Tivoli Data Exchange
Version 1.2.0
Technical Reference
Contents

chapter 1  About This Guide  .................................................. 9
  About Tivoli Data Exchange ........................................... 9
  How This Guide Is Organized ......................................... 11
  Contacting Customer Support ...................................... 12
  Where To Look for More Information ............................ 12
  Conventions ................................................................... 13

chapter 2  Tivoli Data Exchange Concepts  .................. 15
  Tivoli Data Exchange Introduction .................................. 15
  Why Tivoli Data Exchange? .......................................... 15
  Tivoli Data Exchange Architectural Overview .................. 17
    Tivoli Data Exchange Interfaces .................................. 18
    Tivoli Data Exchange Manager .................................... 19
    Tivoli Data Exchange Sender ...................................... 20
    Tivoli Data Exchange Receiver .................................... 20
    Tivoli Data Exchange Status ..................................... 20

chapter 3  Tivoli Data Exchange Interface Commands  .................. 23
  Overview .................................................................. 24
  FTF ........................................................................ 25
    Transferring One File to Multiple Destinations ............ 29
  FTCNCL ........................................................................ 52
  FTPING ...................................................................... 55
    Syntax ....................................................................... 55
  FTFSAGE .................................................................... 64
  Using Staging Queues ................................................ 69
  FTFSTAT ..................................................................... 74

chapter 4  Component Configuration Commands ........ 87
  Overview .................................................................. 88
  FTCFG ...................................................................... 89
    Syntax ....................................................................... 91
  FTFEND ................................................................... 93
  FTFLG ....................................................................... 99
  FTFMGR ................................................................. 107
  FFTRCV .................................................................. 115
chapter 5  C API Reference ........................................... 135
FTFCancel .................................................. 137
FTFPing .................................................. 139
FTFReq .................................................. 141
FTFShutdown ........................................... 143
FTFSend .................................................. 145
FTFSendDelete ........................................... 147
FTFSendFreeDetailList ..................................... 149
FTFSendFreeSummaryList .................................. 151
FTFSendGetDetailList ..................................... 153
FTFSendGetSummaryList .................................. 155
FTFSendSubmitStatusMsg ................................. 158

chapter 6  C Data Structures ................................... 161
Component Status Messages ..................................... 162
  Tivoli Data Exchange Manager Status Messages ............. 163
  Tivoli Data Exchange Sender Status Messages ............... 163
  Tivoli Data Exchange Receiver Status Messages .......... 164
FTFAS400FileInfo ........................................ 165
FTFCA ................................................... 167
FTFCancelInfo ........................................... 169
FTFExitAuthInfo ......................................... 171
FTFExitFileInfo .......................................... 173
FTFExitInfo .............................................. 177
FTFExitJobInfo ........................................... 181
FTFExitQMgrsInfo ........................................ 184
FTFExitRequestInfo ....................................... 186
FTFExitSourceFileInfo .................................... 188
FTFExitStatusOffload ..................................... 191
FTFExitTargetFileInfo ................................... 193
FTFExitUserInfo .......................................... 197
FTFIdentifiersInfo ........................................ 199
FTFJobInfo .............................................. 200
FTFPingInfo ................................................ 203
FTFQMgrsInfo ............................................ 206
FTFRequestMsgInfo ....................................... 208
FTFShutdownInfo ......................................... 211
FTFShutdownReply ........................................ 213
FTFSourceFileInfo ....................................... 216
chapter 7  COBOL API Reference ........................................239
Utility Programs .....................................................260
TERMNULL ..................................................................261
SETADDR ..................................................................263
COBOL API Functions ...............................................264
FTFCAN ..................................................................265
FTFEND ..................................................................268
FTFPING ..................................................................271
FTFREQ ..................................................................274
FTFSL ....................................................................277
Compiling the COBOL API ........................................280
    DLL Support .....................................................280
    Dynamically Linking COBOL Applications ..................281

chapter 8  COBOL Data Structures .................................285
FTFCA ....................................................................286
FTF-CANCEL-INFO ..................................................288
FTF-DELETE-INFO ...................................................290
FTF-DETLIST-INFO ..................................................292
FTF-EXIT-INFO .......................................................294
FTFPING-INFO ........................................................300
FTF-REQUEST-MESSAGE-INFO ....................................303
    FTF-QMGRSINFO ................................................303
    FTF-SOURCE-FILE-INFO .......................................305
    FTF-TARGET-FILE-INFO .......................................308
    FTF-JOB-INFO ....................................................313
    FTF-USER-INFO .................................................317
    FTFAS400-FILE-INFO ..........................................320
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 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:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Tivoli Data Exchange</td>
<td>9</td>
</tr>
<tr>
<td>This guide is intended for the following groups:</td>
<td>10</td>
</tr>
<tr>
<td>How This Guide Is Organized</td>
<td>11</td>
</tr>
<tr>
<td>Contacting Customer Support</td>
<td>12</td>
</tr>
<tr>
<td>Where To Look for More Information</td>
<td>12</td>
</tr>
<tr>
<td>Conventions</td>
<td>13</td>
</tr>
</tbody>
</table>

**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.
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.

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.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>About This Guide</td>
<td>Provides a general introduction to this manual.</td>
</tr>
<tr>
<td>2</td>
<td>Tivoli Data Exchange Concepts</td>
<td>Provides an overview of the major components of Tivoli Data Exchange, its architecture, and how its components work.</td>
</tr>
<tr>
<td>3</td>
<td>Tivoli Data Exchange Interface Commands</td>
<td>Describes the commands used to submit and monitor data-transfer requests from a command-line interface.</td>
</tr>
<tr>
<td>4</td>
<td>Component Configuration Commands</td>
<td>Describes the commands used to start up Tivoli Data Exchange components.</td>
</tr>
<tr>
<td>5</td>
<td>C API Reference</td>
<td>Describes the Tivoli Data Exchange API, which allows you to submit and monitor data-transfer requests using API calls from within business applications.</td>
</tr>
<tr>
<td>6</td>
<td>C Data Structures</td>
<td>Describes the data structures used with the API functions.</td>
</tr>
<tr>
<td>7</td>
<td>COBOL API Reference</td>
<td>Describes the Tivoli Data Exchange COBOL API, which allows COBOL developers in an OS/390 or OS/390 environment to interface with the Tivoli Data Exchange C API from a COBOL application.</td>
</tr>
<tr>
<td>8</td>
<td>COBOL Data Structures</td>
<td>Lists the data structures that are used with the Tivoli Data Exchange COBOL API functions to handle input and output data.</td>
</tr>
<tr>
<td>9</td>
<td>Tivoli Data Exchange and MQSeries Security</td>
<td>Describes the security used by Tivoli Data Exchange and MQSeries.</td>
</tr>
<tr>
<td>10</td>
<td>XML Integration</td>
<td>Describes the characteristics of XML files that can be submitted to the Tivoli Data Exchange Manager to initiate Tivoli Data Exchange functions.</td>
</tr>
<tr>
<td>11</td>
<td>Multi-File Connector</td>
<td>Describes how the Multi-File Connector can be used to transfer files.</td>
</tr>
<tr>
<td>12</td>
<td>File to Message Connector</td>
<td>Describes how the File to Message connector can be used to transfer files to queues and queues messages to file.</td>
</tr>
<tr>
<td>13</td>
<td>Virtual Machine Connector</td>
<td>Describes the characteristics and how to use the Tivoli Data Exchange Virtual Machine Connector.</td>
</tr>
</tbody>
</table>
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 usib2hpdk@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:

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**

Sidebars contain 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.
About This Guide

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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tivoli Data Exchange Introduction</td>
<td>15</td>
</tr>
<tr>
<td>Why Tivoli Data Exchange?</td>
<td>15</td>
</tr>
<tr>
<td>Tivoli Data Exchange Architectural Overview</td>
<td>17</td>
</tr>
</tbody>
</table>

### Tivoli Data Exchange Introduction

Tivoli Data Exchange provides the means to exchange information between dissimilar networks, operating systems, databases, and applications. For developers who want to leverage the strengths of MQSeries, Tivoli Data Exchange is a high-performance solution for distributing mission-critical information throughout the enterprise. Surpassing simple data movement tools, Tivoli Data Exchange 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 Tivoli Data Exchange into existing business processes with minimal impact and/or change.

### Why Tivoli Data Exchange?

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.
Tivoli Data Exchange Architectural Overview

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

The following major components compose Tivoli Data Exchange:

- Tivoli Data Exchange Interfaces
- Tivoli Data Exchange Manager
- Tivoli Data Exchange Sender
- Tivoli Data Exchange Receiver
- Tivoli Data Exchange Status

The flow of a data transfer occurs as follows:

1. A request is submitted to Tivoli Data Exchange by one of the supported interfaces. The request is passed to the Tivoli Data Exchange Manager’s input queue.
2. After the Tivoli Data Exchange Manager accepts the request, but before processing it, the Tivoli Data Exchange Manager transaction preprocess exit can be called.

3. The Tivoli Data Exchange Manager submits the request to the Tivoli Data Exchange Sender via the Tivoli Data Exchange Sender's input queue.

4. After the Tivoli Data Exchange Sender accepts the request, but before processing the data being transferred, the Tivoli Data Exchange Sender can call the sender pre-process exit to perform application-specific processing.

5. The Tivoli Data Exchange Sender reads and transforms the data into MQSeries messages.

6. The messages that make up the data are submitted to the Tivoli Data Exchange Receiver via the Tivoli Data Exchange Receiver's input queue and data queues.

7. After processing the data, the Tivoli Data Exchange Sender can call the post-process exit to perform application-specific processing.

8. After the Tivoli Data Exchange Receiver accepts the data, but before processing it, the Tivoli Data Exchange Receiver can call the receiver pre-process exit to perform application-specific processing.

9. The Tivoli Data Exchange Receiver retrieves the data messages and processes the data accordingly.

10. After the Tivoli Data Exchange Receiver processes the data, it can call the Tivoli Data Exchange Receiver post-process exit to perform application-specific processing.

11. The Tivoli Data Exchange Manager receives all responses and ends the logical unit of work (LUW). Before ending the LUW, the Tivoli Data Exchange 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.

Tivoli Data Exchange Interfaces

The Tivoli Data Exchange interfaces communicate with the Tivoli Data Exchange subsystem. Tivoli Data Exchange currently supports the following interfaces:

- command-line interface
Tivoli Data Exchange Technical Reference

Tivoli Data Exchange Concepts

Tivoli Data Exchange Architectural Overview

- 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 Tivoli Data Exchange. The API places the input messages on the Tivoli Data Exchange input queue and receives replies from a predefined reply queue. The input queue is defined in the Tivoli Data Exchange configuration file.

Tivoli Data Exchange Manager

The Tivoli Data Exchange Manager reads its input queue, creates log entries, submits status messages to the status queues, and manages the state of all transfers. The Tivoli Data Exchange Manager starts the transfer unit of work and ends the transfer unit of work. The processing Tivoli Data Exchange Manager is always the originating queue manager (oqm). Tivoli Data Exchange requests are stored at the Tivoli Data Exchange Manager for processing. The node on which the Tivoli Data Exchange Manager executes is considered the originating node. The request is forwarded to the appropriate Tivoli Data Exchange Sender node as defined by the required input. The message is submitted to the Tivoli Data Exchange Sender's input queue, which is defined in the Tivoli Data Exchange configuration file. If the Tivoli Data Exchange 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 Tivoli Data Exchange Manager correlates all operational replies and reports the final status of the Tivoli Data Exchange 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 Tivoli Data Exchange processing. There are four completion codes that the Tivoli Data Exchange 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.
Tivoli Data Exchange Concepts

Tivoli Data Exchange Architectural Overview

Tivoli Data Exchange Sender

The Tivoli Data Exchange Sender reads its input queue, creates log entries, submits status messages, and transforms the data into MQSeries messages. The processing Tivoli Data Exchange Sender is always where the source data resides and is identified as the source Tivoli Data Exchange Sender or source queue manager (sqm). The Tivoli Data Exchange Sender has the capability to read the data and place it in a staging area or transmit it directly to the target Tivoli Data Exchange Receiver. The request, and all of its associated data, is forwarded to the target Tivoli Data Exchange Receiver as defined by the required input. If the Tivoli Data Exchange 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 Tivoli Data Exchange transaction, the Tivoli Data Exchange Sender replies to the originating Tivoli Data Exchange Manager.

Tivoli Data Exchange Receiver

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

Tivoli Data Exchange Status

The Tivoli Data Exchange status queues are defined in the Tivoli Data Exchange configuration file. Tivoli Data Exchange Status messages, which are submitted by each of the Tivoli Data Exchange components that make up the Tivoli Data Exchange subsystem, are not required for internal Tivoli Data Exchange 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.
Tivoli Data Exchange Concepts
Tivoli Data Exchange Architectural Overview

destined for the queue or list of queues defined in the Tivoli Data Exchange 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 Tivoli Data Exchange can operate without them.
Tivoli Data Exchange Concepts

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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>24</td>
</tr>
<tr>
<td>FTF</td>
<td>25</td>
</tr>
<tr>
<td>FTFCNCL</td>
<td>52</td>
</tr>
<tr>
<td>FTFPING</td>
<td>55</td>
</tr>
<tr>
<td>FTFSTAGE</td>
<td>64</td>
</tr>
<tr>
<td>Using Staging Queues</td>
<td>69</td>
</tr>
<tr>
<td>FTFSTAT</td>
<td>74</td>
</tr>
</tbody>
</table>

**Assumptions**

This chapter makes the following assumptions:

- You have a working knowledge of Tivoli Data Exchange 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
- Windows NT, 95, 98, and 2000
- OS/2
- OS/400
- OpenVMS
- IBM 4690 OS

Description

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

---

**Note:**

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

FTF

Syntax

FTF <Queue Manager arguments> <Source and Target File arguments> <Environment arguments> <Process arguments> <User Exit arguments> <Data Specification arguments> <AS/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)
- Process Arguments (see page 32)
- User Exit Arguments (see page 38)
- Data Specification Arguments (see page 39)
- AS/400 Option Arguments (see page 44)
- OS/390 Option Arguments (see page 47)
- Help Option Arguments (see page 50)

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 Tivoli Data Exchange 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 Tivoli Data Exchange Sender operates. If this value is not specified, the sqm is given the same value as the lqm.
Tivoli Data Exchange Interface Commands

FTF

- \texttt{dqm destQueueMgr} – Determines the queue manager where the Tivoli Data Exchange Receiver operates.

\underline{Note:}

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

\texttt{-dqm CLIENTA@QMGRA.}

\underline{Example}

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

\begin{verbatim}
FTF -lqm PROD16A -dqm PROD16A
\end{verbatim}

\underline{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:

\texttt{FTF -spath sourcePath -dpath destPath}

Where:

- \texttt{-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 \texttt{-fromstage} option] and an FTFID is specified
**Tivoli Data Exchange Interface Commands**

*FTF*

- **-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.

<table>
<thead>
<tr>
<th>File-Naming Conventions and OS/390 Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>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, Tivoli Data Exchange automatically allocates the PDS and creates the member.</td>
</tr>
</tbody>
</table>

**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 Windows NT/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.
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 TG1QMGR1, F:\D2.TXT on TG2QMGR2, and F:\D3.TXT on TG3QMGR3.

**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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points Tivoli Data Exchange 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange version to display:

```
FTF -version
```

**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:
**Tivoli Data Exchange Interface Commands**

*FTF*

FTF -cancelmode cancelMode -delsrc -expiry expDuration -id1 identifierText1 -id2 identifierText2 -id3 identifierText3 -immed -label labelVal -msgsize sizeVal -priority priorityVal -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 Tivoli Data Exchange configuration file. **Valid values:** ON, OFF

  If you specify -cancelmode ON, Tivoli Data Exchange 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, Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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.
**Tivoli Data Exchange Interface Commands**

**FTF**

- **-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 Tivoli Data Exchange data-transfer request that is processed synchronously between the Sender and Receiver rather than the normal asynchronous mode. Using the immediate option, the Tivoli Data Exchange 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 Tivoli Data Exchange data transfers require that the Receiver have all related data messages before writing the data to the disk. For example, using Tivoli Data Exchange 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**
**Tivoli Data Exchange Interface Commands**

*FTF*

- **-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 64).

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 Tivoli Data Exchange Sender, however, cannot purge or query the remotely staged files. These functions must be performed by the Tivoli Data Exchange Sender on the remote queue manager. If purge or query is attempted by the local Tivoli Data Exchange 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 Tivoli Data Exchange shutdown. If you select this argument, you increase the recovery ability of Tivoli Data Exchange, 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.
Tivoli Data Exchange Interface Commands

**FTF**

- **-tranpersist** – Specifies that the data being transferred is persistent. If you specify this argument, the data still exists after a system reboot or Tivoli Data Exchange shutdown. If you select this argument, you increase the recovery ability of Tivoli Data Exchange, 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 Tivoli Data Exchange 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 Tivoli Data Exchange Manager to indicate whether a request has succeeded or failed. If the Tivoli Data Exchange 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 Tivoli Data Exchange Receiver. Required command-line options not listed are specified in the options file.

```
FTF -ofile FTFMQ.FTF.OPT -stageonly
```
Tivoli Data Exchange Interface Commands

FTF

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.
Tivoli Data Exchange Interface Commands

FTF

-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:
Tivoli Data Exchange Interface Commands

FTF

- **compress** – Specifies that the data being sent is compressed using the internal Tivoli Data Exchange compression algorithm.

**Note:**
This compression utility is not supported on the 4690 platform.

- **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 Tivoli Data Exchange.

- **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 Tivoli Data Exchange 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
```
Tivoli Data Exchange Interface Commands

FTF

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

- **-mkdirs** – 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 an FILE OPEN ERROR.

- **-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:**

Tivoli Data Exchange 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.
Tivoli Data Exchange Interface Commands

*FTF*

- **-pool poolName** – Is the name of the data pool used for transferring data from the Tivoli Data Exchange Sender to the Tivoli Data Exchange Receiver. If this value is not specified, the default pool specified in the Tivoli Data Exchange configuration file is used. For this option to function, the specified pool must be defined in the Tivoli Data Exchange configuration file.

- **-recpad recordPad** – Enables or disables the padding facility. Tivoli Data Exchange 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**

Tivoli Data Exchange padding functionality is governed at the source node (the Tivoli Data Exchange Sender) and by the characteristics of the data being transferred. No padding decisions are made at the target node (the Tivoli Data Exchange 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 Tivoli Data Exchange configuration file, and the ISPF panels.
Tivoli Data Exchange Interface Commands

- **-recwrap** wrapMode – Indicates how records will be processed when they reach the target file and the records are longer than the target record length. Tivoli Data Exchange 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** 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 Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trusted</td>
<td>Yes</td>
</tr>
<tr>
<td>Compressed</td>
<td>Yes</td>
</tr>
<tr>
<td>Staged</td>
<td>Yes</td>
</tr>
<tr>
<td>Persistent Transfer Files</td>
<td>Yes</td>
</tr>
<tr>
<td>Persistent Staged Files</td>
<td>Yes</td>
</tr>
<tr>
<td>Message Size</td>
<td>3906</td>
</tr>
</tbody>
</table>
Tivoli Data Exchange Interface Commands

FTF

```
Priority | Setting
---|---
1 | 1

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
```

In this example, data is sent from an AS/400 physical file to another AS/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=,”
```

AS/400 Arguments

This section describes the FTF 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 Tivoli Data Exchange configuration file. You can use multiple queue managers. Each instance of Tivoli Data Exchange must be installed against a single queue manager, but you can install multiple instances of Tivoli Data Exchange against different queue managers.

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

```
FTF -as400ft fileType -ccsid codedCharSetId -crtlib createLib -fileasp fileAuxStoragePool -filetxt fileText -libasp lib/AuxStoragePool -libtxt libText
-redlen recordLength
```

Where:
Tivoli Data Exchange Interface Commands

FTF

- **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 AS/400 Save file) *SRCFP* (specifies an AS/400 source physical file).

- **crtlib createLib** – Specifies that Tivoli Data Exchange 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, Tivoli Data Exchange uses the CCSID of the job. **Valid values:** 1-65535

- **fileasp fileAuxStoragePool** – Specifies the Library Auxiliary Pool for a file that Tivoli Data Exchange creates for a data-transfer request. **Valid values:** 1-16

- **filetxt fileText** – Specifies the file description for a library that Tivoli Data Exchange creates for a data-transfer request.

- **libasp libAuxStoragePool** – Specifies the Library Auxiliary Pool for a library that Tivoli Data Exchange creates for a data-transfer request. **Valid values:** 1-16

- **libtxt libText** – Specifies the library description for a library that Tivoli Data Exchange creates for a data-transfer request.

- **rcdlen recordLength** – Determines the record length for the target file on OS/390. **Valid values:** 13-32766

---

**Note:**

On the AS/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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS/400 File Type</td>
<td>*DFLT</td>
</tr>
<tr>
<td>Create Library</td>
<td>YES</td>
</tr>
<tr>
<td>Library Aux Storage Pool</td>
<td>1</td>
</tr>
<tr>
<td>File Aux Storage Pool</td>
<td>1</td>
</tr>
<tr>
<td>Library Text</td>
<td>TARGET LIBRARY</td>
</tr>
<tr>
<td>File Text</td>
<td>TARGET FILE</td>
</tr>
<tr>
<td>CCSID</td>
<td>129</td>
</tr>
<tr>
<td>Record Length</td>
<td>255</td>
</tr>
</tbody>
</table>

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 Tivoli Data Exchange 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 Tivoli Data Exchange configuration file.
- Specify the unit value in the -unit argument or in the MVSUNITNAME stanza in the Tivoli Data Exchange configuration file on either the Tivoli Data Exchange Sender or the Tivoli Data Exchange Receiver.

