTFCFG 118, 122  
     FTFSUB 101  
 node name  
     FTFCFG 118, 122  
     FTFMGR 135  
 nodename argument  
     FTFMGR 135  
     FTFRCV 142  
     FTFSDR 149  
 nopad mode 42  
 noreplace mode 40  
 NotifyData element 331  
 NotifyStatus element 331  
 number of buffers 46

## O

OBJECTNAME 417  
 ofile argument  
     FTF 32  
     FTFCFG 119  
     FTFCNCL 51  
     FTFEND 125  
     FTFLOG 131  
     FTFMGR 137  
     FTFPING 56, 65  
     FTFPSP 89  
     FTFPUB 95, 102  
     FTFRCV 145  
     FTFSDR 152  
     FTFSTART 159  
     FTFSTAT 77  
 OFile element 331, 335–336  
 operating 344  
 options file  
     FTFPUB 95, 102  
 options file name  
     FTFPSP 89  
 oqm argument  
     FTF 26  
     FTFCNCL 51  
     FTFPING 54  
     FTFPUB 92  
     FTFSTAGE 63  
     FTFSTAT 75  
 Oqm element 331, 335–336  
 org argument  
     FTF 47  
     FTFLOG 129  
     FTFSUB 110  
 Org element 331  
 originating queue manager  
     FTF 26  
     FTFCNCL 51  
     FTFPING 54  
     FTFPUB 92  
     FTFSTAGE 63  
     FTFSTAT 75

## Q

orphaned detail files 74

orphans argument  
    FTFSTAT 74

OS/390

    dataset naming conventions 28

    file allocation items 243, 306

    Generation Data Group 28

    PDS member 28

OS/390 arguments

    FTF 46–48

overrides

    FTF 32

    FTFCFG 119

    FTFCNCL 51

    FTFEND 125

    FTFLOG 131

    FTFMGR 137

    FTFPING 56, 65

    FTFRCV 145

    FTFSDR 152

    FTFSTART 159

    FTFSTAT 77

## P

pad mode 42

padchar argument  
    FTF 41

    FTFPUB 97

PadChar element 331

PDS member 106

    dataset 28, 106

    OS/390 28

persistent transfer 36

ping transaction elements 334

pinging components  
    from the API 185

placing a file

    in staging queue 67

pool argument

    FTF 39, 41

    FTFSUB 107

Pool element 332

pool name

    FTF 39, 41

    FTFSUB 107

preemptive cancel flag override  
    in the configuration file 33

preemptive cancellation 33

primary argument

    FTF 48

    FTFLOG 129

    FTFSUB 111

Primary element 332

priority argument

    FTF 36

    FTFPING 55

    FTFPUB 93

Priority element 332, 335

priority of message 36

priority of processing

    FTFPING 55

Process Arguments 87

    FTFPUB 92

process arguments

    FTF 33–37

    FTFSUB 103

purge argument 63, 104

    FTFSTAT 73

purgenode argument 104

purging records

    FTFSTAT 73

purging subscriptions 104

purging the staging queue 63, 71, 104

## Q

q argument 128

query argument 63, 104

querying the staging queue 63, 69, 104

Queue Manager Arguments 87

Queue Manager Node Arguments

    FTFSUB 92

queue manager node arguments

- FTFSUB 101
- queue permissions
  - limiting 315
- quiesce argument
  - FTFEND 123

## R

- Rcdlen element 332
- rcvauth argument
  - FTFSTART 156
- Receiver
  - description 20
  - starting as background process 143, 150
- receiver authorization
  - FTFSTART 156
- receiver entry point
  - FTFSTART 156
- recfmt argument
  - FTF 48
  - FTFLOG 129
  - FTFSUB 110
- Recfmt element 332
- record format 48, 110, 129
- record length
  - FTF 45
  - FTFSUB 108
- record length argument
  - FTF 45
  - FTFSUB 108
- record padding 42
- recpad argument
  - FTF 39, 42
  - FTFPUB 98
- RecPad element 332
- RECSIZE 405
- recwrap argument
  - FTF 39, 42
  - FTFSUB 107
- RecWrap element 332
- ReplyQ element 332
- request transaction elements 326

- Request Type Tag 321
- requesting queue manager
  - FTFSTAT 75
- required fields
  - XML integration 321
- rqm argument
  - FTFSTAT 75
- RwplyQMgrqm element 332

## S

- sdata argument
  - File-to-Message connector 397
  - FTF 39, 43
- Sdata element 333
- sdrauth argument
  - FTFSTART 155
- secondary argument
  - FTF 48
  - FTFLOG 129
  - FTFSUB 111
- Secondary element 333
- security
  - file resource 315
- security authorization exits
  - invoking on FTFRVCV 143, 150
- security concepts 313
- Security Enabling Interface 313
- security modes 312
- selsrc agument
  - FTF 33
- Sender
  - description 19
- sender authorization
  - FTFSTART 155
- SenderAlwaysCheckStage property 69
- sending a file
  - from staging queue 68
- sending messages
  - from stage queue 36
- SETMQAUT 313
- shutdown