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

```
FTF -alcunit allocUnit -blksize blockSize -bufno numberBuffers -dirblk 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
Tivoli Data Exchange Interface Commands

FTF

- **-bufno numberBuffers** – Allows you to specify the number of internal buffers that are to be used when processing data transfers. The throughput of a Tivoli Data Exchange 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.

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

- **-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:**

Tivoli Data Exchange 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.

- `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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Organization</td>
<td>Physical Sequential</td>
</tr>
<tr>
<td>Record Format</td>
<td>Fixed Block</td>
</tr>
<tr>
<td>Logical Record Length</td>
<td>255</td>
</tr>
<tr>
<td>Block Size</td>
<td>25,500</td>
</tr>
<tr>
<td>Unit Name</td>
<td>SYSALLDA</td>
</tr>
</tbody>
</table>
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 Tivoli Data Exchange arguments used with the FTF command. It contains syntax information and examples.

The FTF command with standard Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF -h</td>
</tr>
</tbody>
</table>
Supported Operating Systems

- OS/390
- UNIX
- Windows NT, 95, 98 or 2000
- OS/2
- OS/400
- OpenVMS

Description

The FTFCNCL command cancels the specified data-transfer request. Each Tivoli Data Exchange component can receive and cancel a data-transfer request at any time before or during the data-transfer transaction. When the appropriate Tivoli Data Exchange 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 Tivoli Data Exchange 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.
**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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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.

- **-cq configQueueName** – Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points Tivoli Data Exchange 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 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 Tivoli Data Exchange Manager operates. If this value is not specified, the oqm is given the same value as the lqm.

- **-version** – Returns the current Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.
Tivoli Data Exchange Interface Commands

FTFCNCL

- **-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 Tivoli Data Exchange transaction by providing a Tivoli Data Exchange transaction identifier which correlates to the transaction that is to be cancelled. If the transaction is not found, the Tivoli Data Exchange 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 Tivoli Data Exchange Manager can communicate with the source Tivoli Data Exchange Sender or with the target Tivoli Data Exchange Receiver, 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 Tivoli Data Exchange Sender or the Tivoli Data Exchange Receiver) stops the transfer’s execution. The Tivoli Data Exchange Manager immediately flags the Tivoli Data Exchange transaction as cancelled, but waits for a reply from both the Tivoli Data Exchange Sender and/or Tivoli Data Exchange 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 Tivoli Data Exchange 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
- Windows NT, 95, 98 or 2000
- OS/2
- OS/400
- IBM 4690 OS
- OpenVMS

Description

The FTFPING command allows you to ping specified Tivoli Data Exchange components. The ping message starts at the lqm, the machine from which you are currently working. It is sent to the Tivoli Data Exchange Manager, as identified by the oqm. The Tivoli Data Exchange Manager sends it to the Tivoli Data Exchange Sender, as identified by the sqm. The Tivoli Data Exchange Sender sends it to the Tivoli Data Exchange Receiver, as identified by the dqm. The Tivoli Data Exchange Receiver then sends an acknowledgment to the Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange Receiver operates.

Example

The following FTFPING command is issued from the queue manager called ADMIN1. The Tivoli Data Exchange Manager running on PROD11A is pinged. If it is running, it sends the ping to the Tivoli Data Exchange Sender running on PROD22. If it is running, it sends the ping to the Tivoli Data Exchange 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 Tivoli Data Exchange environment arguments takes the following form:
Tivoli Data Exchange Interface Commands

FTFPING

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

Where:

- **-cfile configFile** – Can contain the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points Tivoli Data Exchange 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.

- **-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 Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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 Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.
Tivoli Data Exchange Interface Commands

**FTFPING**

**Examples**

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

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

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

```
FTFPING -version
```

**Help Arguments**

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

The FTFPING command with standard Tivoli Data Exchange 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
```
Tivoli Data Exchange Interface Commands

FTFPING

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 Tivoli Data Exchange 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 Tivoli Data Exchange Manager, Tivoli Data Exchange Sender, and Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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:
Tivoli Data Exchange Interface Commands

**FTFPING**

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 Windows NT 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 Tivoli Data Exchange and the channel definitions required for communication between the nodes.

   The FTF.MQS file creates the queues needed by the Tivoli Data Exchange 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 Windows NT, 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 Windows NT, 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 Windows NT, 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 Windows NT. 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 Tivoli Data Exchange components are running. Check that the correct Tivoli Data Exchange 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 Tivoli Data Exchange Sender queue manager node, the response to ftfdr -lqm qMGrName should be “Tivoli Data Exchange Sender initialization complete”.

On the Tivoli Data Exchange Receiver node, the response to ftfrcy -lqm qMGrName should be “Tivoli Data Exchange Receiver initialization complete”.

On the Tivoli Data Exchange Manager node, the response to ftfmgr -lqm qMGrName should be “Tivoli Data Exchange 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
Tivoli Data Exchange Interface Commands

FTFSTAGE

Supported Operating Systems

- OS/390
- UNIX
- Windows NT, 95, 98, or 2000
- OS/2
- 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.
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)
**Tivoli Data Exchange Interface Commands**

**FTFSTAGE**

**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 Tivoli Data Exchange environment arguments takes the following form:

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

Where:

- **-cfile config_file** – Can contain the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange 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.

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 Tivoli Data Exchange 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 Tivoli Data Exchange 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.

Tivoli Data Exchange 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 Tivoli Data Exchange Receiver, you send it to the stage control queue specified in the Tivoli Data Exchange 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.

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
```
Tivoli Data Exchange Interface Commands
Using Staging Queues

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\reptww01.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 FTFSSTAGE 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 71.)
**Using the SenderAlwaysCheckStage Property**

The Tivoli Data Exchange configuration file’s `SenderAlwaysCheckStage` property, if enabled, directs Tivoli Data Exchange 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, Tivoli Data Exchange looks for it at a specified location.

To enable the `SenderAlwaysCheckStage` property, open the Tivoli Data Exchange configuration file, set the property value to ON, then save, close the file, and restart Tivoli Data Exchange.

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 Tivoli Data Exchange 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.

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

The `-wait` option tells Tivoli Data Exchange how long to wait for query output. It is specified in seconds. Default value is 300 seconds (5 minutes).
The following figure displays output from the query argument used with FTFSTAGE. It lists information about each of the files in the staging queue.

<table>
<thead>
<tr>
<th>FTF Id</th>
<th>filename</th>
<th>Date/Time Staged</th>
<th>QMgr</th>
<th>Message Size</th>
<th>Number of messages</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>de675301-8eaa-11d2-af00-ac2f955ce9c8</td>
<td>c:\images\bigfile.avi</td>
<td>12/09/1998 09:32:52</td>
<td>QMGRA</td>
<td>524298</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>e0da1a1-8eaa-11d2-86e6-c3ecf04f68dc</td>
<td>c:\temp\assets.txt</td>
<td>12/09/1998 09:32:54</td>
<td>QMGRA</td>
<td>524298</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>e3574e1a-8eaa-11d2-b371-8e1c4cc0f27f</td>
<td>d:\forwarding\invoice.dat</td>
<td>12/09/1998 09:32:56</td>
<td>QMGRA</td>
<td>524298</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>e6c60f91-8eaa-11d2-9d82-9dd888d7c8b5</td>
<td>d:\images\cover.jpg</td>
<td>12/09/1998 09:33:02</td>
<td>QMGRA</td>
<td>524298</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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
```
Tivoli Data Exchange Interface Commands

FTFSTAT

Supported Operating Systems

- OS/390
- UNIX
- Windows NT, 95, 98, or 2000
- OS/2
- 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 dateVal -edate dateVal -dpath destPath -dqm destQueueMgr -ftfid ftfIdVal -i -label labelVal -oqm origQueueMgr -rqm reqQueueMgr -spath sourcePath -sqm sourceQueueMgr -status statusType
```

Where:
Tivoli Data Exchange Interface Commands

FTFSTAT

- **-sdate** *dateTimeVal* – Determines the earliest start date and time for which status information is returned. If you specify the beginning date, but no ending, Tivoli Data Exchange 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, Tivoli Data Exchange 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.
Tivoli Data Exchange Interface Commands

FTFSTAT

- **-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.

- **-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:** Active, Cancelled, Complete, Expired, Failed

**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.

```plaintext
FTFSTAT -spath FTFMQ.SOURCE.RPT1016 -dpath FTFMQ.SOURCE.RPT1016 -i -label “report transfer” -date 19991204 +date 19991206 -status Complete
```

**Standard Environment Arguments**

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

The FTFSTAT command with standard Tivoli Data Exchange environment arguments takes the following form:

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

Where:

- **-cfile** configFile – Can contain the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no cfile value is specified, this value must be specified. The cq argument points Tivoli Data Exchange 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 – 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 Tivoli Data Exchange 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.HTFCONFIG.INI -ofile FTFMQ.HTF.OPT
```

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

```
FTFSTAT -version
```
Tivoli Data Exchange Interface Commands

FTFSTAT

Help Arguments

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

The FTFSTAT command with Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange configuration file so MQSeries transfers all information to a single queue manager. This approach allows the Tivoli Data Exchange 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 Tivoli Data Exchange user label; otherwise the FTFID value is listed.

**Input**

```plaintext
>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 Tivoli Data Exchange 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.

```
   c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf  
   (TESTNT2):C:\ftfdev\bin\testFiles\smtxt01.txt  
   (TESTNT3):C:\FTFMQ\TempFiles\FTFbeta2-T1-1869.txt

   c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf  
   (TESTNT2):C:\ftfdev\bin\testFiles\smtxt01.txt  
   (TESTNT3):C:\FTFMQ\TempFiles\FTFbeta2-T1-1860.txt

   - 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

   - 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

97250471-c3cb-11d4-bea5-f7b36290e858
  (TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
  (TESTNT2)C:\FTFMQ\TempFiles\FTFbeta2-T1-3211.txt

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 Tivoli Data Exchange 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
Request QMgr : TESTNT2

Originating QMgr : TESTNT2
Status : REQUEST_FAILED
Error : 630 (637) : Receiver reported a failure

Source Qmgr : TESTNT2
Source File Name : C:\ftfdev\bin\testFiles\smtxt01.txt
Status : REQUEST_TRANSMITTED
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

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 Tivoli Data Exchange component within a given node.

Input

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

Output

```
FTF Identifier : ecbcccb1-e4fd-11d1-9bbb-8e4e2be42fed
Request QMgr : TESTNT2

Originating QMgr : TESTNT2
Status : REQUEST_FAILED
Error : 630 (637) : Receiver reported a failure

Source Qmgr : TESTNT2
Source File Name : C:\ftfdev\bin\testFiles\smtxt01.txt
Status : REQUEST_TRANSMITTED
```
**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 ------**

  - **QMgr**: TESTNT2
  - **Component**: FTFREQ_COMMAND_LINE
  - **Status**: REQUEST_SUBMITTED

  - **QMgr**: TESTNT2
  - **Component**: FTF_MANAGER
  - **Status**: REQUEST_SUBMITTED

  - **QMgr**: TESTNT2
  - **Component**: FTF_SENDER
  - **Status**: REQUEST_RECEIVED

  - **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
Tivoli Data Exchange Interface Commands

FTFSTAT

Input

>ftfstat -lqm TESTAIX41 -status COMP

Output

- 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

- 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

Sample 4

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

Input

>ftfstat -lqm TESTAIX41 -status COMP
Tivoli Data Exchange Interface Commands

FTFSTAT

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

c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-2992.txt

c6cd2ea1-c3cb-11d4-b480-da40b1a97ddf
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-0031.txt

97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-5431.txt

97250471-c3cb-11d4-bea5-f7b36290e858
(TESTNT2)C:\ftfdev\bin\testFiles\smtxt01.txt
(TESTAIX41)/tmp/FTFbeta2-T1-4432.txt
This chapter describes the commands that start and configure Tivoli Data Exchange components. It contains the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>88</td>
</tr>
<tr>
<td>FTFCFG</td>
<td>89</td>
</tr>
<tr>
<td>FTFEND</td>
<td>93</td>
</tr>
<tr>
<td>FTFLOG</td>
<td>99</td>
</tr>
<tr>
<td>FTFMGR</td>
<td>107</td>
</tr>
<tr>
<td>TFRCV</td>
<td>115</td>
</tr>
<tr>
<td>FTFSDR</td>
<td>122</td>
</tr>
<tr>
<td>FTFSTART</td>
<td>129</td>
</tr>
</tbody>
</table>

**Assumptions**

This chapter makes the following assumptions:

- You have a working knowledge of Tivoli Data Exchange components and their interrelationships.

- You know how to access and use a command line in the appropriate operating system.
**Component Configuration Commands**

*Overview*

The commands in this chapter allow you to start, configure, and shut down specified Tivoli Data Exchange components or to start the entire Tivoli Data Exchange 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 Tivoli Data Exchange components:

- FTFCFG (see page 89)
- FTFEND (see page 93)
- FTFLOG (see page 99)
- FTFMGR (see page 107)
- FTFRVC (see page 115)
- FTFSDR (see page 122)
- FTFSTART (see page 129)
Supported Operating Systems

- OS/390
- UNIX
- Windows NT, 95, 98, and 2000
- OS/2
- OS/400
- OpenVMS

Description

To accommodate interfaces that process information in queues rather than files, the FTFCFG command populates a queue with the specified Tivoli Data Exchange configuration 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange configuration file to be loaded into the configuration queue. The configuration queue to be populated is defined in the configuration file, FTFConfigQueue.
Component Configuration Commands

FTFCFG

- **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 Tivoli Data Exchange 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, FTFCONFIG, into the configuration queue defined in FTFCONFIG. The node PRD1MQM is the name of the queue manager where the configuration queue resides.

```
FTFCFG -lqm PRD1MQM -node PRD1MQM -cfile FTFCONFIG
         -ofile FTFMQ.FTF.OPT
```

The following FTFCFG command displays the version, release, and patch number for the current Tivoli Data Exchange 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:

```
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
- Windows NT
- OS/2
- OS/400
- IBM 4690 OS
- OpenVMS

Description

The FTFEND command shuts down specified Tivoli Data Exchange components. The components do not process the shutdown message while they are processing a data-transfer request. For example, if the Tivoli Data Exchange 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>
Component Configuration Commands

FTFEND

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 Tivoli Data Exchange 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 local queue manager (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] -immediate -quiesce -timeout expireVal
```
Component Configuration Commands

FTFEND

Where:

- \([-\text{cpt } \text{compNum}] [-\text{component } \text{compNum}]\) – Determines the type of component being shut down. You should use either the -cpt argument or the -component argument. You can only shut down one type of component using this argument. **Valid values:**
  1 – Tivoli Data Exchange Manager
  2 – Tivoli Data Exchange Sender
  3 – Tivoli Data Exchange Receiver
  4 – Tivoli Data Exchange logging daemon
  5 – Tivoli Data Exchange status daemon

**Default value:** If nothing is specified, then all components are shut down.

- \(-\text{immediate}\) – Shuts the component down immediately, before completing any current work. If you specify this argument, you cannot specify the -quiesce argument.

- \(-\text{quiesce}\) – Shuts the component down after its work is completed. If you specify this argument, you cannot specify the -immediate argument.

- \(-\text{timeout } \text{expireVal}\) – 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 Tivoli Data Exchange 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 Tivoli Data Exchange environment arguments used with the FTFEND command. It contains syntax information and examples.
Component Configuration Commands

FTFEND

The FTFEND command with standard Tivoli Data Exchange environment arguments takes the following form:

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

Where:

- **-cfile configFile** – Can contain the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points Tivoli Data Exchange 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** – Contains 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 Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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.
Component Configuration Commands

**FTFEND**

**Examples**

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

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

The following FTFEND command displays the version, release, and patch number for the current Tivoli Data Exchange 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`, or `-?` – 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
```
**Component Configuration Commands**

*FTFEND*

**Additional Information**

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

---

**Note:**

When you submit an FTFEND request to shut down Tivoli Data Exchange component(s) on a local queue manager, a shutdown message is placed on the queue defined as the FTFLogQueue in the Tivoli Data Exchange 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.

---
Component Configuration Commands

FTFLOG

Supported Operating Systems

- OS/390
- UNIX
- Windows NT
- OS/2
- 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 Tivoli Data Exchange arguments takes the following form:

`FTFLOG -lq localQueueMgr -ld 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.

- `logfile 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.

**Note:**

On OS/2 platforms, if you are using the FAT file system, the log file must conform to an 8.3 file-naming convention, with a filename no longer than eight characters and an extension no longer than three letters.

- `-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 Tivoli Data Exchange configuration file.

<table>
<thead>
<tr>
<th>Specifying an Esoteric Unit Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>To specify an esoteric name for the OS/390 UNIT value, follow these steps:</td>
</tr>
<tr>
<td>• Do not set a value in the MVSVOLUME stanza in the Tivoli Data Exchange configuration file.</td>
</tr>
<tr>
<td>• Specify the Unit value in the -unit argument or in the MVSUNITNAME stanza in the Tivoli Data Exchange configuration file on either the Tivoli Data Exchange Sender or the Tivoli Data Exchange Receiver</td>
</tr>
</tbody>
</table>

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.
Component Configuration Commands

FTFLOG

- **-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 Tivoli Data Exchange component(s) on a local queue manager, a shutdown message is placed on the queue defined as the FTFLogQueue in the Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Organization</td>
<td>Physical Sequential</td>
</tr>
<tr>
<td>Record Format</td>
<td>Fixed Block</td>
</tr>
<tr>
<td>Logical Record Length</td>
<td>255</td>
</tr>
<tr>
<td>Block Size</td>
<td>65,535</td>
</tr>
<tr>
<td>Unit Name</td>
<td>SYSALLDA</td>
</tr>
</tbody>
</table>
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.

```plaintext
FTFLOG -ofile FTFMQ.FTF.OPT -org PS -recfmt FB
   -lrecl 255 -blksz 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 arguments takes the following form:

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

Where:

- `-cfile configFileName` – Contains the fully qualified path and filename for the Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume Serial Number</td>
<td>TECH01</td>
</tr>
<tr>
<td>Allocation Unit</td>
<td>TRK</td>
</tr>
<tr>
<td>Primary Allocation</td>
<td>15</td>
</tr>
<tr>
<td>Secondary Allocation</td>
<td>30</td>
</tr>
</tbody>
</table>
**Component Configuration Commands**

**FTFLOG**

- **-cq configQueueName** - Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points Tivoli Data Exchange 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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 Tivoli Data Exchange 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:

```plaintext
FTFLogQueue=PROD1_2.QLOG
```

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.
Component Configuration Commands

FTFLOG

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 Tivoli Data Exchange 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 Tivoli Data Exchange arguments takes the following form:

```
FTFLOG [-help | -h | -?]
```
Component Configuration Commands

FTFLOG

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.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFLOG -h</td>
</tr>
</tbody>
</table>
FTFMGR

Supported Operating Systems

- OS/390
- UNIX
- Windows NT
- OS/2
- OS/400
- OpenVMS
- IBM 4690 OS

Description

The FTFMGR command starts a Tivoli Data Exchange 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-nodename nodeName
Component Configuration Commands

FTFMGR

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.

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

- **-nodeName** *nodeName* – Associates the node name with the instance of Tivoli Data Exchange components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.

**Examples**

In the following example, the Tivoli Data Exchange Manager is started on PROD11A. Its configuration file is CONFIGMQ.

```
FTFMGR -lqm PROD11A -cfile CONFIGMQ -nodeName shadow
```
In the following example, the Tivoli Data Exchange 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/ftfmr.log >/dev/null&
```

## Standard Environment Arguments

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

The FTFMGR command with standard Tivoli Data Exchange 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 Tivoli Data Exchange 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.
Component Configuration Commands

**FTFMGR**

- **-cq configQueueName** – Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points Tivoli Data Exchange 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.

- **-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. Tivoli Data Exchange 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 Tivoli Data Exchange 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
```
Component Configuration Commands

FTFMGR

The FTFMGR command displays the version, release, and patch number for the current Tivoli Data Exchange software.

| FTFMGR  -version |

Help Arguments

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

The FTFMGR command with standard Tivoli Data Exchange 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 |
The startup parameters for the Tivoli Data Exchange Sender, Tivoli Data Exchange Manager, and Tivoli Data Exchange 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 Tivoli Data Exchange components on CLIENTA and CLIENTB must be defined in the Tivoli Data Exchange configuration 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 Tivoli Data Exchange prior to V1.2 do not support this configuration. Use FTFNodeOverride to define queue managers and client nodes in previous versions of Tivoli Data Exchange. In FTFNodeOverride, you must define each client separately.
To start the Tivoli Data Exchange 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
```
If you are using a version of Tivoli Data Exchange 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 Tivoli Data Exchange 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.FTRCV.CONTROL
  ReceiverSyncQueue=CLIENTA.FTRCV.SYNC
  ReceiverStageQueue=CLIENTA.FTRCV.STAGE
  ReceiverSystemQueue=CLIENTA.FTRCV.SYSTEM
  ReceiverCancel=NO

CLIENTB:
  QueueManager=QMGRA
  ReceiverNumInstances=1
  ReceiverControlQueue=CLIENTB.FTRCV.CONTROL
  ReceiverSyncQueue=CLIENTB.FTRCV.SYNC
  ReceiverStageQueue=CLIENTB.FTRCV.STAGE
  ReceiverSystemQueue=CLIENTB.FTRCV.SYSTEM
  ReceiverCancel=NO
```
Component Configuration Commands

FTFRCV

Supported Operating Systems

- OS/390
- UNIX
- Windows NT, 2000, 95, and 98
- OS/2
- OS/400
- IBM 4690 OS
- OpenVMS

Description

The FTFRCV command starts the Tivoli Data Exchange 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 -nodename nodeName -sync queueName
Component Configuration Commands
FTFRCV

Where:

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

- `-lqm localQueueMgr` – Determines the queue manager from which the FTFRCV command is issued. This value is required on OS/390 systems. On all other systems, if it is not specified, the FTFRCV 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 Tivoli Data Exchange components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.

- `-sync queueName` – Determines the sync queue used by the Tivoli Data Exchange Receiver. The Tivoli Data Exchange 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 follows:

```
SenderSyncQueue=FTFSDR.SYNC
```
Component Configuration Commands

FTFRCV

A valid sync queueName can be FTFSDR.SYNC.1 or FTFSDR.SYNC.2. The queueName is valid as long as the 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 FTFRCV command starts the Tivoli Data Exchange Receiver on PROD11A.

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

In the following example, the Tivoli Data Exchange 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**

- `-ftfauth` – Indicates that a security authorization exit is being used on the Tivoli Data Exchange Receiver.

**Example**

The following FTFRCV command invokes a security authorization exit for the specified Tivoli Data Exchange Receiver.

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

**Standard Environment Arguments**

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

The FTFRCV command with standard Tivoli Data Exchange 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 Tivoli Data Exchange 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.
Component Configuration Commands

FTFRCV

- **cq** *configQueueName* – Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points Tivoli Data Exchange 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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. Tivoli Data Exchange 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 Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version information.

**Examples**

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

```
FTFRCV -sync SYNC1 -ofile FTFOPTS
```
Component Configuration Commands

**FTFRCV**

The following FTFRCV command displays the version, release, and patch number for the current Tivoli Data Exchange 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 -connectorentry entryPointName -connectordata dataString
```

Where:
Component Configuration Commands

FTFRCV

- **-connectordll ***dllName* – The name of the DLL, shared object, or load module used to invoke the connector module.

- **-connectorentry ***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 Tivoli Data Exchange Receiver to the connector. This value is optional.

Example

In the following example, the Tivoli Data Exchange 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 -connectorentry receiveDailyReport
```
**Component Configuration Commands**

**FTFSDR**

**Supported Operating Systems**

- OS/390
- UNIX
- Windows NT, 95, 98, and 2000
- OS/2
- OS/400
- OpenVMS
- IBM 4690 OS

**Description**

The FTFSDR command starts the Tivoli Data Exchange 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 -Iqm localQueueMgr -nodename nodeName -lfile logFile
-sync queueName
```
Component Configuration Commands

FTFSDR

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 Tivoli Data Exchange components. The nodeName must be defined in the configuration file as FTFNodeAlias or FTFNodeOverride.

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

Notes:

- On OS/2 platforms, if you are using the FA T file system, the log file must conform to an 8.3 file-naming convention, with a filename no longer than eight characters and an extension no longer than three letters.

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

- **-sync queueName** – Determines the sync queue used by the Tivoli Data Exchange Sender. The Tivoli Data Exchange 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 Tivoli Data Exchange Sender on PROD11A.

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

In the following example, the Tivoli Data Exchange 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.
Component Configuration Commands

FTFSDR

- `ftfauth` – Indicates that a security authorization exit is being used on the Tivoli Data Exchange Sender.

Example

The following FTFSDR command invokes a security authorization exit for the specified Tivoli Data Exchange 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 Tivoli Data Exchange 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.
Component Configuration Commands

**FTFSDR**

- `-cq configQueueName` – Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no `-cfile` value is specified, this value must be specified. The `-cq` argument points Tivoli Data Exchange 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.

- `-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. Tivoli Data Exchange 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 Tivoli Data Exchange version, release, and patch number. Any command entered with this argument ignores all other arguments and returns the version 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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.

**Examples**

The following FTFSDR command starts the Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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:
Component Configuration Commands

*FTFSDR*

- `-connectordll dllName` – The name of the DLL, shared object, load module, or service program used to invoke the connector module.

- `-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 Tivoli Data Exchange Sender to the connector. This value is optional.

**Example**

The following FTFSDR command starts the Tivoli Data Exchange Sender on queue manager PROD12A. It connector uses for the entry point of `sendDailyReport` in the DLL `sendrept.dll`.

```
FTFSDR -1qm PROD12A -connector -connectordll sendrept.dll -connectorentry sendDailyReport
```
Component Configuration Commands

FTFSTART

Supported Operating Systems

- OS/390

Description

The FTFSTART command starts all Tivoli Data Exchange components identified in the Tivoli Data Exchange 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 Tivoli Data Exchange Sender and the Tivoli Data Exchange 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 Tivoli Data Exchange Sender’s authorization module.
- **-exitentry** `sEntryPoint` – Contains the Tivoli Data Exchange 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 Tivoli Data Exchange Receiver’s authorization module.

- **-exitentry** `rEntryPoint` – Contains the Tivoli Data Exchange Receiver’s entry point to the DLL.

**Example**

The following FTFSTART command installs the sample security authorization provided with Tivoli Data Exchange.

```
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 Tivoli Data Exchange 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.
**Component Configuration Commands**

**FTFSTART**

### Example

The following FTFSTART command sets the sysoutclass character to H.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSTART -sysoutclass H</td>
</tr>
</tbody>
</table>

### Startup Arguments

Startup arguments allow you to specify the local queue manager and the Tivoli Data Exchange configuration file for the components being started by FTFSTART. The FTFSTART command with standard Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no -cfile value is specified, this value must be specified. The -cq argument points Tivoli Data Exchange 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, Tivoli Data Exchange checks the FTF_CONFIG_QUEUE environment variable and uses the specified queue. If this environment variable is not set, Tivoli Data Exchange 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 Tivoli Data Exchange environment arguments used with the FTFSTART command. It contains syntax information and an example.

The FTFSTART command with standard Tivoli Data Exchange environment arguments takes the following form:

```
FTFSTART -ofile optionsFile -version [-help | -h | -?]```
Component Configuration Commands

**FTFSTART**

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 Tivoli Data Exchange 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 Tivoli Data Exchange software.

```
FTFSTART -version
```

The following FTFSTART command displays its command-line options.

```
FTFSTART -h
```
This chapter describes the Tivoli Data Exchange C API, which allows you to leverage Tivoli Data Exchange 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:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFCancel</td>
<td>137</td>
</tr>
<tr>
<td>FTFPing</td>
<td>139</td>
</tr>
<tr>
<td>FTFReq</td>
<td>141</td>
</tr>
<tr>
<td>FTFShutdown</td>
<td>143</td>
</tr>
<tr>
<td>FTFStage</td>
<td>145</td>
</tr>
<tr>
<td>FTFStatusDelete</td>
<td>147</td>
</tr>
<tr>
<td>FTFStatusFreeDetailList</td>
<td>149</td>
</tr>
<tr>
<td>FTFStatusFreeSummaryList</td>
<td>151</td>
</tr>
<tr>
<td>FTFStatusGetDetailList</td>
<td>153</td>
</tr>
<tr>
<td>FTFStatusGetSummaryList</td>
<td>155</td>
</tr>
<tr>
<td>FTFSubmitStatusMsg</td>
<td>158</td>
</tr>
</tbody>
</table>
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 Tivoli Data Exchange subsystem. The cancel request must be submitted to the same Tivoli Data Exchange 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 Tivoli Data Exchange status messages.

**General Prototype**

```c
#include "ftfc.h"

FTFVOID FTFCancel(MQHCONN hQM,
   FTFCancelInfo *pCancelInfo,
   FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"

FTFVOID FTFCAN(MQHCONN hQM,
   FTFCancelInfo *pCancelInfo,
   FTFCA *pftfca);
```
C API Reference
FTFCancel

Parameters

The following table lists and describes the FTFCancel API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN)</td>
<td>Connection handle</td>
<td>Represents the connection to the queue manager. The value of MQHconn was</td>
</tr>
<tr>
<td>- input</td>
<td></td>
<td>returned by a previous MQCONN call.</td>
</tr>
<tr>
<td>pcancelInfo</td>
<td>Pointer to</td>
<td>Represents a properly formatted data structure used by the FTFCancel API.</td>
</tr>
<tr>
<td>(FTFCancelInfo *)</td>
<td>FTFCancelInfo data</td>
<td>This data structure contains all of the attributes and input criteria to</td>
</tr>
<tr>
<td>-input</td>
<td>structure</td>
<td>cancel a request that has already been submitted to the Tivoli Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exchange subsystem.</td>
</tr>
<tr>
<td>pftfca (FTFCA *)</td>
<td>Pointer to Tivoli</td>
<td>Represents an output area that returns properly formatted API reply</td>
</tr>
<tr>
<td>- output</td>
<td>Data Exchange reply</td>
<td>information. This information includes the return codes and optional</td>
</tr>
<tr>
<td></td>
<td>message data structure</td>
<td>messages.</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTCA (page 179)
- FTFCancelInfo (page 181)


**FTFPing**

**Description**

This API allows you to ping specified the Tivoli Data Exchange components. The ping message starts at the lqm, the machine from which you are currently working. It is sent to the Tivoli Data Exchange Manager, as identified by the oqm. The Tivoli Data Exchange Manager sends it to the Tivoli Data Exchange Sender, as identified by the sqm. The Tivoli Data Exchange Sender sends it to the Tivoli Data Exchange Receiver, as identified by the dqm. The Tivoli Data Exchange Receiver then sends an acknowledgment to the Tivoli Data Exchange Manager.

The FTFPing API allows you to ensure that all Tivoli Data Exchange 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**

```c
#include "ftfc.h"

FTFVOID FTFPing(MQHCONN hQM,
FTFPingInfo *pPingInfo,
FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"

FTFVOID FTFPING(MQHCONN hQM);
FTFPingInfo *pPingInfo,
FTFCA *pftfca);
```
### Parameters

The following table lists and describes the FTFPing API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN)</td>
<td>Connection handle</td>
<td>Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.</td>
</tr>
<tr>
<td></td>
<td>- input</td>
<td></td>
</tr>
<tr>
<td>pPingInfo (FTFPingInfo *)</td>
<td>Pointer to a PingInfo data structure</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>-input</td>
<td></td>
</tr>
<tr>
<td>ptfca (FTFCA *)</td>
<td>Pointer to a Tivoli Data Exchange reply message data structure</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
<tr>
<td></td>
<td>- output</td>
<td></td>
</tr>
</tbody>
</table>

**Valid Values** - A valid FTFCA pointer, null.

### Related Data Structures

- FTFCA (page 179)
- FTFPingInfo (page 215)
FTFReq

Description

This API generates a Tivoli Data Exchange 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 Tivoli Data Exchange does not otherwise allow.
- Specifying a queue pool to make transfers more effective.

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 Tivoli Data Exchange subsystem before returning control to the application.

Note:

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

General Prototype

```c
#include “ftfc.h”

FTFVOID FTFReq(MQHCONN hQM,
FTFRequestMsgInfo *pRequestInfo,
```
C API Reference

FTFReq

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.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN)</td>
<td>- input</td>
<td>Connection Handle</td>
</tr>
<tr>
<td>pRequest</td>
<td>(FTFRequestMsgInfo *) - input</td>
<td>Pointer to a Request Message data structure</td>
</tr>
<tr>
<td>pftfca</td>
<td>(FTFCA *) - output</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTFAS400FileInfo (page 177)
- FTFCA (page 179)
- FTFExitInfo (page 189)
- FTFJobInfo (page 212)
- FTFQMgrsInfo (page 218)
- FTFRequestMsgInfo (page 220)
- FTFSrcFileInfo (page 228)
- FTFTargetFileInfo (page 263)
- FTFUserInfo (page 268)
**FTFShutdown**

**Description**

This API shuts down specified Tivoli Data Exchange components. A shutdown request is submitted and can target all components or a specific component (Tivoli Data Exchange Manager, Tivoli Data Exchange Sender, or Tivoli Data Exchange 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 Tivoli Data Exchange subsystem.

The Tivoli Data Exchange components being shut down do not process the shutdown message while they are processing a data-transfer request. For example, if the Tivoli Data Exchange 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 Tivoli Data Exchange Sender by other means may cause the failure of the data-transfer request.

**General Prototype**

```c
#include "ftfc.h"

FTFVOID FTFShutdown(MQHCONN hQM,
                    FTFShutdownInfo *pShutdownInfo,
                    FTFShutdownReply *pShutdownReply,
                    FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#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.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN)</td>
<td>Connection handle</td>
<td>Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.</td>
</tr>
<tr>
<td>pShutdownInfo (FTFShutdownInfo *)</td>
<td>Pointer to input data structure</td>
<td>Represents the input data structure that contains all the required and optional attributes that pertain to the shutdown options.</td>
</tr>
<tr>
<td>pShutdownReply (FTFShtdownReply *)</td>
<td>Pointer to output data structure</td>
<td>Represents the output data structure that contains all the output and results from the shutdown API request.</td>
</tr>
<tr>
<td>pttfca (TFCA *)</td>
<td>Pointer to a Tivoli Data Exchange reply message data structure</td>
<td>Represents an output area that returns properly formatted API reply information. This includes the return codes and optional messages.</td>
</tr>
</tbody>
</table>

## Related Data Structures

- FTFCA (page 179)
- FTFShutdownInfo (page 223)
- FTFShutdownReply (page 225)
**FTFStage**

**Description**

This API allows you either to query the items in the staging queue for the specified Tivoli Data Exchange Sender, or to purge the items in a specified Tivoli Data Exchange 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**

```c
#include "ftfc.h"

FTFVOID FTFStage
(MQHCONN hQM,
FTFStageMsgInfo *pRequest,
FTFStagedListMsgInfo *pStagedListMsgInfo,
FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"

FTFVOID FTFSTAGE
(MQHCONN hQM,
FTFStageMsgInfo *pRequest,
FTFStagedListMsgInfo *pStagedListMsgInfo,
FTFCA *pftfca);
```
# C API Reference

## FTFStage

### Parameters

The following table lists and describes the FTFStage API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN)</td>
<td>Connection handle</td>
<td>Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.</td>
</tr>
<tr>
<td>pRequest</td>
<td>Pointer to input data structure</td>
<td>Represents the input data structure that contains the information required for the staging operation to function.</td>
</tr>
<tr>
<td>pStagedListMsgInfo</td>
<td>Pointer to output data structure</td>
<td>Represents the output data structure that contains all data structures with staging query information.</td>
</tr>
<tr>
<td>pTfca (TFCA *)</td>
<td>Pointer to a Tivoli Data Exchange reply message data structure</td>
<td>Represents an output area that returns properly formatted API reply information. This includes the return codes and optional messages.</td>
</tr>
</tbody>
</table>

### Related Data Structures

- FTFCA (page 179)
- FTFStagedListMsgInfo (page 231)
- FTFStagedTransactionMsgInfo (page 232)
- FTFStageMsgInfo (page 234)
C API Reference
FTFStatusDelete

FTFStatusDelete

Description

This API purges status messages from the MQSeries environment. It removes all status messages associated with a specific FTFID from the Tivoli Data Exchange 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

```c
#include "ftfc.h"

FTFVOID FTFStatusDelete(MQHCONN hQM,
FTFStatusDeleteInfo *info,
FTFCAPCA *pftfca);
```

OS/390 Prototype

```c
#include "ftfc.h"

FTFVOID FTFSD(MQHCONN hQM,
FTFStatusDeleteInfo *info,
FTFCAPCA *pftfca);
```

Parameters

The following table lists and describes the FTFStatusDelete API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HQM (MQCONN)</td>
<td>Connection Handle</td>
<td>Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.</td>
</tr>
</tbody>
</table>
C API Reference

FTFStatusDelete

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info (FTFStatusDelete Info *)</td>
<td>Pointer to a status delete data structure</td>
<td>Represents the input area that determines the status records being deleted.</td>
</tr>
<tr>
<td>pftfca (FTFCA *) - output</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTFCA (page 179)
- FTFStatusDeleteInfo (page 253)
**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**

```c
#include "ftfc.h"

FTFVOID FTFStatusFreeDetailList
(FTFStatDetailList *pStatDetailList,
 FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"

FTFVOID FTFSFDL
(FTFStatDetailList *pStatDetailList,
 FTFCA *pftfca);
```

**Parameters**

The following table lists and describes the FTFStatusFreeDetailList API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pStatDetailList (FTFStatDetailList *)</td>
<td>Pointer to a Status Detail List data</td>
<td>Represents the data structure that contains all the detailed status information for a complete Tivoli Data Exchange data transfer.</td>
</tr>
<tr>
<td>- input</td>
<td>structure</td>
<td></td>
</tr>
<tr>
<td>pftfca (FTFCA *)</td>
<td>Pointer to a Tivoli Data Exchange Reply</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
<tr>
<td>- output</td>
<td>Message Data Structure</td>
<td></td>
</tr>
</tbody>
</table>
C API Reference

FTFStatusFreeDetailList

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 179)
- FTFStatDetail (page 236)
- FTFStatDetailList (page 246)
**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**

```c
#include “ftfc.h”

FTFVOID FTFStatusFreeSummaryList
(FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include “ftfc.h”

FTFVOID FTFSFSL
(FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```
C API Reference

FTFStatusFreeSummaryList

Parameters

The following table lists and describes the FTFStatusFreeSummaryList API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pStatSummaryList *(FTFStatSummaryList *)</td>
<td>Pointer to a Status Summary List data structure</td>
<td>Represents the data structure that contains the summary status information of the Tivoli Data Exchange components for all data-transfer requests that meet the filter criteria. It contains the most recent status updates for each Tivoli Data Exchange component in each matching data transfer.</td>
</tr>
<tr>
<td>pftfca *(FTFCA *)</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
<td>Represents an output area that returns properly formatted API reply information, including return codes and optional messages.</td>
</tr>
</tbody>
</table>

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 179)
- FTFStatSummary (page 247)
- FTFStatSummaryList (page 251)
FTFStatusGetDetailList

Description

This API retrieves all detailed summary updates from Tivoli Data Exchange components during a Tivoli Data Exchange 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 Tivoli Data Exchange data transfer.

General Prototype

```c
#include "ftfc.h"

FTFVOID FTFStatusGetDetailList
 (MQHCONN hQM,
  FTFStatusGetDetailListInfo *info,
  FTFStatDetailList *pStatDetailList,
  FTFCA *pftfca);
```

OS/390 Prototype

```c
#include "ftfc.h"

FTFVOID FTFDL
 (MQHCONN hQM,
  FTFStatusGetDetailListInfo *info,
  FTFStatDetailList *pStatDetailList,
  FTFCA *pftfca);
```
C API Reference
FTFStatusGetDetailList

Parameters

The following table lists and describes the FTFStatusGetDetailList API’s parameters.

<table>
<thead>
<tr>
<th>Parameter (Data Type)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HQM (MQHCONN)</td>
<td>- input</td>
<td>Connection Handle</td>
</tr>
<tr>
<td>pStatDetailList</td>
<td>- output</td>
<td>Pointer to a Status Detail List data structure</td>
</tr>
<tr>
<td>info</td>
<td>- output</td>
<td>Pointer to Tivoli Data Exchange Status Get Detail List Information Data Structure</td>
</tr>
<tr>
<td>pftfca</td>
<td>- output</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
</tr>
</tbody>
</table>

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 179)
- FTFStatDetail (page 236)
- FTFStatDetailList (page 246)
- FTFStatusGetDetailListInfo (page 255)
Each Tivoli Data Exchange component sends status updates during the processing cycle of a Tivoli Data Exchange 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**

```c
#include "ftfc.h"
FTFVOID FTFStatusGetSummaryList
(MQHCONN hQM,
FTFStatusGetSummaryListInfo *info,
FTFStatusSummaryFilter *pSummaryFilter,
FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"
FTFVOID FTFSL
(MQHCONN hQM,
FTFStatusGetSummaryListInfo *info,
FTFStatusSummaryFilter *pSummaryFilter,
FTFStatSummaryList *pStatSummaryList,
FTFCA *pftfca);
```
C API Reference

FTFStatusGetSummaryList

Parameters

The following table lists and describes the FTFStatusGetSummaryList API’s parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hQM (MQHCONN) - input</td>
<td>Connection Handle</td>
<td>Represents the connection to the queue manager. The value of Hconn was returned by a previous MQCONN call.</td>
</tr>
<tr>
<td>info (FTFStatusGetSummaryList Info*)</td>
<td>Pointer to Tivoli Data Exchange Status Get Summary List Information Data Structure</td>
<td>Represents an input area that determines the summary information being returned.</td>
</tr>
<tr>
<td>pSummaryFilter (FTFStatusSummaryFilter *) - input</td>
<td>Pointer to a Status Summary Filter data structure</td>
<td>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 Tivoli Data Exchange transactions.</td>
</tr>
<tr>
<td>pStatSummaryList (FTFStatSummaryList *) - output</td>
<td>Pointer to a Status Summary List data structure</td>
<td>Represents the output data structure that contains the summary status information of the Tivoli Data Exchange components of all transactions that meet the filter criteria. It contains the most recent status updates for each Tivoli Data Exchange component associated with the transactions.</td>
</tr>
<tr>
<td>pftfca (FTFCA *) - output</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
<td>Represents an output area that returns properly formatted API reply information, including return codes and optional messages.</td>
</tr>
</tbody>
</table>

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.
C API Reference

FTFStatusGetSummaryList

- The FTFStatusFreeDetailList API call fails if you increment the pRow data element before you call the API.

Related Data Structures

- FTFCA (page 179)
- FTFStatSummaryList (page 251)
- FTFStatusGetSummaryListInfo (page 257)
- FTFStatusSummaryFilter (page 259)
**C API Reference**

**FTFSubmitStatusMsg**

**Description**

This API allows you to submit a status message from a user exit module to the Tivoli Data Exchange status subsystem. The message is then processed in the same manner as Tivoli Data Exchange statues messages are. It appears in status detail information retrieved by FTFStatusGetDetailList.

**General Prototype**

```c
#include "ftfc.h"

FTFVOID FTFSubmitStatusMsg
(FTFExitInfo *pFTFExitInfo,
FTFCHAR customComponent[],
FTFCHAR customStatusType[],
FTFCHAR errorText[],
FTFC *pftfca);
```

**OS/390 Prototype**

```c
#include "ftfc.h"