- gradual 123
- immediate 123
- shutting down components 192
- SINGLE 406
- software version
  - FTFPUB 95, 102
- source path
  - FTF 27
  - FTFPUB 93
  - FTFSTAT 76
- source queue manager
  - FTF 27
  - FTFPING 54
  - FTFPUB 92
  - FTFSTAGE 63
  - FTFSTAT 76
- source type
  - FTF 39–40, 43
- spath argument
  - File-to-Message connector 398
  - FTF 27
  - FTFPUB 93
  - FTFSTAT 76
- Spath element 333
- specifying a log directory 128
- specifying a log queue 128
- specifying directory blocks 47, 110
- specifying the staging queue 67
- sqm argument
  - FTF 27
  - FTFPING 54
  - FTFPUB 92
  - FTFSTAGE 63
  - FTFSTAT 76
- Sqm element 333, 335
- stage argument
  - FTF 36
- Stage element 333
- stageonly argument
  - FTF 36
- StageOnly element 333
- stagepersist argument
  - FTF 36
- StagePersist element 333
- staging a message
  - FTF 36
- staging messages 36
- staging queue
  - file 63
  - file id 63
  - placing file in 67
  - purging 71
  - querying 69
  - sending a file 68
  - specifying 67
- staging queue ID
  - FTF control argument 36
  - FTF immediate argument 34
- standard output
  - eliminating log information 136, 143, 150
- STARTDLM 404, 406
- status
  - date range 75
- status information
  - deleting 196
  - getting 202
- status memory
  - freeing 198, 200
- status messages
  - manager 211
  - receiver 212
  - sender 211
- status subsystem
  - description 20
- status type argument
  - FTFSTAT 76
- stream argument
  - FTFPUB 93
  - FTFSUB 103
- stype argument
  - File-to-Message connector 397
  - FTF 39, 43
- SType element 333
- submitting a file-transfer request
  - from the API 187
- subscription criteria arguments
  - FTFSUB 103
- SWING\_HOME environment variable 429

sync argument  
     FTFRCV 142  
     FTFSDR 149  
 sync queue  
     FTFRCV 142  
     FTFSDR 149  
 SYSOUTCLASS argument 156

## T

target file block size 46, 111, 129  
 target file organization 47, 110  
 target file unit name  
     FTF 48  
     FTFLOG 129  
     FTFSUB 111  
 target volume serial number 48, 111, 129  
 time limit for processing  
     FTFPING 55, 335  
 timeout argument  
     FTFEND 124  
     FTFPING 55, 335  
 Tivoli Customer Support 12  
 Tivoli Data Exchange  
     description 9–10  
 topic argument  
     FTFSUB 103  
 topic string  
     FTFSUB 103  
 transaction priority  
     FTFPUB 93  
 transpersist argument  
     FTF 36  
 Transpersist element 333  
 troubleshooting FTFPING  
     59  
 TRUNCATE mode 42  
 trusted argument  
     FTF 37  
 Trusted element 333  
 trusted transaction 37  
 type argument 39, 43, 97

Type element 333

## U

unit argument  
     FTF 48  
     FTFLOG 129  
     FTFSUB 111  
 Unit element 334  
 USAGE 345  
 User Exit Arguments  
     FTFPUB 96  
 user exit arguments  
     FTF 38  
     FTFSUB 104  
 user-defined label  
     FTFcommand 35

## V

version  
     FTFPSP 89  
 version argument  
     FTF 32  
     FTFCFG 119  
     FTFCNCL 51  
     FTFEND 125  
     FTFLOG 131  
     FTFMGR 137  
     FTFPING 56, 65  
     FTFPSP 89  
     FTFPUB 95, 102  
     FTFRCV 145  
     FTFSDR 152  
     FTFSTART 159  
     FTFSTAT 77  
 volser argument  
     FTF 48  
     FTFLOG 129  
     FTFSUB 111

## X

VolSer element 334  
volume serial number 48, 111, 129  
VSAM data 348

## W

wait argument  
    FTF 37  
    FTFPUB 93  
    FTFSTAGE 63  
    FTFSUB 104  
Wait element 334  
wait time  
    FTFPUB 93  
wait time for a reply  
    FTF 37  
    FTFSUB 104  
WRAP mode 42

## X

XML Element  
    AlcUnit 326  
    AS400FT 326  
    BlkSize 326  
    BufNo 327  
    CancelMode 327  
    CCSid 327  
    CFile 327, 334–335  
    Compress 327  
    CQ 327, 334, 336  
    CrtLib 327  
    DDData 327  
    Delsrc 327  
    DirBlks 327  
    Dpath 327  
    Dqm element 328, 334  
    DType 328  
    Exit 328  
    ExitData 328

ExitDll 328  
ExitEntry 328  
Expiry 328  
FileASP 328  
FileTxt 328  
FromStage 328  
FTFId 328, 336  
ID1 329  
ID2 329  
ID3 329  
Immed 329  
Label 329  
Libasp 329  
LibTxt 330  
Lqm element 330, 334, 336  
LRecl 330  
MkDirs 330  
Mode 330  
Model 330  
ModifyType 331  
MsgSize 331, 335  
NotifyData 331  
NotifyStatus 331  
OFile 331, 335–336  
Oqm element 331, 336  
Org 331  
PadChar 331  
Pool 332  
Primary 332  
Priority 332, 335  
Rcdlen 332  
Recfmt 332  
RecPad 332  
RecWrap 332  
ReplyQ 332  
ReplyQMgr 332  
Sdata 333  
Spath 333  
Sqm element 333, 335  
Stage 333  
StageOnly 333  
StagePersist 333  
Transpersist 333  
Trusted 333



- Type 333
- Unit 334
- VolSer 334
- Wait 334
- XML Element Names 325
- XML Elements
  - Oqm 335
  - secondary 333
  - SType 333
- XML integration 320
- XML Message Identifier 321