FTFVOID FTFSSM
(FTFExitInfo *pFTFExitInfo,
FTFCHAR customComponent[],
FTFCHAR customStatusType[],
FTFCHAR errorText[],
FTFC *pftfca);
```
The following table lists and describes the FTFSubmitStatusMsg API’s parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pFTFExitInfo (FTFExitInfo) - input</td>
<td>Pointer to an FTFExitInfo data structure</td>
<td>Contains the address of the variable that contains the FTFExitInfo data structure. This data structure contains the information required for processing user exit modules.</td>
</tr>
<tr>
<td>customComponent (FTFCHAR) - input</td>
<td>Character array</td>
<td>Contains the user-specified custom component submitted to the status subsystem.</td>
</tr>
<tr>
<td>customStatusType (FTFCHAR) - input</td>
<td>Character array</td>
<td>Contains the custom status type being submitted to the status subsystem.</td>
</tr>
<tr>
<td>errorText (FTFCHAR) - input</td>
<td>Character array</td>
<td>Contains a textual error text submitted to the status subsystem.</td>
</tr>
<tr>
<td>pftfca (FTFCA *)</td>
<td>Pointer to a Tivoli Data Exchange Reply Message Data Structure</td>
<td>Contains the address of the variable that returns the properly formatted API reply information, including return codes and optional messages.</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTFCA (page 179)
- FTFExitInfo (page 189)
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 root directory.

This chapter includes the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Status Messages</td>
<td>162</td>
</tr>
<tr>
<td>FTFAS400FileInfo</td>
<td>165</td>
</tr>
<tr>
<td>FTFCA</td>
<td>167</td>
</tr>
<tr>
<td>FTFCancelInfo</td>
<td>169</td>
</tr>
<tr>
<td>FTFExitAuthInfo</td>
<td>171</td>
</tr>
<tr>
<td>FTFExitFileInfo</td>
<td>173</td>
</tr>
<tr>
<td>FTFExitInfo</td>
<td>177</td>
</tr>
<tr>
<td>FTFExitJobInfo</td>
<td>181</td>
</tr>
<tr>
<td>FTFExitQMgrsInfo</td>
<td>184</td>
</tr>
<tr>
<td>FTFExitRequestInfo</td>
<td>186</td>
</tr>
<tr>
<td>FTFExitSourceFileInfo</td>
<td>188</td>
</tr>
<tr>
<td>FTFExitStatusOffload</td>
<td>191</td>
</tr>
<tr>
<td>FTFExitTargetFileInfo</td>
<td>193</td>
</tr>
<tr>
<td>FTFExitUserInfo</td>
<td>197</td>
</tr>
<tr>
<td>FTFIdentifiersInfo</td>
<td>199</td>
</tr>
<tr>
<td>FTFJobInfo</td>
<td>200</td>
</tr>
<tr>
<td>FTPingInfo</td>
<td>203</td>
</tr>
<tr>
<td>FTFQMgrsInfo</td>
<td>206</td>
</tr>
</tbody>
</table>
## C Data Structures

### Component Status Messages

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFRequestMsgInfo</td>
<td>208</td>
</tr>
<tr>
<td>FTFShutdownInfo</td>
<td>211</td>
</tr>
<tr>
<td>FTFShutdownReply</td>
<td>213</td>
</tr>
<tr>
<td>FTFSourceFileInfo</td>
<td>216</td>
</tr>
<tr>
<td>FTFStagedListMsgInfo</td>
<td>219</td>
</tr>
<tr>
<td>FTFStagedTransactionMsgInfo</td>
<td>220</td>
</tr>
<tr>
<td>FTFStageMsgInfo</td>
<td>222</td>
</tr>
<tr>
<td>FTFStatDetail</td>
<td>224</td>
</tr>
<tr>
<td>FTFStatDetailList</td>
<td>234</td>
</tr>
<tr>
<td>FTFStatSummary</td>
<td>235</td>
</tr>
<tr>
<td>FTFStatSummaryList</td>
<td>239</td>
</tr>
<tr>
<td>FTFStatusDeleteInfo</td>
<td>241</td>
</tr>
<tr>
<td>FTFStatusGetDetailListInfo</td>
<td>243</td>
</tr>
<tr>
<td>FTFStatusGetSummaryListInfo</td>
<td>245</td>
</tr>
<tr>
<td>FTFStatusSummaryFilter</td>
<td>247</td>
</tr>
<tr>
<td>FTFTargetFileInfo</td>
<td>251</td>
</tr>
<tr>
<td>FTFUserInfo</td>
<td>256</td>
</tr>
</tbody>
</table>

### Component Status Messages

This section lists and describes the most common Tivoli Data Exchange component and status combinations used in a data-transfer request. It lists status messages generated by the Tivoli Data Exchange Manager, the Tivoli Data Exchange Sender, and the Tivoli Data Exchange Receiver.
### Tivoli Data Exchange Manager Status Messages

<table>
<thead>
<tr>
<th>Tivoli Data Exchange Manager Status Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSTAT_REQUEST_SUBMITTED</td>
<td>The Tivoli Data Exchange Manager has sent the data-transfer request to the Tivoli Data Exchange Sender for processing.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_FLAGGED_FOR_EXPIRATION</td>
<td>The Tivoli Data Exchange Manager has marked the data-transfer request for expiration.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_EXPIRED</td>
<td>The data-transfer request has expired.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_FLAGGED_FOR_CANCEL</td>
<td>The Tivoli Data Exchange Manager has marked the data-transfer request for cancellation.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_CANCELLED</td>
<td>The data-transfer request has been canceled.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_FAILED</td>
<td>The data-transfer request has failed.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_COMPLETE</td>
<td>The data-transfer request has completed successfully.</td>
</tr>
</tbody>
</table>

### Tivoli Data Exchange Sender Status Messages

<table>
<thead>
<tr>
<th>Tivoli Data Exchange Sender Status Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSTAT_REQUEST_RECEIVED</td>
<td>The Tivoli Data Exchange Sender has received the data-transfer request.</td>
</tr>
<tr>
<td>FTFSTAT_PROCESSING</td>
<td>The Tivoli Data Exchange Sender is processing the data-transfer request.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_TRANSMITTED</td>
<td>The Tivoli Data Exchange Sender has transmitted the required messages to the Tivoli Data Exchange Receiver.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_RECOVERING</td>
<td>Tivoli Data Exchange could not complete the data-transfer request. The Tivoli Data Exchange Sender is now recovering.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_FAILED</td>
<td>The data-transfer request has failed.</td>
</tr>
</tbody>
</table>
C Data Structures

Component Status Messages

<table>
<thead>
<tr>
<th>Tivoli Data Exchange Sender Status Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSTAT_FAILED</td>
<td>The Tivoli Data Exchange Sender could not process the data-transfer request.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_COMPLETE</td>
<td>The Tivoli Data Exchange Sender has completed processing the data-transfer request.</td>
</tr>
</tbody>
</table>

Tivoli Data Exchange Receiver Status Messages

<table>
<thead>
<tr>
<th>Tivoli Data Exchange Receiver Status Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSTAT_REQUEST_RECEIVED</td>
<td>The Tivoli Data Exchange Receiver has received the request.</td>
</tr>
<tr>
<td>FTFSTAT_PROCESSING</td>
<td>The Tivoli Data Exchange Receiver is processing the messages transmitted by the Tivoli Data Exchange Sender.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_RECOVERING</td>
<td>Tivoli Data Exchange could not complete the data-transfer request. The Tivoli Data Exchange Receiver is now recovering.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_EXPIRED</td>
<td>The request has expired.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_CANCELLED</td>
<td>The request has been cancelled.</td>
</tr>
<tr>
<td>FTFSTAT_FAILED</td>
<td>The Tivoli Data Exchange Receiver component could not process the request.</td>
</tr>
<tr>
<td>FTFSTAT_REQUEST_COMPLETE</td>
<td>The Tivoli Data Exchange Receiver has completed processing the request.</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccsid (FTFCHAR[ ])</td>
<td>Fixed length array that contains the CCSID value used as an identifier for the data-transfer request. If no CCSID is specified, Tivoli Data Exchange uses the job’s CCSID.</td>
</tr>
<tr>
<td>rcdlen (FTFLONG)</td>
<td>Determines the logical record length for the AS/400 target file. <strong>Valid values:</strong> 1-32767</td>
</tr>
<tr>
<td>fileType (FTFFileTypeInfo)</td>
<td>Enumerated data type that determines the type of file being written to the AS/400 target. <strong>Valid values:</strong> FTF_BINARY, FTF_TEXT</td>
</tr>
<tr>
<td>fileTypeAS400 (FTFAS400FileType)</td>
<td>Enumerated data type that determines the AS/400 file type for the target file. <strong>Valid values:</strong> FTF_AS400_TYPE_SAVE, FTF_AS400_TYPE_PHYSICAL</td>
</tr>
<tr>
<td>isCreateLibrary (FTFBOOL)</td>
<td>Determines whether Tivoli Data Exchange should create the specified library if it does not already exist.</td>
</tr>
<tr>
<td>libAsp (FTFLONG)</td>
<td>Specifies the library auxiliary storage pool (ASP) for a library created as part of the data-transfer request.</td>
</tr>
<tr>
<td>fileAsp (FTFLONG)</td>
<td>Specifies the file ASP for a file created as part of the data-transfer request.</td>
</tr>
<tr>
<td>libText (FTFCHAR[ ])</td>
<td>Fixed length array that specifies the description for a library created as part of the data-transfer request.</td>
</tr>
<tr>
<td>fileText (FTFCHAR[ ])</td>
<td>Fixed length array that specifies the description for a file created as part of the data-transfer request.</td>
</tr>
</tbody>
</table>
C Data Structures

FTFAS400FileInfo

Related Functions

- FTFCancel (page 147)

Related Data Structures

- FTFRequestMsgInfo (page 208)

Data Structure

typedef struct _FTFAS400FileInfo {
    FTFCHAR ccsid[FTF_AS400_CCSID_SIZE];
    FTFLONG rcdlen;
    FTFFileTypeInfo fileType;
    FTFAS400FileType fileTypeAS400;
    FTFOOL libAsp;
    FTFLONG fileAsp;
    FTFCHAR libText[FTF_AS400_TEXT_SIZE];
    FTFCHAR 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc1 (FTFRC)</td>
<td>Contains the primary return code or FTFRC. Primary return codes are defined in the ftfc.h header file and represent a Tivoli Data Exchange internal code with an associated message.</td>
</tr>
<tr>
<td>rc2 (FTFRC)</td>
<td>Contains the secondary return code. This data element contains external return codes or reason codes. For example, if Tivoli Data Exchange encounters an MQSeries error, the primary return code is a Tivoli Data Exchange return code and the secondary return code is an MQSeries return code (found in the cmqc.h). However, if the error is a Tivoli Data Exchange internal error the secondary code (rc2) can be either 0 or an additional Tivoli Data Exchange return code.</td>
</tr>
<tr>
<td>FTFErrMsg (FTFCHAR[ ])</td>
<td>Fixed length array that contains message text and is optional in many cases.</td>
</tr>
</tbody>
</table>
C Data Structures

FTFCA

Related Functions

- FTFCancel (page 147)
- FTFPing (page 149)
- FTFReq (page 151)
- FTFShutdown (page 154)
- FTFStage (page 156)
- FTFStatusDelete (page 158)
- FTFStatusFreeDetailList (page 160)
- FTFStatusFreeSummaryList (page 162)
- FTFStatusGetDetailList (page 164)
- FTFStatusGetSummaryList (page 166)
- FTFSubmitStatusMsg (page 169)

Related Data Structures

None

Data Structure

```
typedef struct _FTFCA {
    FTFRC    rc1;
    FTFRC    rc2;
    FTFCHAR  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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| lqm (FTFCHAR[ ]) | Fixed length array that contains the name of the queue manager to which the **FTFCancel** API has a connection. If it is not provided, the local queue manager takes the value of the default queue manager.  
**Note**: 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 **FTFCancel** API. This value represents the queue manager for which the Tivoli Data Exchange 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 Tivoli Data Exchange 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 147)
C Data Structures

FTFCancelInfo

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUserid (FTFCHAR *)</td>
<td>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.</td>
</tr>
<tr>
<td>pPassword (FTFCHAR *)</td>
<td>Included for possible future use. Leave this data element null.</td>
</tr>
<tr>
<td>pFileName (FTFCHAR *)</td>
<td>Pointer to a variable that contains the name of the file Tivoli Data Exchange will read or write.</td>
</tr>
<tr>
<td>pSystem (FTFCHAR *)</td>
<td>Pointer to a variable that contains the NT domain name on Windows NT machines. For other operating systems, this data element should be left null.</td>
</tr>
<tr>
<td>accessFlag (FTFLONG)</td>
<td>Set this data element to FTF_READ on the Tivoli Data Exchange Sender and FTF_WRITE on the Tivoli Data Exchange Receiver.</td>
</tr>
</tbody>
</table>

**Related Functions**


**Related Data Structures**

None
Data Structure

typedef struct _FTFExitAuthInfo {
    FTFCHAR *pUserid;
    FTFCHAR *pPassword;
    FTFCHAR *pFileName;
    FTFCHAR *pSystem;
    FTFLONG accessFlag;
} FTFExitAuthInfo;
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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFileName (FTFCHAR*)</td>
<td>Determines whether the file is to 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 transmitted to each of its destinations. <strong>Valid values:</strong> 1 (place file in staging queue), 0 (do not place file in staging queue).</td>
</tr>
<tr>
<td>isStaged (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (make staged data persistent), 0 (do not make staged data persistent).</td>
</tr>
<tr>
<td>isStagePersistent (FTFBOOL)</td>
<td></td>
</tr>
<tr>
<td>isDataPersistent (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (transmission is persistent), 0 (transmission not persistent).</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Determines whether the file being sent is compressed using the Tivoli Data Exchange internal compression algorithm. <strong>Valid values:</strong> 1 (compression is used) 0 (compression is not used)</td>
</tr>
<tr>
<td>fileType (FTFFileTypeInfo)</td>
<td>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. <strong>Valid values:</strong> FTF_BINARY (binary), FTF_TEXT (text).</td>
</tr>
</tbody>
</table>
### C Data Structures

**FTFExitFileInfo**

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| fileMode (FTFFileModeInfo) | Determines how the data is processed at the destination. **Valid values:**  
  - FTF>Create_FILE (create a new file),  
  - FTF_APPEND_FILE (append to an existing file),  
  - FTF_NOREPLACE_FILE (do not replace an existing file). |
| fileOrg (FTFFileOrgInfo) | Determines the target file’s OS/390 file organization. **Valid values:** PS (Physical Sequential), PDS (Partitioned Data Set)  
  **Note:** See the following paragraph, *OS/390 File Allocation Items*. |
| 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. **Valid values:** F (fixed), V (variable), FB (fixed block), and VB (variable block)  
  **Note:** See the following paragraph, *OS/390 File Allocation Items*. |
| lrecl (FTFLONG) | Determines the target file’s OS/390 logical record length. **Valid values:** 1 - 32760.  
  **Note:** See the following paragraph, *OS/390 File Allocation Items*. |
<p>| 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. <strong>Valid values:</strong> 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. <strong>Valid values:</strong> 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. |</p>
<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| textWrapRecord            | Indicates how records are processed when they are longer than the specified target-file record length. **Valid values:**  
| (FTFTextFileWrapInfo)     |   • FTFREQ_WRAP (wraps records)                                               |
|                          |   • FTFREQ_FAIL (causes the data transfer to fail)                            |
|                          |   • FTFREQ_TRUNC (truncates records at the specified record length)           |
| createDirectory          | Determines whether the specified target directory is created when it does not already exist. **Valid values:** 1 (create directory), 0 (do not create a directory). |
| (FTFBOOL)                |                                                                             |

**Related Functions**

None

**Related Data Structures**

- FTFExitAuthInfo (page 171)
- FTFExitJobInfo (page 181)
- FTFExitQMgrsInfo (page 184)
- FTFExitRequestInfo (page 186)
- FTFExitSourceFileInfo (page 188)
- FTFExitTargetFileInfo (page 193)
- FTFExitUserInfo (page 197)
C Data Structures

FTFExitFileInfo

Data Structure

typedef struct _FTFEXITFILEINFO {
    FTFCHAR *pFileName;
    FTFBOOL isStaged;
    FTFBOOL isStagePersistent;
    FTFBOOL isDataPersistent;
    FTFBOOL isCompressed;
    FTTFileTypeInfo fileType;
    FTT FileModeInfo fileMode;
    FTTFileOrgInfo fileOrg;
    FTFLONG cDirectoryBlocks;
    FTFRecordFormatInfo recordFormat;
    FTFLONG lrecl;
    FTFLONG blockSize;
    FTFCHAR unitName[FTFREQ_UNITNAME_SIZE];
    FTFCHAR volser[FTFREQ_VOLSER_SIZE];
    FTFAllocUnitInfo allocationUnit;
    FTFLONG primaryAllocSize;
    FTFLONG secondaryAllocSize;
    FTTTextFileWrapInfo 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 Tivoli Data Exchange. 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).
- 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftfid (FTFCHAR[ ])</td>
<td>Fixed length array that contains a Tivoli Data Exchange identifier used to identify the current data-transfer request. The FTFReq API generates this unique identifier.</td>
</tr>
<tr>
<td>exitNumber (FTFLONG)</td>
<td>Contains the number of the exit to be called. The following exits are available:</td>
</tr>
<tr>
<td></td>
<td>3. Manager pre-process exit User-written exit for pre-processing requirements at the Tivoli Data Exchange Manager. This exit is invoked before the data-transfer request is sent to the Tivoli Data Exchange Sender for processing.</td>
</tr>
</tbody>
</table>
### C Data Structures

**FTF Exit Info**

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Manager post-process exit</td>
<td>User-written exit for pre-processing requirements at the Tivoli Data Exchange 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 Tivoli Data Exchange Manager post-process exit can be used to audit file transfer for all Tivoli Data Exchange requests that it processes in order to detect failure or success.</td>
</tr>
<tr>
<td>5. Sender pre-process exit</td>
<td>User-written exit for pre-processing requirements at the Tivoli Data Exchange Sender. This exit is invoked before the Tivoli Data Exchange Sender reads the source file.</td>
</tr>
<tr>
<td>6. Sender post-process exit</td>
<td>User-written exit for post-processing requirements at the Tivoli Data Exchange Sender. This exit is invoked after the Tivoli Data Exchange Sender deposits the outbound messages on the transmission queues.</td>
</tr>
<tr>
<td>7. Receiver pre-process exit</td>
<td>User-written exit for pre-processing requirements at the Tivoli Data Exchange Receiver. This exit is invoked before the Tivoli Data Exchange Receiver processes inbound messages and writes them to the target file.</td>
</tr>
<tr>
<td>8. Receiver post-process exit</td>
<td>User-written exit for post-processing requirements at the Tivoli Data Exchange Receiver. This exit is invoked by the Tivoli Data Exchange Receiver after the target file is written. Because the Tivoli Data Exchange receiver performs the last step in the data-transfer request just before the Tivoli Data Exchange Manager logs the final status, this exit is useful for auditing failed or successful data transfer requests.</td>
</tr>
<tr>
<td>9. Sender connector</td>
<td>For information about custom connector development, contact CommerceQuest (<a href="http://www.Commercequest.com">www.Commercequest.com</a>).</td>
</tr>
<tr>
<td>10. Receiver connector</td>
<td>For information about custom connector development, contact CommerceQuest (<a href="http://www.commercequest.com">www.commercequest.com</a>).</td>
</tr>
<tr>
<td>pDllName (FTFCHAR*)</td>
<td>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).</td>
</tr>
</tbody>
</table>
C Data Structures

FTFExitInfo

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

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pEntryPoint (FTFCHAR*)</td>
<td>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.</td>
</tr>
<tr>
<td>pUserId (FTFCHAR*)</td>
<td>Pointer to a variable that contains a user ID. Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>pPassword (FTFCHAR*)</td>
<td>Pointer to a variable that contains a password. Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>cUserData (FTFLONG)</td>
<td>Contains the number of bytes within the userData data element.</td>
</tr>
<tr>
<td>pUserData (FTFCHAR *)</td>
<td>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.</td>
</tr>
<tr>
<td>rc (FTFRC)</td>
<td>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.</td>
</tr>
<tr>
<td>rc2 (FTFRC)</td>
<td>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.</td>
</tr>
<tr>
<td>pInternal (FTFVOID*)</td>
<td>Contains the three user-defined fields which are associated with the data-transfer request.</td>
</tr>
<tr>
<td>Ids (FTFIdentifiersInfo)</td>
<td>Ids (FTFIdentifiersInfo)</td>
</tr>
</tbody>
</table>
C Data Structures

FTFExitInfo

Related Data Structures

- FTFExitAuthInfo (page 171)
- FTFExitJobInfo (page 181)
- FTFExitQMgrsInfo (page 184)
- FTFExitRequestInfo (page 186)
- FTFExitSourceFileInfo (page 188)
- FTFExitTargetFileInfo (page 193)
- FTFExitUserInfo (page 197)
- FTFIdentifiersInfo (page 199)

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>poolName (FTFCHAR[ ])</td>
<td>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 <em>Tivoli Data Exchange User's Guide</em>, “About Pools.”</td>
</tr>
<tr>
<td>priority (FTFLONG)</td>
<td>Represents the priority of the current data-transfer request. <strong>Valid values:</strong> 1 (highest) - 5 (lowest)</td>
</tr>
<tr>
<td>expirationDateTime (FTFLONG)</td>
<td>Represents the number of seconds between the current time and the time that the data-transfer request expires.</td>
</tr>
<tr>
<td>isTrusted (FTFBOOL)</td>
<td>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 Tivoli Data Exchange protection), Tivoli Data Exchange does not perform syncpoint control. In the event of a failure, the data-transfer request is terminated with a failing notification. <strong>Valid values:</strong> 1 (trusted), 0 (not trusted). <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>reserved (FTFBOOL)</td>
<td>For internal use only. Do not populate this data element.</td>
</tr>
<tr>
<td>isStageOnly (FTFBOOL)</td>
<td>Determinates whether the source information is to be sent only to a staging queue and not to another destination. <strong>Valid values:</strong> 1 (send only to a staging queue), any other value (do not send staging queue)</td>
</tr>
</tbody>
</table>
## C Data Structures

### FTFExitJobInfo

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isStagedftfid (FTFBOOL)</td>
<td>This data element determines whether the pSource data element in the FTFExitRequestInfo 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. <strong>Valid values:</strong> 1 (True, use the FTFID), any other value (False, do not use the FTFID)</td>
</tr>
<tr>
<td>isStagedFile (FTFBOOL)</td>
<td>This data element determines whether the pSource 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. <strong>Valid values:</strong> 1 (True, use the filename), any other value (False, do not use the filename)</td>
</tr>
</tbody>
</table>
| cancelMode (FTFCancelMode) | Determines how pre-emptive cancellation should be handled. **Valid values:**  
  - FTF_PREEMPTCANCEL_NEUTRAL (uses the cancellation value set in the Tivoli Data Exchange configuration file),  
  - FTF_PREEMPTCANCEL_ON (allows the file transfer to be preemptively canceled),  
  - FTF_PREEMPTCANCEL_OFF (does not allow preemptive cancellation) |
| 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 186)
- FTFIdentifiersInfo (page 199)
- FTFJobInfo (page 200)

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;
    FTFIdentifiersInfo Ids;
} FTFExitJobInfo;
C Data Structures

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 Tivoli Data Exchange subsystem. If not specified, the requesting interface connects to the default queue manager.  
**Note:** This element is required on OS/390.  
**Valid values:** 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 Tivoli Data Exchange data-transfer request originates. If omitted, the Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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
Related Data Structures

- FTFExitRequestInfo (page 186)
- FTFQMgrsInfo (page 206)

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;
C Data Structures

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pQMgrs (FTFExitQMgrsInfo*)</td>
<td>Pointer to a variable that contains queue manager information that resides in the FTFExitQMgrsInfo data structure.</td>
</tr>
<tr>
<td>pSource (FTFExitSourceFileInfo*)</td>
<td>Pointer to a variable that contains source file information that resides in the FTFExitSourceFileInfo data structure.</td>
</tr>
<tr>
<td>pTarget (FTFExitTargetFileInfo*)</td>
<td>Pointer to a variable that contains target file information that resides in the FTFExitTargetFileInfo data structure.</td>
</tr>
<tr>
<td>pJob (FTFExitJobInfo*)</td>
<td>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.</td>
</tr>
<tr>
<td>pUser (FTFExitUserInfo*)</td>
<td>Pointer to a variable that contains information about the data-transfer group name and label. This information resides in the FTFExitUserInfo data structure.</td>
</tr>
</tbody>
</table>

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 171)
- FTFExitInfo (page 177)
- FTFExitJobInfo (page 181)
- FTFExitQMgrsInfo (page 184)
- FTFExitSourceFileInfo (page 188)
- FTFExitTargetFileInfo (page 193)
- FTFExitUserInfo (page 197)
- FTFRequestMsgInfo (page 208)

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilename (FTFCHAR*)</td>
<td>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.</td>
</tr>
<tr>
<td>isStaged (FTFBOOL)</td>
<td>Determines whether the file is to 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 transmitted to each of its destinations. <strong>Valid values:</strong> 1 (place file in staging queue), 0 (do not place file in staging queue).</td>
</tr>
<tr>
<td>isStagePersistent (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (make staged data persistent), 0 (do not make staged data persistent).</td>
</tr>
<tr>
<td>isDataPersistent (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (transmission is persistent), 0 (transmission not persistent).</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Determines whether the file being sent is compressed using the Tivoli Data Exchange internal compression algorithm. <strong>Valid values:</strong> 1 (compression is used), 0 (compression is not used).</td>
</tr>
</tbody>
</table>
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 171)
- FTFExitInfo (page 177)
- FTFExitJobInfo (page 181)
- FTFExitQMgrsInfo (page 184)
- FTFExitSourceFileInfo (page 188)
- FTFExitTargetFileInfo (page 193)
- FTFExitUserInfo (page 197)
- FTFExitSourceFileInfo (page 188)
**C Data Structures**

*FTFExitSourceFileInfo*

**Data Structure**

```c
typedef struct _FTFExitSourceFileInfo {
  FTFCHAR    *pFileName;
  FTFBOOL    isStaged;
  FTFBOOL    isStagePersistent;
  FTFBOOL    isDataPersistent;
  FTFBOOL    isCompressed;
  FTFFInfoType fileTypeInfo;
  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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF_STATD_ACTION</td>
<td>Action to take when the exit is called (initiate, submit, end).</td>
</tr>
<tr>
<td>pszDBName (FTFCHAR)</td>
<td>Name of the data source.</td>
</tr>
<tr>
<td>pszDBUserID (FTFCHAR)</td>
<td>Database user ID.</td>
</tr>
<tr>
<td>pszDBPassword (FTFCHAR)</td>
<td>Database password.</td>
</tr>
<tr>
<td>StatMsgBuff (FTFCHAR)</td>
<td>Pointer to an array of XML strings.</td>
</tr>
<tr>
<td>pszUserData (FTFCHAR)</td>
<td>Pointer to user exit information.</td>
</tr>
<tr>
<td>lBatchSize (FTFLONG)</td>
<td>Number of records in this batch.</td>
</tr>
<tr>
<td>cControlMsg (FTFLONG)</td>
<td>Count of the number of control messages (always at the top of the buffer).</td>
</tr>
<tr>
<td>cDetailMsg (FTFLONG)</td>
<td>Count of the number of detail messages.</td>
</tr>
<tr>
<td>pLogFile</td>
<td>Pointer to the FTFSTATD log file to log any SQL error text</td>
</tr>
<tr>
<td>rc (FTFLONG)</td>
<td>Contains the primary return code value.</td>
</tr>
<tr>
<td>rc2 (FTFLONG)</td>
<td>Contains the secondary (SQL) return code value.</td>
</tr>
</tbody>
</table>

**Related Functions**

None
C Data Structures

FTFExitStatusOffload

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFileName (FTFCHAR*)</td>
<td>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.</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Reflects whether inbound data is compressed or uncompressed. <strong>Valid values:</strong> 1 (compressed), 0 (uncompressed).</td>
</tr>
<tr>
<td>fileType (FTFFileTypeInfo)</td>
<td>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. <strong>Valid values:</strong> FTF_BINARY (binary), FTF_TEXT (text).</td>
</tr>
<tr>
<td>FileMode (FTFFileModeInfo)</td>
<td>Determines how the data is processed at the destination. <strong>Valid values:</strong> • FTF_CREATE_FILE (create a new file), • FTF_APPEND_FILE (append to an existing file), • FTF_NOREPLACE_FILE (do not replace an existing file).</td>
</tr>
<tr>
<td>fileOrg (FTFFileOrgInfo)</td>
<td>Determines the target file’s OS/390 file organization. <strong>Valid values:</strong> PS (Physical Sequential), PDS (Partitioned Data Set) <strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>cDirectoryBlocks (FTFLONG)</td>
<td>Determines the number of directory blocks allocated for the target file on OS/390.</td>
</tr>
<tr>
<td>recordFormat (FTFRecordFormatInfo)</td>
<td>Determines the target file’s OS/390 record format. <strong>Valid values:</strong> F (fixed), V (variable), FB (fixed block), and VB (variable block) <strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
</tbody>
</table>
### C Data Structures

#### FTFExitTargetFileInfo

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrecl (FTFLONG)</td>
<td>Determines the target file’s OS/390 logical record length. <strong>Valid values:</strong> 1 - 32760. <strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>blockSize (FTFLONG)</td>
<td>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. <strong>Valid values:</strong> 0 - 32760</td>
</tr>
<tr>
<td>unitName (FTFCHAR[ ])</td>
<td>Fixed length array that determines the target file’s OS/390 unit name.</td>
</tr>
<tr>
<td>volser (FTFCHAR[ ])</td>
<td>Fixed length array that determines the target file’s volume serial number.</td>
</tr>
<tr>
<td>allocationUnit (FTFAllocUnitInfo)</td>
<td>Determines the target file’s OS/390 allocation. <strong>Valid values:</strong> CYL (cylinder), BLK (block), and TRK (track)</td>
</tr>
<tr>
<td>primaryAllocSize (FTFLONG)</td>
<td>Determines the target file’s OS/390 primary allocation unit size.</td>
</tr>
<tr>
<td>secondaryAllocSize (FTFLONG)</td>
<td>Determines the target file’s OS/390 secondary allocation unit size.</td>
</tr>
</tbody>
</table>
| textWrapRecord (FTFTextFileWrapInfo) | Indicates how records are processed when they are longer than the specified target-file record length. **Valid values:**  
  - FTFREQ_WRAP (wraps records)  
  - FTFREQ_FAIL (causes the data transfer to fail)  
  - FTFREQ_TRUNC (truncates records at the specified record length) |
| createDirectory (FTFBOOL)       | Determines whether the specified target directory is created when it does not already exist. **Valid values:** 1 (create directory), 0 (do not create a directory). |
C Data Structures

FTFExitTargetFileInfo

### 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 Tivoli Data Exchange configuration file fill in the missing entries. If items are specified in the Tivoli Data Exchange configuration file at the source Tivoli Data Exchange Sender's node or the target Tivoli Data Exchange 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 Tivoli Data Exchange Sender. Any outstanding items are attempted for resolution at the Tivoli Data Exchange 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
C Data Structures

**FTFExitTargetFileInfo**

Related Data Structures

- FTFExitAuthInfo (page 171)
- FTFExitInfo (page 177)
- FTFExitJobInfo (page 181)
- FTFExitQMgrsInfo (page 184)
- FTFExitSourceFileInfo (page 188)
- FTFExitUserInfo (page 197)
- FTFTargetFileInfo (page 251)

Data Structure

```c
typedef struct _FTFExitTargetFileInfo {
    FTFCHAR *pFileName;
    FTFBOOL isCompressed;
    FTFFileTypeInfo fileType;
    FTF FileModeInfo fileMode;
    FTFFileOrgInfo fileOrg;
    FTFLONG cDirectoryBlocks;
    FTFRecordFormatInfo recordFormat;
    FTFLONG lrecl;
    FTFLONG blockSize;
    FTFCHAR unitName[FTFREQ_UNITNAME_SIZE];
    FTFCHAR volser[FTFREQ_VOLSER_SIZE];
    FTFAllocUnitInfo allocationUnit;
    FTFLONG primaryAllocSize;
    FTFLONG secondaryAllocSize;
    FTFTextFileWrapInfo textWrapRecord;
    FTFBOOL createDirectory;
    FTFBOOL isDataPersistent;
    FTFLONG bufNo;
} FTFExitTargetFileInfo;
```
**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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName (FTFCHAR[])</td>
<td>Fixed length array that determines the name of the group to which this data-transfer request belongs.</td>
</tr>
<tr>
<td>label (FTFCHAR[])</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 186)
- FTFUserInfo (page 256)
### Data Structure

```c
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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id1 (FTFCHAR[ ])</td>
<td>Fixed length array that contains the text data that was entered into the data identifier 1 field.</td>
</tr>
<tr>
<td>Id2 (FTFCHAR[ ])</td>
<td>Fixed length array that contains the text data that was entered into the data identifier 2 field.</td>
</tr>
<tr>
<td>Id3 (FTFCHAR[ ])</td>
<td>Fixed length array that contains the text data that was entered into the data identifier 3 field.</td>
</tr>
</tbody>
</table>

**Related Functions**

None.

**Related Data Structures**

None.

**Data Structure**

```c
typedef struct _FTFIdentifiersInfo {
  FTFCHAR Id1[FTF_IDENTIFIER_SIZE];
  FTFCHAR Id2[FTF_IDENTIFIER_SIZE];
  FTFCHAR Id3[FTF_IDENTIFIER_SIZE];
} FTFIdentifiersInfo;
```
C Data Structures

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 following table lists and describes the FTFJobInfo data structure’s elements.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>poolName (FTFCHAR[])</td>
<td>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 Tivoli Data Exchange User’s Guide, “About Pools.”</td>
</tr>
<tr>
<td>transferPriority (FTFINT)</td>
<td>Represents the priority of the current data-transfer request. <strong>Valid values:</strong> 1 (highest) - 5 (lowest)</td>
</tr>
<tr>
<td>numberOfRetries (FTFINT)</td>
<td>Determines the number of retry attempts for the transfer request.</td>
</tr>
<tr>
<td>expirationDateTime (FTFINT)</td>
<td>Represents the number of seconds between the current time and the time that the data-transfer request expires.</td>
</tr>
<tr>
<td>MQMsgSize (FTFINT)</td>
<td>Determines the size (in bytes) of each individual MQSeries message created and used during the data-transfer request’s execution.</td>
</tr>
<tr>
<td>isTrusted (FTFBOOL)</td>
<td>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 Tivoli Data Exchange protection), Tivoli Data Exchange does not perform syncpoint control. In the event of a failure, the data-transfer request is terminated with a failing notification. <strong>Valid values:</strong> 1 (trusted), 0 (not trusted). <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>reserved (FTFBOOL)</td>
<td>Reserved. Do not use this data element.</td>
</tr>
</tbody>
</table>
### C Data Structures

#### FTFJobInfo

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isStageOnly (FTFCHAR)</td>
<td>Determines that the current transaction is staged only. No destination is specified.</td>
</tr>
<tr>
<td>isStagedftfid (FTFBOOL)</td>
<td>This data element determines whether the pSource 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. <strong>Valid values</strong>: 1 (True, use the FTFID), any other value (False, do not use the FTFID)</td>
</tr>
<tr>
<td>isStagedFile (FTFBOOL)</td>
<td>This data element determines whether the pSource 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. <strong>Valid values</strong>: 1 (True, use the filename), any other value (False, do not use the filename)</td>
</tr>
</tbody>
</table>
| cancelMode (FTFCancelMode) | Records how a pre-emptive cancellation should be handled. **Valid values**:  
  - FTF_PREEMPTCANCE_ _NEUTRAL (uses the cancellation value set in the Tivoli Data Exchange configuration file),  
  - FTF_PREEMPTCANCE_ _ON (allows the file transfer to be preemptively canceled),  
  - FTF_PREEMPTCANCE_ _OFF (does not allow preemptive cancellation) |
| delsrc (FTFBOOL)          | Indicates whether 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. |

### Related Functions

- FTFReq (page 151)

### Related Data Structures

- FTFExitJobInfo (page 181)
- FTFRequestMsgInfo (page 208)
C Data Structures

FTFJobInfo

Data Structure

typedef struct _FTFJobInfo {
    FTFCHAR    poolName [MQ_Q_NAME_LENGTH+1];
    FTFLONG    transferPriority;
    FTFLONG    numberOfRetries;
    FTFLONG    expirationDateTime;
    FTFLONG    MQMsgSize;
    FTFBOOL    isTrusted;
    FTFBOOL    reserved;
    FTFBOOL    isStageOnly;
    FTFBOOL    isStagedftfid;
    FTFBOOL    isStagedFile;
    FTFCancelMode cancelMode;
    FTFBOOL    delsrc;
    FTFIdentifiersInfo Ids;
    FTFBOOL    immed;
    FTFBOOL    isFormatXML;
} 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lqm (FTFCHAR[ ])</td>
<td>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.</td>
</tr>
<tr>
<td>oqm (FTFCHAR[ ])</td>
<td>Fixed length array that contains the name of the queue manager to which the Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>sqm (FTFCHAR[ ])</td>
<td>Fixed length array that contains the name of the queue manager to which the Tivoli Data Exchange Manager at the oqm forwards the ping message. This parameter is optional. If it is not provided, Tivoli Data Exchange uses the lqm value. In other words, the Tivoli Data Exchange Sender is on the lqm.</td>
</tr>
<tr>
<td>dqm (FTFCHAR[ ])</td>
<td>Fixed length array that contains the name of the queue manager to which the Tivoli Data Exchange Sender at the sqm forwards the ping message. This parameter is optional. If it is not provided, Tivoli Data Exchange uses the lqm value. In other words, the Tivoli Data Exchange Receiver is on the lqm.</td>
</tr>
<tr>
<td>cFile (FTFCHAR[ ])</td>
<td>Fixed length array that contains the fully qualified path to the Tivoli Data Exchange configuration file. This element is not required if you specify a configuration queue.</td>
</tr>
</tbody>
</table>
### C Data Structures

**FTFPingInfo**

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| timeout (FTFLONG) | Determines how long the FTFPing API should wait for the final response from the Tivoli Data Exchange Manager. This value is specified in seconds.  
**Valid values:** 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.  
**Valid values:** 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.  
**Valid values:** 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.  
**Valid values:** 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 149)

### 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;
**C Data Structures**

*FTFQMgrsInfo*

**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 Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 Tivoli Data Exchange subsystem. If this element is omitted, the requesting interface connects to the MQSeries default queue manager.  
  Note: This element is required on OS/390. |
| originating (FTFCHAR[ ])    | Fixed length array that contains the name of the originating queue manager on which the Tivoli Data Exchange data-transfer request originates. The Tivoli Data Exchange Manager must be running on this node. If this element is omitted, the Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange Receiver must be running on this node. |
C Data Structures

Related Functions

- FTFReq (page 151)

Related Data Structures

- FTFExitQMgrsInfo (page 184)
- FTFRequestMsgInfo (page 208)
- FTFStageMsgInfo (page 222)

Data Structure

```c
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 + LENGTH + 2];
    FTFCHAR target[MQ_Q_MGR_NAME_LENGTH + MQ_Q_MGR_NAME_LENGTH + 2];
} FTFQMgrsInfo;
```
**C Data Structures**

*FTFRequestMsgInfo*

**FTFRequestMsgInfo**

**Description**

The FTFRequestMsgInfo data structure contains input information required by the FTFReq API. It is also passed to the Tivoli Data Exchange Manager, Tivoli Data Exchange Sender, and Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qmgrs (FTFQMgrsInfo)</td>
<td>Contains the data structure that contains the local, originating, source, and destination queue managers used in the data-transfer request.</td>
</tr>
<tr>
<td>sourceFile (FTFSrcFileInfo)</td>
<td>Contains the data structure that contains information about the data-transfer request’s source file.</td>
</tr>
<tr>
<td>targetFile (FTFTgtFileInfo)</td>
<td>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.</td>
</tr>
<tr>
<td>job (FTFJobInfo)</td>
<td>Contains the data structure that contains the rules governing the data-transfer request.</td>
</tr>
<tr>
<td>userInfo (FTFUserInfo)</td>
<td>Contains the data structure that contains label and group information for the current data-transfer request.</td>
</tr>
<tr>
<td>isReply (FTFBOOL)</td>
<td>Determines whether Tivoli Data Exchange should wait for a reply before returning control to the calling application. <strong>Valid Values:</strong> True (wait for reply), another value (do not wait for reply).</td>
</tr>
<tr>
<td>timeout (FTFULONG)</td>
<td>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 Tivoli Data Exchange and MQSeries for the expiration of this request within the system.</td>
</tr>
</tbody>
</table>
### C Data Structures

**FTFRequestMsgInfo**

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replyWaitTime (FTFULONG)</td>
<td>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.</td>
</tr>
<tr>
<td>configFile (FTFCHAR *)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>cExitInfo (FTFLONG)</td>
<td>Determines the number of exits that to be invoked during the data-transfer request.</td>
</tr>
<tr>
<td>exitInfo (FTFEXITINFO *)</td>
<td>Pointer to a variable that contains a data structure that governs how user exits are invoked and executed.</td>
</tr>
<tr>
<td>as400 (FTFAS400FileInfo)</td>
<td>Contains a data structure that describes information about the target files to be written to the AS/400 platform.</td>
</tr>
<tr>
<td>ftfid (FTFCHAR[ ])</td>
<td>Fixed length array that contains a Tivoli Data Exchange identifier used to identify the current data-transfer request. The FTFReq API generates this unique identifier.</td>
</tr>
<tr>
<td>cq (FTFCHAR *)</td>
<td>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.</td>
</tr>
<tr>
<td>isDBCS (FTFBOOL)</td>
<td>Indicates whether DBCS processing is required.</td>
</tr>
</tbody>
</table>

### Related Functions

- FTFReq (page 151)

### Related Data Structures

- FTFAS400FileInfo (page 165)
- FTFExitInfo (page 177)
- FTFS400FileInfo (page 200)
- FTFQMgrsInfo (page 206)
- FTFSSourceFileInfo (page 216)
- FTFTargetFileInfo (page 251)
- FTFSUserInfo (page 256)
C Data Structures

FTFRequestMsgInfo

Data Structure

typedef struct _FTFRequestMsgInfo {
    FTFQMgrsInfo qmgrs;
    FTFSourceFileInfo sourceFile;
    FTFTargetFileInfo targetFile;
    FTFJobInfo job;
    FTFUserInfo userInfo;
    FTFBOOL isReply;
    FTFULONG timeout;
    FTFULONG replyWaitTime;
    FTFCHAR *configFile;
    FTFLONG cExitInfo;
    FTFEXITINFO *exitInfo;
    FTFAS400FileInfo as400;
    FTFCHAR ftfid[FTF_ID_SIZE];
    FTFCHAR *cq;
    FTFBOOL isDBCS;
} FTFRequestMsgInfo;
C Data Structures

FTFShutdownInfo

Description

The FTFShutdownInfo data structure contains the input information required for the FTFShutdown API. The FTFShutdown API will shuts down specified Tivoli Data Exchange components and tracks which components have actually shut down.

Data Elements

The following table lists and describes the FTFShutdownInfo data structure’s elements.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lqm (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager to which the FTFShutdown API has a connection handle.</td>
</tr>
<tr>
<td>node (FTFCHAR)</td>
<td>Determines the node to which the FTFShutdown API directs its request.</td>
</tr>
<tr>
<td>component (FTFCOMPONENT)</td>
<td>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.</td>
</tr>
<tr>
<td>immediate (FTFBOOL)</td>
<td>Determines whether Tivoli Data Exchange performs a queisce or immediate shutdown. A quiesce shutdown (the default) waits for all work to finish before the shutdown request is processed. An immediate shutdown pre-empts any work that is pending and immediately shuts down the specified component(s). <strong>Valid values:</strong> 1 (immediate), 0 (quiese).</td>
</tr>
<tr>
<td>timeout (FTFULONG)</td>
<td>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 Tivoli Data Exchange shutdown API returns the status. Otherwise, a timeout condition occurs. <strong>Valid values:</strong> 0 to ULONGMAX.</td>
</tr>
</tbody>
</table>
C Data Structures

FTFShutdownInfo

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configFile (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>cq (FTFCHAR)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFShutdown (page 154)

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;
    FTBOOL immediate;
    FTULONG 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manager.replied (FTFBOOL)</td>
<td>Indicates whether the Tivoli Data Exchange Manager replied to the shutdown request. This data element is set to True if the Tivoli Data Exchange Manager has replied and False if it has not. <strong>Valid values:</strong> 1 (Tivoli Data Exchange Manager has replied), 0 (Tivoli Data Exchange Manager has not replied).</td>
</tr>
<tr>
<td>sender.cSender (FTFLONG)</td>
<td>Indicates the number of Tivoli Data Exchange Senders that should respond to the shutdown request. This value is determined by the number of instances specified in the Tivoli Data Exchange configuration file. <strong>Valid values:</strong> 0 to ULONGMAX.</td>
</tr>
<tr>
<td>sender.cReply (FTFLONG)</td>
<td>Indicates the number of Tivoli Data Exchange Senders that have replied to the shutdown request.</td>
</tr>
<tr>
<td>receiver.cReceiver (FTFLONG)</td>
<td>Indicates the number of Tivoli Data Exchange Receivers that should respond to the shutdown request. This value is determined by the number of instances specified in the Tivoli Data Exchange configuration file. <strong>Valid values:</strong> 0 to ULONGMAX.</td>
</tr>
<tr>
<td>receiver.cReply (FTFLONG)</td>
<td>Indicates the number of Tivoli Data Exchange Receivers that have replied to the shutdown request.</td>
</tr>
<tr>
<td>statd.cStatd (FTFLONG)</td>
<td>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 Tivoli Data Exchange configuration file. <strong>Valid values:</strong> 0 to ULONGMAX.</td>
</tr>
</tbody>
</table>
## C Data Structures

### FTFShutdownReply

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statd.cReply (FTFLONG)</td>
<td>Indicates the number of status offload daemons that have replied to the shutdown request.</td>
</tr>
</tbody>
</table>
| logger.cLogger (FTFLONG) | Indicated the number of FTFLOG processes that should respond to the shutdown request.  
**Valid values:** 0 to ULONGMAX |
| logger.cReply (FTFLONG) | Indicates the number of FTFLOG processes that have replied to the shutdown request. |

### Related Functions

- FTFShutdown (page 154)

### Related Data Structures

None

### Data Structure

```c
typedef struct _FTFShutdownReply {
    struct {
        FTFBOOL 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;
```
} logger;
} FTFShutdownReply;
C Data Structures
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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilename (FTFCHAR*)</td>
<td>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.</td>
</tr>
<tr>
<td>isStaged (FTFBOOL)</td>
<td>Determines whether the file is to 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 transmitted to each of its destinations. <strong>Valid values:</strong> 1 (staging required), 0 (staging not required).</td>
</tr>
<tr>
<td>isStagePersistent (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (staging queue data is persistent), 0 (staging queue data is not persistent).</td>
</tr>
<tr>
<td>isDataPersistent (FTFBOOL)</td>
<td>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. <strong>Valid values:</strong> 1 (transmission is persistent), 0 (transmission not persistent).</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Determines whether the file being sent is compressed using the Tivoli Data Exchange internal compression algorithm. <strong>Valid values:</strong> 1 (compression is used) 0 (compression is not used)</td>
</tr>
<tr>
<td>isDelete (FTFBOOL)</td>
<td>Used internally by Tivoli Data Exchange.</td>
</tr>
</tbody>
</table>
### FTFSourceFileInfo

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileType (FTFFieldType)</td>
<td>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. <strong>Valid values:</strong> FTF_BINARY (binary), FTF_TEXT (text).</td>
</tr>
<tr>
<td>filePad (FTFFieldPad)</td>
<td>Enumerated data type that determines whether padding takes place. <strong>Valid values:</strong> FTFREQ_PAD, FTFREQ_NOPAD</td>
</tr>
<tr>
<td>padChar (FTFFIELD)</td>
<td>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. <strong>Valid values:</strong> A-Z, a-z, 0-9, space, 0x00-0xFF (where the last two positions are a hexadecimal value mapped to a character). <strong>Default value:</strong> space</td>
</tr>
<tr>
<td>sType (FTFFIELD)</td>
<td>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 Tivoli Data Exchange configuration file.</td>
</tr>
<tr>
<td>cstypeData (FTFFIELD)</td>
<td>Determines the length in bytes of the *pstypeData data element.</td>
</tr>
<tr>
<td>pstypeData (FTFFIELD*)</td>
<td>Contains the source data that is not contained in a file.</td>
</tr>
<tr>
<td>bufNo (FTFFIELD)</td>
<td>Determines the number of buffers used during a data-transfer request on the OS/390 platform.</td>
</tr>
</tbody>
</table>

### Related Functions

None

### Related Data Structures

- FTFExitSourceFileInfo (page 188)
- FTFRequestMsgInfo (page 208)
C Data Structures

FTFSourceFileInfo

Data Structure

typedef struct _FTFSourceFileInfo {
    FTFCHAR     *pFileName;
    FTFBOOL     isStaged;
    FTFBOOL     isStagePersistent;
    FTFBOOL     isDataPersistent;
    FTFBOOL     isCompressed;
    FTFBOOL     isDelete;
    FTFFInfoType fileTypeInfo;
    FTFRecordPadInfo filePad;
    FTFBYTE     padChar;
    FTFCHAR     stype [FTF_FILETYPE_SIZE];
    FTFLONG     cstypeData;
    FTFCHAR     *pstypeData;
    FTFLONG     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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cRows (FTFLONG)</td>
<td>Contains the number of staged items in the query return list.</td>
</tr>
<tr>
<td>pRows (<em>FTFStagedTransactionMsgInfo</em>)</td>
<td>Contains the address of the variable that contains the first record returned from the query request.</td>
</tr>
</tbody>
</table>

**Related Functions**

- FTFStage (page 156)

**Related Data Structures**

- FTFStagedTransactionMsgInfo (page 220)

**Data Structure**

```c
typedef struct _FTFStagedListMsgInfo
{
    FTFLONG cRows;
    FTFStagedTransactionMsgInfo *pRows;
} FTFStagedListMsgInfo;
```
**C Data Structures**

*FTFStagedTransactionMsgInfo*

## 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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftfid[FTF_ID_SIZE] (FTFCHAR)</td>
<td>Contains the staged transaction’s FTFID.</td>
</tr>
<tr>
<td>sqm[FTF_NAME_SIZE] (FTFCHAR)</td>
<td>Contains the name of the staged transaction’s source queue manager.</td>
</tr>
<tr>
<td>pSourceNameFile (FTFCHAR *)</td>
<td>Contains the name of the staged file.</td>
</tr>
<tr>
<td>stagedTimeStamp (FTFTS)</td>
<td>Contains the time and date the file was staged.</td>
</tr>
<tr>
<td>MQMsgSize (FTFLONG)</td>
<td>Contains the size of the data messages that compose the staged file.</td>
</tr>
<tr>
<td>cDataBlocks (FTFLONG)</td>
<td>Contains the number of data messages that compose the staged file.</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Determines whether the staged messages are compressed.</td>
</tr>
<tr>
<td><strong>Valid values:</strong> True (compressed), False (not compressed)</td>
<td></td>
</tr>
</tbody>
</table>

### Related Functions

- FTFStage (page 156)
Related Data Structures

- FTFStagedListMsgInfo (page 219)

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;
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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qmgrs (FTFQMgrsInfo)</td>
<td>Contains the data structure that contains the local, originating, source, and destination queue managers used by the FTFStage API.</td>
</tr>
<tr>
<td>pConfigFile (FTFCHAR*)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange 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.</td>
</tr>
<tr>
<td>pFile (FTFCHAR*)</td>
<td>Pointer to the variable that contains the FTFID or the filename of the item to be purged from the staging area.</td>
</tr>
<tr>
<td>isReply (FTFCHAR)</td>
<td>Determines whether a reply will be generated after the purge or query operation takes place.</td>
</tr>
<tr>
<td>replyWaitTime (FTFLONG)</td>
<td>Determines the time to wait for a reply from a query request. This value does not apply to purge requests.</td>
</tr>
</tbody>
</table>
C Data Structures

FTFStageMsgInfo

Related Functions

- FTFStage (page 156)

Related Data Structures

- FTFQMgrsInfo (page 206)

Data Structure

typedef struct _FTFStageMsgInfo {
  FTFQMgrsInfo qmgrs;
  FTFCHAR *pConfigFile;
  FTFCHAR *pFile;
  FTFCHAR isReply;
  FTFLONG replyWaitTime;
  FTFStageReq action;
  FTFCHAR* cq;
} FTFStageMsgInfo;

Name (Data Type) | Description
--- | ---
action (FTFStageReq) | Determines the type of action to be carried out.  
**Valid values:**  
- FTFSTAGE_QUERY – If value is 1, the action is to query the staging queue.  
- FTFSTAGE_PURGE_FILE – If the value is 2, the action is to purge a specific file from the staging queue.  
- FTFSTAGE_PURGE_FTFID – If the value is 3, the action is to purge entries for a specific FTFID.  
- FTFSTAGE_PURGE_ALL – If the value is 4, the action is to purge all files from the staging queues.
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.
C Data Structures

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.

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftfId (FTFCHAR)</td>
<td>Pointer to a variable that contains the unique FTFID that matches the data-transfer request for which the detailed status was retrieved.</td>
</tr>
<tr>
<td>timestamp (FTFTS)</td>
<td>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.</td>
</tr>
<tr>
<td>localQmgr (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager to which the reporting component has a connection handle.</td>
</tr>
<tr>
<td>component (FTFComponent)</td>
<td>Contains the type of the Tivoli Data Exchange component that issued the detailed status message.</td>
</tr>
<tr>
<td>status (FTFStatus)</td>
<td>Contains the status associated with the Tivoli Data Exchange component 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 Tivoli Data Exchange subsystem.</td>
</tr>
<tr>
<td>ftfcode (FTFCODE)</td>
<td>Contains the primary processing or return code. The last three digits correspond to the Tivoli Data Exchange return codes outlined in this guide and defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>src (FTFCODE)</td>
<td>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 Tivoli Data Exchange error occurs it may be a Tivoli Data Exchange return code.</td>
</tr>
<tr>
<td>errorText (FTFCHAR)</td>
<td>Pointer to the variable that contains message text associated with an error.</td>
</tr>
</tbody>
</table>
C Data Structures

**FTFStatDetail**

<table>
<thead>
<tr>
<th>Name (Data Type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curMsgNumber (FTF_LONG)</td>
<td>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 Tivoli Data Exchange Sender and Tivoli Data Exchange Receiver.</td>
</tr>
<tr>
<td>totalMsgCount (FTF_LONG)</td>
<td>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 Tivoli Data Exchange Sender and Tivoli Data Exchange Receiver.</td>
</tr>
<tr>
<td>CustomComponent (FTF_CHAR)</td>
<td>Can contain user-specified custom component information submitted to the Tivoli Data Exchange status subsystem.</td>
</tr>
<tr>
<td>CustomStatusType (FTF_CHAR)</td>
<td>Can contain user-specified custom status information submitted to the Tivoli Data Exchange status subsystem.</td>
</tr>
</tbody>
</table>

**Related Functions**

- FTFStatusFreeDetailList (page 160)
- FTFStatusGetDetailList (page 164)

**Related Data Structures**

- FTFStatDetailList (page 234)

**Data Structure**

```c
typedef struct _FTFStatDetail {
    FTFCHAR ftfId[FTF_ID_SIZE];
    FTFTS timestamp;
    FTFCHAR localQmgr[FTF_NAME_SIZE];
    FTFComponent component;
    FTFStatus status;
    FTFCODE ftfcode;
    FTFCODE src;
    FTFCHAR errorText[FTF_MAX_ERR_TXT_SIZE];
    FTFLONG curMsgNumber;
    FTFLONG totalMsgCount;
    FTFCHAR CustomComponent [FTF_MAX_CUSTOM_CPT_TXT_SIZE];
} FTFStatDetail;
```
### FTFStatDetail

```c
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 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 FTFComponent 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 FTFComponent.

```c
// 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);
C Data Structures

FTFStatDetail

/***************************************************************/
/* 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 Manager status of COMPLETE or FAILED this will be the */
/* final update. If this status doesn't exist for the Manager */
/* 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("%s\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);
C Data Structures

FTFStatDetail

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.

```c
C Data Structures
FTFStatDetail

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.
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.

```c
/******************************************************************************
/* Define variables. */
/******************************************************************************
FTFCALL 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);
    }
}
```
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cRows (FTFLONG)</td>
<td>Contains the number of detailed records that have been returned.</td>
</tr>
<tr>
<td>pRows (FTFStatDetail*)</td>
<td>Pointer a variable that contains to the FTFStatDetail data structure which contains detail status records operated during a data-transfer request.</td>
</tr>
</tbody>
</table>

### Related Functions

- FTFStatusFreeDetailList (page 160)
- FTFStatusGetDetailList (page 164)

### Related Data Structures

- FTFStatDetail (page 224)

### Data Structure

```c
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 Tivoli Data Exchange component used during the associated data-transfer request. It contains the unique FTFID and a section for each components: the Tivoli Data Exchange Manager, the Tivoli Data Exchange Sender, and the Tivoli Data Exchange Receiver.

Data Elements

The following table lists and describes the FTFStatSummary data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftfId (FTFCHAR)</td>
<td>Pointer to a variable that contains the unique FTFID for the data-transfer request for which summary status information was retrieved.</td>
</tr>
<tr>
<td>timestamp (FTFTS)</td>
<td>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.</td>
</tr>
<tr>
<td>ftfcode (FTFCODE)</td>
<td>Contains the primary processing or return code. The last three digits of this code correspond to the Tivoli Data Exchange return codes outlined in the Tivoli Data Exchange Errors and Code Guide.</td>
</tr>
<tr>
<td>src (FTFCODE)</td>
<td>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 Tivoli Data Exchange error occurs it may be a Tivoli Data Exchange return code.</td>
</tr>
<tr>
<td>localQmgr (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager to which the reporting component has a connection handle.</td>
</tr>
<tr>
<td>originatingQmgr (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager to which the data-transfer request’s Tivoli Data Exchange Manager is connected.</td>
</tr>
</tbody>
</table>
## C Data Structures

### FTFStatSummary

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originatingQmgrStatus (FTFStatus)</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Manager for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>originatingQmgrTimestamp (FTFTS)</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Manager. The FTFTS data type is defined in the ftfc.h header file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>sourceQmgr (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager where the source file was processed by the Tivoli Data Exchange Sender for the data-transfer request associated with the FTFID.</td>
</tr>
<tr>
<td>sourceQmgrStatus (FTFStatus)</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Sender for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>sourceQmgrTimestamp (FTFTS)</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Sender. The FTFTS data type is defined in the ftfc.h header file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>sourceFilename (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path and filename of the source file used in the data-transfer request.</td>
</tr>
<tr>
<td>targetQmgr (FTFCHAR)</td>
<td>Pointer to a variable that contains the name of the queue manager to which data is sent. The Tivoli Data Exchange Receiver is attached to this queue manager.</td>
</tr>
<tr>
<td>targetQmgrStatus (FTFStatus)</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Receiver for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>targetQmgrTimestamp (FTFTS)</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Receiver. The FTFTS data type is defined in the ftfc.h header file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>targetFilename (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path and filename of the target file used in the data-transfer request.</td>
</tr>
</tbody>
</table>
C Data Structures

FTFStatSummary

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestingQmgrStatus (FTFStatus)</td>
<td>Contains the most recent status update from the component that requested the data-transfer. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>requestingQmgrTimestamp (FTFTS)</td>
<td>Contains the date and time of the status update from the Tivoli Data Exchange transaction requesting component. The FTFTS data type is defined in the ftfc.h header file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>groupName (FTFCHAR)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>label (FTFCHAR)</td>
<td>Contains the user-defined label that is associated with the data-transfer request.</td>
</tr>
<tr>
<td>Ids (FTFIdentifiersInfo)</td>
<td>Contains the three user-defined fields which are associated with the data-transfer request.</td>
</tr>
<tr>
<td>priority (FTFLONG)</td>
<td>Contains the priority code associated with the data-transfer request.</td>
</tr>
<tr>
<td>persistent (FTFBOOL)</td>
<td>Contains the persistence code associated with the data-transfer request.</td>
</tr>
<tr>
<td>bytesSent(FTFLONG)</td>
<td>Contains a count of the number bytes of data in the data-transfer request.</td>
</tr>
<tr>
<td>numberOfMessages (FTulong)</td>
<td>Contains a count of the number of message that make up the data-transfer request.</td>
</tr>
<tr>
<td>transferTime (FTFLONG)</td>
<td>Contains the time, in seconds, that has elapsed since the data-transfer request has begun.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFStatusFreeSummaryList (page 162)
- FTFStatusGetSummaryList (page 166)

Related Data Structures

- FTFStatusSummaryFilter (page 247)
C Data Structures

FTFStatSummary

Data Structure

typedef struct _FTFStatSummary {
    FTFCHAR ftfId[FTF_ID_SIZE];
    FTFTS 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 Tivoli Data Exchange component.

Data Elements

The following table lists and describes the FTFStatSummaryList data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cRows (FTFLONG)</td>
<td>Contains the number of returned data-transfer request summary records.</td>
</tr>
<tr>
<td>pRows (FTFStatSummary*)</td>
<td>Pointer to a variable that contains the FTFStatSummary data type.</td>
</tr>
<tr>
<td></td>
<td>The FTFStatSummary data structure contains data-transfer request summary status records.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFStatusFreeSummaryList (page 162)
- FTFStatusGetSummaryList (page 166)

Related Data Structures

- FTFStatSummary (page 235)
- FTFStatusSummaryFilter (page 247)
C Data Structures
FTFStatSummaryList

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lqm (FTFCHAR)</td>
<td>Contains the name of the lqm from which the delete operation occurs and for which status records are deleted.</td>
</tr>
<tr>
<td>cFile (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange configuration file. This element is not required if you specify a configuration queue.</td>
</tr>
<tr>
<td>cq (FTFCHAR)</td>
<td>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.</td>
</tr>
<tr>
<td>ftfid (FTFCHAR)</td>
<td>Contains the FTFID for which status information is deleted.</td>
</tr>
</tbody>
</table>

**Related Functions**

- FTFStatusDelete (page 158)

**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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lqm (FTFCHAR)</td>
<td>Contains the name of the lqm from which the set status operations occurs and for which detailed status information is retrieved.</td>
</tr>
<tr>
<td>cFile (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange configuration file. This element is not required if you specify a configuration queue.</td>
</tr>
<tr>
<td>cq (FTFCHAR)</td>
<td>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.</td>
</tr>
<tr>
<td>ftfid (FTFCHAR)</td>
<td>Contains the FTFID for which detail status information is retrieved.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFStatusGetDetailList (page 164)

Related Data Structures

None
C Data Structures

FTFStatusGetDetailListInfo

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lqm (FTFCHAR)</td>
<td>Contains the name of the lqm from which the set status operations occurs and for which detailed status information is retrieved.</td>
</tr>
<tr>
<td>cFile (FTFCHAR)</td>
<td>Pointer to a variable that contains the fully qualified path to the Tivoli Data Exchange configuration file. This element is not required if you specify a configuration queue.</td>
</tr>
<tr>
<td>cq (FTFCHAR)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFStatusGetSummaryList (page 166)

Related Data Structures

None
C Data Structures

FTFStatusGetSummaryListInfo

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 Tivoli Data Exchange 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 following table lists and describes the FTFStatusSummaryFilter data structure's elements.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftfId (FTFCCHAR)</td>
<td>Contains the unique FTFID that matches the data-transfer request for which status information is retrieved.</td>
</tr>
<tr>
<td>statusIgnoreActive (FTFLONG)</td>
<td>Contains a flag that tells the API whether to ignore data-transfer request statuses with a type of Active. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>statusIgnoreComplete (FTFLONG)</td>
<td>Contains a flag that tells the API to ignore data-transfer request statuses with a type of Complete. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>statusIgnoreFailed (FTFLONG)</td>
<td>Contains a flag that tells the API to ignore data-transfer request statuses with a type of Failed. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>statusIgnoreCancelled (FTFLONG)</td>
<td>Contains a flag that tells the API to ignore data-transfer request statuses with a type of Canceled. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
</tbody>
</table>
## C Data Structures

### FTFStatusSummaryFilter

<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
</table>
| statusIgnoreExpired (FTFLONG) | Contains a flag that tells the API to ignore data-transfer request statuses with a type of *Expired*.  
**Valid values:** 1 (ignore this status type), 0 (include this status type). |
| localQmgr (FTFCHAR)           | Reserved.                                                                   |
| caseSensitive (FTFBOOL)       | Contains 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 Tivoli Data Exchange 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.  
**Note:** This field is mandatory.  
**Format:** 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.  
**Format:** YYYYMMDDhhmmss |
| sourceQmgr(FTFCHAR)           | Pointer to a variable that contains the name of the sqm. (The Tivoli Data Exchange 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 Tivoli Data Exchange Sender. Only records with a value greater than the value stored in this data element are included in the returned status information.  
**Format:** YYYYMMDDhhmmss |
<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
</table>
| sourceQmgrEndTimestamp(FTFCHAR) | Pointer to a variable that contains the latest time and date at which the data-transfer request started at the Tivoli Data Exchange Sender. Only records with a value less than the value stored in this data element are included in the returned status information.  
**Format:** YYYMMDDhhmmss |
| 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 Tivoli Data Exchange 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 Tivoli Data Exchange Receiver. Only records with a value greater than the value stored in this data element are included in the returned status information.  
**Format:** YYYMMDDhhmmss |
| targetQmgrEndTimestamp(FTFCHAR) | Pointer to a variable that contains the latest time and date at which the data-transfer request started at the Tivoli Data Exchange Receiver. Only records with a value less than the value stored in this data element are included in the returned status information.  
**Format:** YYYMMDDhhmmss |
| targetFilename(FTFCHAR) | Pointer 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) | Represents a 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. |
C Data Structures

FTFStatusSummaryFilter

Related Functions

- FTFStatusGetSummaryList (page 166)

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 originatingTimestamp[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;
**FTFTargetFileInfo**

**Description**

The FTFTargetFileInfo data structure contains the information required for describing the target file and its location, attributes, and allocation rules.

<table>
<thead>
<tr>
<th>Specifying an Esoteric Unit Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>To specify an esoteric name for the OS/390 UNIT value, follow these steps:</td>
</tr>
<tr>
<td>• Do not set a value in the MVSVOLUME stanza.</td>
</tr>
<tr>
<td>• Specify the unit value in the unitName data element or in the MVSUNITNAME stanza in the Tivoli Data Exchange configuration file on either the Tivoli Data Exchange Sender or the Tivoli Data Exchange Receiver.</td>
</tr>
</tbody>
</table>

**Data Elements**

The following table lists and describes the FTFTargetFileInfo data structure’s elements:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFileName (FTFCHAR*)</td>
<td>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.</td>
</tr>
<tr>
<td>isCompressed (FTFBOOL)</td>
<td>Reflects whether inbound data is compressed or uncompressed. Valid values: 1 (compressed), 0 (uncompressed).</td>
</tr>
<tr>
<td>fileType (FTFFileTypeInfo)</td>
<td>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. Valid values: FTF_BINARY (binary), FTF_TEXT (text).</td>
</tr>
</tbody>
</table>
### C Data Structures

#### FTTTargetFileInfo

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileMode (FTFFileModeInfo)</td>
<td>Determines how the data is processed at the destination. <strong>Valid values:</strong> - FTF_CREATE_FILE (create a new file), - FTF_APPEND_FILE (append to an existing file), - FTF_NOREPLACE_FILE (do not replace an existing file).</td>
</tr>
<tr>
<td>fileOrg (FTFFileOrgInfo)</td>
<td>Determines the target file’s OS/390 file organization. <strong>Valid values:</strong> PS (Physical Sequential), PDS (Partitioned Data Set) <strong>Note:</strong> See the following paragraph, <em>OS/390 File Allocation Items</em>.</td>
</tr>
<tr>
<td>cDirectoryBlocks (FTFLONG)</td>
<td>Determines the number of directory blocks allocated for the target file on OS/390.</td>
</tr>
<tr>
<td>recordFormat (FTFRecordFormatInfo)</td>
<td>Determines the target file’s OS/390 record format. <strong>Valid values:</strong> F (fixed), V (variable), FB (fixed block), and VB (variable block) <strong>Note:</strong> See the following paragraph, <em>OS/390 File Allocation Items</em>.</td>
</tr>
<tr>
<td>lrecl (FTFLONG)</td>
<td>Determines the target file’s OS/390 logical record length. <strong>Valid values:</strong> 1 - 32760. <strong>Note:</strong> See the following paragraph, <em>OS/390 File Allocation Items</em>.</td>
</tr>
<tr>
<td>blockSize (FTFLONG)</td>
<td>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. <strong>Valid values:</strong> 0 - 32760</td>
</tr>
<tr>
<td>unitName (FTFCHAR)</td>
<td>Determines the target file’s OS/390 unit name.</td>
</tr>
<tr>
<td>volser (FTFCHAR)</td>
<td>Determines the target file’s volume serial number.</td>
</tr>
<tr>
<td>allocationUnit (FTFAlocUnitInfo)</td>
<td>Determines the target file’s OS/390 allocation unit. <strong>Valid values:</strong> CYL (cylinder), BLK (block), TRK (track)</td>
</tr>
<tr>
<td>primaryAllocSize (FTFLONG)</td>
<td>Determines the target file’s OS/390 primary allocation unit size.</td>
</tr>
<tr>
<td>secondaryAllocSize (FTFLONG)</td>
<td>Determines the target file’s OS/390 secondary allocation unit size.</td>
</tr>
</tbody>
</table>
### C Data Structures

**FTFTargetFileInfo**

<table>
<thead>
<tr>
<th>Name (FTFTextFileWrapInfo)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>textWrapRecord</td>
<td>Indicates how records are processed when they are longer than the specified target-file record length. <strong>Valid values:</strong>  - FTFREQ_WRAP (wraps records)  - FTFREQ_FAIL (causes the data transfer to fail)  - FTFREQ_TRUNC (truncates records at the specified record length)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name (FTFBOOL)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDirectory</td>
<td>Determines whether the specified target directory is created when it does not already exist. <strong>Valid values:</strong> 1 (create directory), 0 (do not create a directory).</td>
</tr>
<tr>
<td>isDataPersistent</td>
<td>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. <strong>Valid values:</strong> 1 (transmission is persistent), 0 (transmission not persistent).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name (FTFCHAR)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDataset</td>
<td>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. <strong>Valid values:</strong> up to 44 characters</td>
</tr>
<tr>
<td>dType (FTFCHAR)</td>
<td>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 Tivoli Data Exchange configuration file.</td>
</tr>
<tr>
<td>cdtypeData (FTFLONG)</td>
<td>Determines the length in bytes of the *pdtypeData data element.</td>
</tr>
<tr>
<td>pdtypeData (FTFCHAR*)</td>
<td>Contains the receiver data that will not be stored in a file.</td>
</tr>
<tr>
<td>bufNo (FTFLONG)</td>
<td>Determines the number of buffers used during a data transfer request on the OS/390 platform.</td>
</tr>
</tbody>
</table>
**C Data Structures**

*FTFTargetFileInfo*

---

**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 Tivoli Data Exchange configuration file fill in the missing entries. If items are specified in the Tivoli Data Exchange configuration file at the source Tivoli Data Exchange Sender's node or the target Tivoli Data Exchange 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 Tivoli Data Exchange Sender. Any outstanding items are attempted for resolution at the Tivoli Data Exchange Receiver.

---

**Related Functions**

- FTFReq (page 151)

---

**Related Data Structures**

- FTFExitRequestInfo (page 186)
- FTFRequestMsgInfo (page 208)

---

**Data Structure**

```c
typedef struct _FTFTargetFileInfo {
    FTFCHAR       * pFileName;
    FTFBOOL       isCompressed;
    FTFFileTypeInfo   fileType;
    FTFFileModeInfo   fileMode;
    FTFFileOrgInfo   fileOrg;
    FTFLONG        cDirectoryBlocks;
    FTFRecordFormatInfo   recordFormat;
    FTFLONG        lrecl;
    FTFLONG        blockSize;
    FTFCHAR     unitName [FTFREQ_UNITNAME_SIZE];
    FTFCHAR     volser [FTFREQ_VOLSER_SIZE];
    FTFAllocUnitInfo    allocationUnit;
    FTFLONG        primaryAllocSize;
    FTFLONG        secondaryAllocSize;
    FTFTextFileWrapInfo   textWrapRecord;
    FTFBOOL       createDirectory;
} FTFTargetFileInfo;
```
C Data Structures

FTFTargetFileInfo

FTFBOOL isDataPersistent;
FTFCHAR modelDataset[FTFREQ_DATASETNAME_SIZE];
FTFCHAR dtype [FTF_FILETYPE_SIZE];
FTFLONG cdtypeData;
FTFCHAR *pdtypeData;
FTFLONG 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName (FTFCHAR)</td>
<td>Represents the name of the group to which this data-transfer request belongs.</td>
</tr>
<tr>
<td>label (FTFCHAR)</td>
<td>Represents a user-specified label that can be used in Tivoli Data Exchange status queries. This value has no bearing on the internal processing of data-transfer request. It is used for output and query specifications.</td>
</tr>
<tr>
<td>replyQ (FTFCHAR)</td>
<td>Represents the queue to which reply messages are to be routed.</td>
</tr>
<tr>
<td>replyQmgr (FTFCHAR)</td>
<td>Represents the queue manager to which reply messages are to be routed.</td>
</tr>
<tr>
<td>notifyStatus (FTFNotifyStatus)</td>
<td>Defines when a notification message will be sent to the Notify Queue. If you specify this argument, you must also specify Notify Data and Notify Type arguments.</td>
</tr>
<tr>
<td>notifyType (FTFCHAR)</td>
<td>Specifies the user-defined method used to deliver a notification message. If you specify this argument, you must also specify Notify Data and Notify Status arguments.</td>
</tr>
<tr>
<td>pNotifyData (FTFCHAR*)</td>
<td>A 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 Notify Status and Notify Type arguments.</td>
</tr>
</tbody>
</table>

Related Functions

- FTFReq (page 151)
Related Data Structures

- FTFExitUserInfo (page 197)
- FTFRequestMsgInfo (page 208)

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;
} FTFUserInfo;
C Data Structures

FTFUserInfo
This chapter describes the Tivoli Data Exchange COBOL API, which allows COBOL developers in an OS/390 environment to interface with the Tivoli Data Exchange C API from a COBOL application.

**Note:**
- Tivoli Data Exchange currently supports IBM for OS/390 and VM, version 2.4 and higher.

This chapter includes the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMNULL</td>
<td>261</td>
</tr>
<tr>
<td>SETADDR</td>
<td>263</td>
</tr>
<tr>
<td>FTFCAN</td>
<td>265</td>
</tr>
<tr>
<td>FTFEND</td>
<td>268</td>
</tr>
<tr>
<td>FTFPING</td>
<td>271</td>
</tr>
<tr>
<td>FTFREQ</td>
<td>274</td>
</tr>
<tr>
<td>FTFSL</td>
<td>277</td>
</tr>
<tr>
<td>Compiling the COBOL API</td>
<td>280</td>
</tr>
</tbody>
</table>
Utility Programs

This section describes the utility programs included with the Tivoli Data Exchange COBOL API. It includes the following information about each utility:

- Description
- Prototype
- Parameters

The following utilities are included with the COBOL API:

- TERMNULL (page 261)
- SETADDR (page 263)
TERMNULL

Description
The TERMNULL utility replaces blank pad characters with null characters, (x'00'). Tivoli Data Exchange C APIs expect strings to be delimited with nulls. The normal operation of COBOL pads fields with blanks. The blanks must be changed to nulls before the field can be passed to the Tivoli Data Exchange C APIs. This utility processes any string field up to 1024 bytes in length.

Example
In this example, QUEUE-MANAGER-NAME contains a string padded with blanks (x'40'). After returning from TERMNULL, the QUEUE-MANAGER-NAME contains a string padded with null characters.

The COBOL code in the following figure defines the input and output fields.

```
WORKING-STORAGE SECTION.
  01 INPUT-LENGTH          PIC S9(4) BINARY.
  01 QUEUE-MANAGER-NAME    PIC X(48).
```

The utility is invoked in the following figure.

```
MOVE +48 TO INPUT-LENGTH.
MOVE 'MQA2' TO QUEUE-MANAGER-NAME.
CALL 'TERMNULL' USING INPUT-LENGTH,
          QUEUE-MANAGER-NAME.
```
## Parameters

The following table contains TERMNULL’s address.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT-LENGTH</td>
<td>Length of the string that requires null termination.</td>
<td>The TERMNULL routine replaces blanks from right to left until a non-blank character is found.</td>
</tr>
<tr>
<td>QUEUE-MANAGER-NAME</td>
<td>This is a sample string that requires null termination.</td>
<td>Any alphanumeric field can be passed.</td>
</tr>
</tbody>
</table>
SETADDR

Description

C programming language functions use address pointers in structures as parameters. The SETADDR utility assists you in setting address pointers for use with the C API. It populates a pointer field to the address of a WORKING-STORAGE, 01-level data item. Any 01-level data item can be used and any pointer field can be used for the resulting address.

Example

In this example, CONFIG-FILE-POINTER contains the address of CONFIG-FILE-NAME after executing the subroutine. This provides the C routine with the address that it needs to process the contents of the field.

The following figure displays the COBOL code defines the field and its pointer.

WORKING-STORAGE SECTION.
01 CONFIG-FILE-POINTER USAGE IS POINTER.
01 CONFIG-FILE-NAME PIC X(256) VALUE "FTFHLQ.FTF.INI".

The utility is invoked in the following figure.

PROCEDURE DIVISION.
CALL 'SETADDR' USING CONFIG-FILE-NAME,
              CONFIG-FILE-POINTER.

Parameters

The following table contains SETADDR’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG-FILE-NAME</td>
<td>The field for which an address must be obtained.</td>
<td>This field can be any COBOL WORKING-STORAGE SECTION 01 level.</td>
</tr>
<tr>
<td>CONFIG-FILE-POINTER</td>
<td>This field receives the address of the desired field.</td>
<td>This field must be declared as USAGE IS POINTER.</td>
</tr>
</tbody>
</table>
COBOL API Functions

This section describes each COBOL API. It includes the following information about each function:

- Description
- Prototype
- Related Sample Program
- Parameters
- Related Data Structures
- Program Flow

After installation, the copybook that contains the data structures used by the COBOL API is located in the ++FTFHLQ++:FTF:COBSAMP PDS data set, where ++FTFHLQ++ represents the high-level qualifier for the Tivoli Data Exchange dataset. The copybook is located in the FTFICOPY member.

The COBOL API set contains the following APIs:

- FTFCAN (page 265)
- FTFEND (page 268)
- FTFPING (page 271)
- FTFREQ (page 274)
- FTFSL (page 277)
**FTFCAN**

**Description**

The FTFCAN function cancels a data-transfer request that has been submitted to the Tivoli Data Exchange subsystem. The cancel request must be submitted to the same Tivoli Data Exchange Manager as the initial data-transfer request. FTFCAN submits a cancel request to the system and does not wait for a response that the cancel occurred.

**Prototype**

```cobol
WORKING-STORAGE.
COPY FTFCOPY.
PROCEDURE DIVISION.
   CALL 'FTFCAN' USING BY VALUE CONNECTION-HANDLE
   BY REFERENCE FTF-CANCEL-INFO
   BY REFERENCE FTFCA.
```

**Related Sample Program**

- FTFCANCL
Parameters

The following table lists and describes the FTFCAN API’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION-HANDLE</td>
<td>CONNECTION-HANDLE</td>
<td>Represents the connection to the queue manager. The value of CONNECTION-HANDLE was returned by a previous call.</td>
</tr>
<tr>
<td>FTF-CANCEL-INFO</td>
<td>Address of FTF-CANCEL-INFO in the WORKING-STORAGE SECTION.</td>
<td>Represents a properly formatted data structure used by FTFCAN. This data structure contains all of the attributes and input criteria to cancel a data-transfer request that has already been submitted to the Tivoli Data Exchange subsystem.</td>
</tr>
<tr>
<td>FTFCA</td>
<td>Address of FTFCA in the WORKING-STORAGE SECTION.</td>
<td>Represents an output area that returns properly formatted API reply information. This information includes the return codes and optional messages.</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTFCA (page 298)
- FTF-CANCEL-INFO (page 300)

Program Flow

The program flow for the FTFCAN API is:
Get Input Parameters - The routine retrieves the values established in your calling program.

Connect to the Queue Manager - The utility routine establishes a connection to the MQSeries queue manager.

Set Up Parameters - The routine uses the values in your program to construct a command to issue to Tivoli Data Exchange.

Call FTFCAN - The program calls the FTFCAN API for execution.

Report Results - The results of the FTFCAN call are reported.

Disconnect from the Queue Manager - The utility routine breaks the connection with the MQSeries queue manager.
**COBOL API Reference**

**FTFEND**

**Description**

The FTFEND function shuts down specified Tivoli Data Exchange components. A shutdown request can target all components or a specific component on a local or remote queue manager. Before you can call the FTFEND API, you must set a value that indicates which component will be shut down.

The Tivoli Data Exchange components being shut down do not process the shutdown message while they are processing a data-transfer request. For example, if the Tivoli Data Exchange 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 Tivoli Data Exchange Sender by other means may cause the failure of the data-transfer request.

**Prototype**

```
WORKING-STORAGE.
    COPY FTFICOPY.
PROCEDURE DIVISION.
    CALL 'FTFEND' USING BY VALUE CONNECTION-HANDLE
    BY REFERENCE FTF-SHUTDOWN-INFO
    BY REFERENCE FTF-SHUTDOWN-REPLY
    BY REFERENCE FTFCA.
```

**Related Sample Program**

- FTFEND
# Parameters

The following table lists and describes the FTFEND API’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION-HANDLE</td>
<td>CONNECTION-HANDLE</td>
<td>Represents the connection to the queue manager established in the MQCONN call.</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO</td>
<td>Address of the Tivoli Data Exchange</td>
<td>Represents the input data structure that contains all the required and optional attributes related to the shutdown options.</td>
</tr>
<tr>
<td></td>
<td>SHUTDOWN-INFO data structure in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORKING-STORAGE SECTION.</td>
<td></td>
</tr>
<tr>
<td>FTF-REPLY-INFO</td>
<td>Address of the Tivoli Data Exchange</td>
<td>Represents the output data structure that contains all the output and results from the FTFEND API request.</td>
</tr>
<tr>
<td></td>
<td>FTF-SHUTDOWN-REPLY data structure in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORKING-STORAGE.</td>
<td></td>
</tr>
<tr>
<td>FTFCA</td>
<td>Address of the FTFCA data structure in the</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
<tr>
<td></td>
<td>WORKING-STORAGE SECTION.</td>
<td></td>
</tr>
</tbody>
</table>

## Related Data Structures

- FTFCA (page 298)
- FTF-SHUTDOWN-INFO (page 352)
- FTF-SHUTDOWN-REPLY (page 355)

## Program Flow

The program flow for the FTFEND API is:
Get input parameters

- **Get Input Parameters** - The routine retrieves the values that you established in your calling program.

Connect to the queue manager

- **Connect to the Queue Manager** - The utility routine establishes a connection to the MQSeries Queue Manager.

Set up parameters

- **Set Up Parameters** - The routine uses the values in your program to construct a command to issue to Tivoli Data Exchange.

Call FTFEND

- **Call FTFEND** - The program calls the FTFEND API for execution.

Reports results

- **Report Results** - The results of the FTFEND call are reported.

Disconnect from the queue manager

- **Disconnect from the Queue Manager** - The utility routine breaks the connection with the MQSeries Queue Manager.
FTFPING

Description

The FTFPING API allows you to ping specified Tivoli Data Exchange components. The ping message starts at the local queue manager, the machine from which you are currently working. It is sent to the Tivoli Data Exchange Manager (identified by the originating queue manager). The Tivoli Data Exchange Manager sends it to the Tivoli Data Exchange Sender (identified by the source queue manager). The Tivoli Data Exchange Sender sends it to the Tivoli Data Exchange Receiver (identified by the destination queue manager). The Tivoli Data Exchange Receiver then sends an acknowledgment to the Tivoli Data Exchange Manager.

The FTFPING API allows you to ensure that all Tivoli Data Exchange components involved in a file-transfer request are available before you start the data-transfer request. You can also use the FTFPING API to test various message size values until you find the optimal message size setting.

Prototype

```
WORKING-STORAGE.
COPY FTFICOPY.
PROCEDURE DIVISION.
CALL 'FTFPING' USING BY VALUE CONNECTION-HANDLE
BY REFERENCE FTFPING-INFO
BY REFERENCE FTFCA.
```

Related Sample Program

- FTFPING
**Parameters**

The following table lists and describes the FTFPING API’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION-HANDLE</td>
<td>CONNECTION-HANDLE</td>
<td>Represents the connection to the queue manager established in the MQCONN call.</td>
</tr>
<tr>
<td>FTFPING-INFO</td>
<td>Address of FTFPING-INFO in the WORKING-STORAGE SECTION.</td>
<td>Represents a properly formatted data structure used by the FTFPING call. This data structure contains all of the attributes and input criteria required to submit a ping message.</td>
</tr>
<tr>
<td>FTFCA</td>
<td>Address of FTFCA in the WORKING-STORAGE SECTION.</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
</tbody>
</table>

**Related Data Structures**

- FTFCA (page 298)
- FTFPING-INFO (page 312)

**Program Flow**

The program flow for the FTFPING API is:
- **Get Input Parameters** - The routine retrieves the values that you established in your calling program.

- **Connect to the Queue Manager** - The utility routine establishes a connection to the MQSeries Queue Manager.

- **Set Up Parameters** - The routine uses the values in your program to construct a command to issue to Tivoli Data Exchange.

- **Call FTFPING** - The program calls FTFPING API for execution.

- **Report Results** - The results of the FTFPING call are reported.

- **Disconnect from the Queue Manager** - The utility routine breaks the connection with the MQSeries Queue Manager.
**COBOL API Reference**

**FTFREQ**

**Description**

The FTFREQ function generates a Tivoli Data Exchange data-transfer request. The data-transfer request is initiated when the FTFREQ API is called and submitted to the Tivoli Data Exchange subsystem. 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 Tivoli Data Exchange subsystem before returning control to the application.

---

**Note:**

When a data-transfer request fails and the file mode for the Tivoli Data Exchange Receiver is set to “Create” or “Replace”, the Tivoli Data Exchange Receiver deletes the file. If the data-transfer request fails before the Tivoli Data Exchange Receiver starts processing to the target file, clean-up is not required. However, if a data-transfer request that is set to “Create” or “Replace” fails and the Tivoli Data Exchange Receiver has started writing data to the target file, the Tivoli Data Exchange Receiver deletes the file.

---

**Prototype**

```cobol
WORKING-STORAGE.
COPY FTFICOPY.
PROCEDURE DIVISION.
   CALL 'FTFREQ' USING BY VALUE CONNECTION-HANDLE
       BY REFERENCE
       FTF-REQUEST-MESSAGE-INFO
       BY REFERENCE FTFCA.
```

**Related Sample Program**

- FTFREQ
Parameters

The following table lists and describes the FTFREQ API’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION-HANDLE</td>
<td>CONNECTION-HANDLE</td>
<td>Represents the connection to the queue manager established in the MQCONN call.</td>
</tr>
<tr>
<td>FTF-REQUEST-MESSAGE-INFO</td>
<td>Address of FTFREQ-REQUEST-MESSAGE-INFO in the WORKING-STORAGE SECTION.</td>
<td>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 Tivoli Data Exchange data-transfer request, the details for delivery, and the criteria for the source and target file.</td>
</tr>
<tr>
<td>FTFCA</td>
<td>Address of FTFCA in the WORKING-STORAGE SECTION.</td>
<td>Represents an output area that returns properly formatted API reply information, including the return codes and optional messages.</td>
</tr>
</tbody>
</table>

Related Data Structures

- FTFCA (page 298)
- FTF-REQUEST-MESSAGE-INFO (page 315)

Program Flow

The program flow for the FTFREQ API is:
**Get Input Parameters** - The routine retrieves the values that you established in your calling program.

**Connect to the Queue Manager** - The utility routine establishes a connection to the MQSeries Queue Manager.

**Set Up Parameters** - The routine uses the values in your program to construct a command to issue to Tivoli Data Exchange.

**Call FTFREQ** - The program calls FTFREQ API for execution.

**Report Results** - The results of the FTFREQ call are reported.

**Disconnect from the Queue Manager** - The utility routine breaks the connection with the MQSeries Queue Manager.
**Description**

Each Tivoli Data Exchange component sends status updates during the processing cycle of a Tivoli Data Exchange data-transfer request. FTFSL retrieves the latest summary updates from each of the components during a file-transfer request. It accepts a filter argument and retrieves updates for each data-transfer request that matches the given filter. For example, it can be used to retrieve the status summary for all file-transfer requests that occurred on a given date or within a date range. It can also be used to retrieve the status summary for all data-transfer requests that have failed. FTFSL allows the calling program to track the status of current data-transfer requests and the status of completed data-transfer requests.

**Prototype**

```cobol
WORKING-STORAGE.
COPY FTFICOPY.
PROCEDURE DIVISION.
  CALL 'FTFSL' USING
  BY VALUE CONNECTION-HANDLE
  BY REFERENCE FTF-STATSUM-INFO
  **BY REFERENCE FTF-STATUS-SUMMARY-FILTER
  BY VALUE NULL-POINTER
  BY REFERENCE FTF-STATUS-SUMMARY-LIST
  BY REFERENCE FTFCA.
```

**Note:**

A NULL pointer is passed instead of a filter to obtain all status information.
**Parameters**

The following table lists and describes the FTFSL API’s parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTION-HANDLE</td>
<td>CONNECTION-HANDLE</td>
<td>Represents the connection to the queue manager established in the MQCONN call.</td>
</tr>
<tr>
<td>FTF-STATSUM-INFO</td>
<td>Data structure</td>
<td>Represents the data structure containing the information that determines what summary status information is returned by the API.</td>
</tr>
<tr>
<td>FTF-STATUS-SUMMARY-FILTER</td>
<td></td>
<td>Contains the filter information required for the API to retrieve the most recent status for all data-transfer requests that meet the filter requirements.</td>
</tr>
<tr>
<td>NULL-POINTER</td>
<td>A pointer set to NULLS.</td>
<td></td>
</tr>
<tr>
<td>FTF-STATUS-SUMMARY-LIST</td>
<td>The number of entries and a pointer to the first entry.</td>
<td>Represents the output data structure that contains the summary status information of the Tivoli Data Exchange components of all data-transfer requests that meet the filter criteria. It contains the most recent status updates for each Tivoli Data Exchange component associated with the data-transfer requests.</td>
</tr>
<tr>
<td>FTFCA</td>
<td>Address of FTFCA in the WORKING-STORAGE SECTION.</td>
<td>Represents an output area that returns properly formatted API reply information, including return codes and optional messages.</td>
</tr>
</tbody>
</table>

**Related Data Structures**

- FTFCA (page 298)
- FTF-STATUS-SUMMARY-LIST (page 351)
The general flow of all of the utility routine is:

- **Get Input Parameters** - The routine retrieves the values that you established in your calling program.
- **Connect to the Queue Manager** - The utility routine establishes a connection to the MQSeries Queue Manager.
- **Set Up Parameters** - The routine uses the values in your program to construct a command to issue to Tivoli Data Exchange.
- **Call FTFSL** - The program calls FTFSL API for execution.
- **Report Results** - The results of the FTFSL call are reported.
- **Disconnect from the Queue Manager** - The utility routine breaks the connection with the MQSeries Queue Manager.
Compiling the COBOL API

DLL support is provided only for the IBM COBOL for OS/390 and VM.

Tivoli Data Exchange distributes its API library in two formats:

- A dynamic load library (DLL)
- A data set containing the statically linkable object deck

Depending on the COBOL product version you are using to write your applications, integration choices and procedures differ. In general, if the language and the product version support the DLL facility, it is recommended that you use the DLL support provided with Tivoli Data Exchange.

DLL Support

Tivoli Data Exchange ships the FTFBLIB DLL for integrating enterprise applications. This DLL is available in the APILIB data set. A SIDEDECK, which contains references to all the exported symbols of this DLL, is available in the FTFBLIB member of the SIDEDECK data set.

The FTFBLIB DLL has been linked with the MQSeries Batch interface stub calls. As a result, this DLL can be used for batch applications only. If you require support for other MQSeries environments, contact your Tivoli Data Exchange representative.
Dynamically Linking COBOL Applications

The following JCL sample is provided as the basis for dynamically linking and compiling COBOL programs. It is also located in the Tivoli Data Exchange
Compiling the COBOL API

JCLLIB data set in the FDLLLINK member.
//FDLLLINK JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID

/******************************************************************************
  ** Licensed Materials - Property of Tivoli Systems                        *
  ** Tivoli (R) Data Exchange                                              *
  ** Version 1.2.0                                                         *
  ** Product Number 5698-TDE                                                *
  ** (C) Copyright Tivoli Systems 2000.                                   *
  ** (C) Copyright CommerceQuest 2000.                                    *
  ** All rights reserved.                                                 *
  **                                                                  *
  ** US Government Users Restricted Rights - Use,                         *
  ** duplication or disclosure restricted by GSA ADP                      *
  ** Schedule Contract with IBM Corp.                                     *
  **                                                                  *
  ** Tivoli is a trademark of Tivoli Systems Inc.                         *
  ******************************************************************************

Tivoli Data Exchange sample Link-Edit using FTFBLIB DLL
This sample JCL provides an example of how an application that uses the DLL support provided by Tivoli Data Exchange should be linked.

REPLACE THE JOB CARD ABOVE WITH A VALID JOB CARD

REQUIRED SYMBOLIC CHANGES:

++FTFHLQ++ HIGH LEVEL QUALIFIER OF THE Tivoli Data Exchange DATASETS (E.G. TDE)
++MQMHLQ++ HIGH LEVEL QUALIFIER OF THE MQSERIES DATASETS (E.G. SYS1.V114)
++MEMBER++ MEMBER NAME OF THE SAMPLE THAT NEEDS TO BE COMPILED (E.G. COBREQ)
++OBJLIB++ OBJECT DATASET NAME WHERE COMPILER OUTPUT OF SOURCE MEMBER IS KEPT
++LOADLIB++ OUTPUT DATASET NAME FOR THE COMPILED/LINK-EDITED OUTPUT
++SIDEED++ OPTIONAL SIDE DECK SPECIFICATION IF YOU ARE COMPILING A DLL YOURSELF, FOR CREATING A Tivoli Data Exchange EXIT DLL

 reel text block
This JCL assumes that the source module has already been precompiled. As input to the prelinker, the compiled object module and the SIDEDECK definition for Tivoli Data Exchange DLL FTFBLIB is provided. Providing the side deck definition to the prelinker indicates that the symbol information should be resolved at runtime and that the DLL should be implicitly loaded.

The data set that contains FTFBLIB DLL must be made available during execution of the application, using any one of the standard means, such as STEPLIB concatenation, LINKLISTs, or VLF. A program abend occurs if the FTFBLIB DLL is not available for fetching during the execution of the application.

Tivoli Data Exchange provides short-name mappings for all available API calls for OS/390 and OS/400 support. Name mappings, which are less than 8 characters, are provided for each supported API. These mappings are listed in the “C API Reference” chapter in this manual under the “OS/390 Prototype” heading.
This section lists the data structures that are used with the Tivoli Data Exchange COBOL API functions to handle input and output data.

The copybooks for all of the examples included within this document are included in the COBSAMP data set in the member named FTFICOPY.

This chapter includes the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFCA</td>
<td>286</td>
</tr>
<tr>
<td>FTF-CANCEL-INFO</td>
<td>288</td>
</tr>
<tr>
<td>FTF-DELETE-INFO</td>
<td>290</td>
</tr>
<tr>
<td>FTF-DETLIST-INFO</td>
<td>292</td>
</tr>
<tr>
<td>FTF-EXIT-INFO</td>
<td>294</td>
</tr>
<tr>
<td>FTFPING-INFO</td>
<td>300</td>
</tr>
<tr>
<td>FTF-REQUEST-MESSAGE-INFO</td>
<td>303</td>
</tr>
<tr>
<td>FTF-STATSUM-INFO</td>
<td>323</td>
</tr>
<tr>
<td>FTF-STATUS-DETAIL</td>
<td>325</td>
</tr>
<tr>
<td>FTF-STATUS-DETAIL-LIST</td>
<td>328</td>
</tr>
<tr>
<td>FTF-STATUS-SUMMARY</td>
<td>329</td>
</tr>
<tr>
<td>FTF-STATUS-SUMMARY-FILTER</td>
<td>333</td>
</tr>
<tr>
<td>FTF-STATUS-SUMMARY-LIST</td>
<td>338</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO</td>
<td>339</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-REPLY</td>
<td>342</td>
</tr>
</tbody>
</table>
FTFCA

Description

This data structure receives output from all modules. It contains 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFC-RETURN-CODE1</td>
<td>S9(9) BINARY</td>
<td>Contains the primary return code or FTFRC. These codes are defined in the copybook file and represent a Tivoli Data Exchange internal code with an associated message.</td>
</tr>
<tr>
<td>FTFC-RETURN-CODE2</td>
<td>S9(9) BINARY</td>
<td>Contains the secondary return code. This data element contains external return or reason codes. For example, if Tivoli Data Exchange encounters an MQSeries error, the primary return code is an FTFRC and the secondary is an MQRC. If the error is a Tivoli Data Exchange internal error the secondary code can be either 0 or an additional FTFRC.</td>
</tr>
<tr>
<td>FTFC-ERROR-MESSAGE</td>
<td>X(1024)</td>
<td>Contains message text and is optional in many cases.</td>
</tr>
</tbody>
</table>
Data Structure

*********************************************************************
* FTF RETURN COMMON AREA FOR RC1, RC2 AND ERROR MESSAGE          *
*********************************************************************
*                                                                   *
  01  FTFCA  SYNC  RIGHT.
    05  FTFC-RETURN-CODE1  PIC  S9(9)  BINARY.
    05  FTFC-RETURN-CODE2  PIC  S9(9)  BINARY.
    05  FTFC-ERROR-MESSAGE  PIC  X(1024).  

Related Functions

- FTFCAN (page 277)
- FTFEND (page 280)
- FTPPING (page 283)
- FTFRQ (page 286)
- FTFSL (page 289)
**FTF-CANCEL-INFO**

**Description**

This data structure contains input information required by the FTFCAN API.

**Data Elements**

The following table lists and describes the FTF-CANCEL-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFCI-LQM</td>
<td>X(48)</td>
<td>Contains a properly formatted queue manager name to which the FTFCAN API has a connection. If this element is not provided, the lqm takes the value of the default queue manager. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFCI-OQM</td>
<td>X(48)</td>
<td>Contains the name of the queue manager used by the FTFCAN API. This value represents the queue manager for which the originating Tivoli Data Exchange Manager has serviced the initial request.</td>
</tr>
<tr>
<td>FTFCI-FTF-ID</td>
<td>X(37)</td>
<td>Contains the unique FTFID that correlates to the Tivoli Data Exchange data-transfer request being canceled.</td>
</tr>
<tr>
<td>FTFCI-CONFIG-FILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFCI-TIMEOUT</td>
<td>S9(9)</td>
<td>This is the timeout value that is used by the cancel request.</td>
</tr>
<tr>
<td></td>
<td>BINARY</td>
<td></td>
</tr>
<tr>
<td>FTFCI-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
</tbody>
</table>
Data Structure

<table>
<thead>
<tr>
<th>01 FTF-CANCEL-INFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 FTFCI-LQM PIC X(48) VALUE SPACES.</td>
</tr>
<tr>
<td>05 FILLER PIC X VALUE X'00'.</td>
</tr>
<tr>
<td>05 FTFCI-OQM PIC X(48) VALUE SPACES.</td>
</tr>
<tr>
<td>05 FILLER PIC X VALUE X'00'.</td>
</tr>
<tr>
<td>05 FTFCI-FTF-ID PIC X(37) VALUE SPACES.</td>
</tr>
<tr>
<td>05 FTFCI-CONFIG-FILE PIC X(256) VALUE SPACES.</td>
</tr>
<tr>
<td>05 FILLER PIC X VALUE X'00'.</td>
</tr>
<tr>
<td>05 FTFCI-TIMEOUT PIC S9(9) BINARY VALUE 0.</td>
</tr>
<tr>
<td>05 FTFCI-CQ PIC X(48) VALUE SPACES.</td>
</tr>
<tr>
<td>05 FILLER PIC X VALUE X'00'.</td>
</tr>
</tbody>
</table>

Related Functions

- FTCICAN (page 277)
**COBOL Data Structures**

**FTF-DELETE-INFO**

### Description

This data structure contains the information that determines which status records are deleted by the FTFSD API.

### Data Elements

The following table lists and describes the FTF-DELETE-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFDELETE-LQM</td>
<td>X(48)</td>
<td>Contains the name of the lqm from which the delete operation occurs and for which status records are deleted. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFDELETE-CONFIG-FILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFDELETE-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFDETL-FTF-ID</td>
<td>X(37)</td>
<td>Contains the FTFID for which status information is deleted.</td>
</tr>
</tbody>
</table>
**Data Structure**

```cobol
* FTFSTATUSDELETE FTF-DELETE-INFO STRUCTURE
* **************************************************************
01 FTF-DELETE-INFO.
  05 FTFDELETE-LQM PIC X(48) VALUE SPACES.
  05 FILLER PIC X VALUE X'00'.
  05 FTFDELETE-CONFIG-FILE PIC X(256) VALUE SPACES.
  05 FILLER PIC X VALUE X'00'.
  05 FTFDELETE-CQ PIC X(48) VALUE SPACES.
  05 FILLER PIC X VALUE X'00'.
  05 FTFDETL-FTF-ID PIC X(37) VALUE SPACES.
* **************************************************************
* END OF FTF EXIT INFO
* **************************************************************
```

**Related Functions**

None
COBOL Data Structures

FTF-DETLLIST-INFO

Description

This data structure is for future use and contains the information determining what detailed status information will be returned by the FTFDL API.

Data Elements

The following table lists and describes the FTF-DETLLIST-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFDETL-LQM</td>
<td>X(48)</td>
<td>Contains the name of the lqm to which the FTFDL API has a connection. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFDETL-CONFIG-FILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFDETL-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFDETL-FTF-ID</td>
<td>X(37)</td>
<td>Contains the FTIFD for which detail status information is retrieved.</td>
</tr>
</tbody>
</table>
Data Structure

```
01 FTF-DETLLIST-INFO.
   05 FTFDETL-LQM PIC X(48) VALUE SPACES.
   05 FILLER PIC X VALUE X'00'.
   05 FTFDETL-CONFIG-FILE PIC X(256) VALUE SPACES.
   05 FILLER PIC X VALUE X'00'.
   05 FTFDETL-CQ PIC X(48) VALUE SPACES.
   05 FILLER PIC X VALUE X'00'.
   05 FTFDETL-FTF-ID PIC X(37) VALUE SPACES.
```

Related Functions

For future use
**FTF-EXIT-INFO**

**Description**

This data structure contains the exit information required to use the pre- and post-processing exits supplied with Tivoli Data Exchange. This data structure represents a list of exits, each of which applies to the specific data-transfer request. It is passed to the exit being called for the user-written program to reference.

**Notes:**

- For more information about custom connector development, contact CommerceQuest (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 FTF-EXIT-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFE-FTFID</td>
<td>X(48)</td>
<td>The identifier associated with the data-transfer request.</td>
</tr>
<tr>
<td>FTFE-EXIT-NUMBER</td>
<td>S9(9) BINARY</td>
<td>Represents the number of the exit to be called. The available exits and their numbers are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Compression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supplied or user-written exit for data compression. This exit is called just before outbound messages are submitted at the source or sending node.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Decompression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supplied or user-written exit for data decompression. This exit is called when inbound messages at the destination node are retrieved.</td>
</tr>
</tbody>
</table>
3. Manager pre-process exit
User-written exit for preprocessing requirements at the Tivoli Data Exchange Manager.

4. Manager post-process exit
User-written exit for preprocessing requirements at the Tivoli Data Exchange Manager. This exit is useful for processing acknowledgments to user programs. Each of the exits receives the current FTFRC as input. Therefore, the Tivoli Data Exchange Manager post-process exit can be used to audit all Tivoli Data Exchange processed data-transfer requests in order to detect failure or success.

5. Sender pre-process exit
User-written exit for preprocessing requirements at the Tivoli Data Exchange Sender. This is specifically useful for performing various tasks on the data before the file is sent.

6. Sender post-process exit
User-written exit for postprocessing requirements at the Tivoli Data Exchange Sender. This exit is useful for performing various tasks just after all data is submitted to the destination node. These tasks include audit of success and failure of a file transfer at the sending node.

7. Receiver pre-process exit
User-written exit for preprocessing requirements at the Tivoli Data Exchange Receiver. This exit is useful for performing various tasks just before the file is deposited at the target location.

8. Receiver post-process exit
User-written exit for postprocessing requirements at the Tivoli Data Exchange Receiver. This exit is useful for performing various tasks just after the file is deposited at its destination. Because the Tivoli Data Exchange Receiver performs the last step in the data-transfer request before the Tivoli Data Exchange Manager logs the final status, this exit is specifically useful for auditing failed or successful file transfer requests.
## COBOL Data Structures

### FTF-EXIT-INFO

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9. Sender connector</td>
<td></td>
<td>Connector development requires that you contact CommerceQuest for any necessary custom development.</td>
</tr>
<tr>
<td>10. Receiver connector</td>
<td></td>
<td>Connector development requires that you contact CommerceQuest for any necessary custom development.</td>
</tr>
<tr>
<td>FTFE-DLL-POINTER</td>
<td>POINTER</td>
<td>Represents the name of the run-time module to be called for the desired exit. This data element is case sensitive and must be in the run-time path of the environment calling the exit (unless the dllName contains the fully qualified path of the run-time module).</td>
</tr>
<tr>
<td>FTFE-ENTRYPOINT-POINTER</td>
<td>POINTER</td>
<td>Represents the entry point within the dllName (or run-time module) to be executed. Typically, multiple exits can use the same run-time module but have different entry points.</td>
</tr>
<tr>
<td>FTFE-USER-ID-POINTER</td>
<td>POINTER</td>
<td>Represents a user ID. Tivoli Data Exchange performs no action based on the value of this field. It is passed to the called exit and can be used for any purpose by the exit module.</td>
</tr>
<tr>
<td>FTFE-PASSWORD-POINTER</td>
<td>POINTER</td>
<td>Represents a user ID’s password. Tivoli Data Exchange performs no action based on the value of this field. It is passed to the called exit and can be used for any purpose by the exit module.</td>
</tr>
<tr>
<td>FTFE-C-USER-DATA</td>
<td>S9(9) BINARY</td>
<td>Represents the number of bytes within the user data field.</td>
</tr>
<tr>
<td>FTFE-CUSER-DATA-POINTER</td>
<td>POINTER</td>
<td>Represents 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.</td>
</tr>
<tr>
<td>FTFE-RETURN-CODE</td>
<td>S9(9) BINARY</td>
<td>Represents the primary 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.</td>
</tr>
<tr>
<td>FTFE-RETURN-CODE2</td>
<td>S9(9) BINARY</td>
<td>Represents the 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.</td>
</tr>
</tbody>
</table>
### COBOL Data Structures

**FTF-EXIT-INFO**

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-INTERNAL-POINTER</td>
<td>POINTER</td>
<td>Reserved. Do not use this data element.</td>
</tr>
<tr>
<td>FTF-USER-FTFE-IDENTIFIERS</td>
<td></td>
<td>Designates the identifier fields.</td>
</tr>
<tr>
<td>FTFE-ID1</td>
<td>X(50)</td>
<td>Contains the first text field that is associated with the message.</td>
</tr>
<tr>
<td>FTFE-ID2</td>
<td>X(50)</td>
<td>Contains the second text field that is associated with the message.</td>
</tr>
<tr>
<td>FTFE-ID3</td>
<td>X(50)</td>
<td>Contains the third text field that is associated with the message.</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

*FTF-EXIT-INFO*

**Data Structure**

```
*** FTF EXIT INFO  
***

01 FTF-EXIT-INFO.
  05 FTFE-FTFID PIC X(48) VALUE SPACES.
  05 FTFE-EXIT-NUMBER PIC S9(9) BINARY.
  05 FTFE-DLL-POINTER USAGE IS POINTER.
  05 FTFE-ENTRYPOINT-POINTER USAGE IS POINTER.
  05 FTFE-USER-ID-POINTER USAGE IS POINTER.
  05 FTFE-PASSWORD-POINTER USAGE IS POINTER.
  05 FTFE-CUSER-DATA PIC S9(9) BINARY.
  05 FTFE-CUSER-DATA-POINTER USAGE IS POINTER.
  05 FTFE-RETURN-CODE PIC S9(9) BINARY.
  05 FTFE-RETURN-CODE2 PIC S9(9) BINARY.
  05 FTFE-INTERNAL-POINTER USAGE IS POINTER.
  05 FTF-USER-FTFE-IDENTIFIERS.
    10 FTFE-ID1 PIC X(50) VALUE LOW-VALUES.
    10 FTFE-ID2 PIC X(50) VALUE LOW-VALUES.
    10 FTFE-ID3 PIC X(50) VALUE LOW-VALUES.

*---------------------------------------------------------------------

* FTF REQUEST FIELDS POINTED TO BY FTF EXIT INFO  
*---------------------------------------------------------------------

01 FTFE-DLL-NAME PIC X(256).
01 FTFE-ENTRYPOINT PIC X(48).
01 FTFE-USER-ID PIC X(48).
01 FTFE-PASSWORD PIC X(20).

*** END OF FTF EXIT INFO  
***
```

**Related Functions**

None.
### FTF Request Fields

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFE-DLL-NAME</td>
<td>X(256)</td>
<td>Full name of user DLL</td>
</tr>
<tr>
<td>FTFE-ENTRYPOINT</td>
<td>X(48)</td>
<td>Entry point of the DLL</td>
</tr>
<tr>
<td>FTFE-USER-ID</td>
<td>X(48)</td>
<td>User ID to access the module</td>
</tr>
<tr>
<td>FTFE-PASSWORD</td>
<td>X(20)</td>
<td>Password to access the module</td>
</tr>
</tbody>
</table>

### Data Structure

```cobol
* FTF REQUEST FIELDS POINTED TO BY FTF EXIT INFO
*---------------------------------------------------------------------*
01 FTFE-DLL-NAME PIC X(256).
01 FTFE-ENTRYPOINT PIC X(48).
01 FTFE-USER-ID PIC X(48).
01 FTFE-PASSWORD PIC X(20).
```
**FTFPING-INFO**

**Description**

This data structure represents input information required by the FTFPING API. Its data elements describe all of the components of the ping operation.

**Data Elements**

The following table lists and describes the FTF-PING-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFP-LQM</td>
<td>X(48)</td>
<td>Contains the name of the local queue manager to which the FTFPING API has a connection. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFP-OQM</td>
<td>X(48)</td>
<td>Represents a properly formatted originating queue manager name used by the FTFPING API. The FTFPING message is forwarded to this queue manager. Except on OS/390, this parameter is optional. If it is not provided, the local queue manager value is used as the originating queue manager. In other words, the Tivoli Data Exchange Manager is on the same queue manager to which the FTFPING API has connected.</td>
</tr>
<tr>
<td>FTFP-SQM</td>
<td>X(48)</td>
<td>Contains the name of the queue manager to which the Tivoli Data Exchange Manager forwards the ping message. This parameter is optional. If it is not provided, Tivoli Data Exchange uses the local queue manager value. In other words, the Tivoli Data Exchange Sender is on the same queue manager to which the FTFPING API has connected.</td>
</tr>
</tbody>
</table>
### COBOL Data Structures

**FTFPING-INFO**

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFP-DQM</td>
<td>X(48)</td>
<td>Contains the name of the queue manager to which the Tivoli Data Exchange Sender forwards the ping message. This parameter is optional. If it is not provided, Tivoli Data Exchange used the local queue manager value. In other words, the Tivoli Data Exchange Receiver is on the same queue manager to which the FTFPING API has connected.</td>
</tr>
<tr>
<td>FTFP-CFILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFP-TIMEOUT</td>
<td>S9(9) BINARY</td>
<td>Represents how long FTFPING should wait for the final response from the Tivoli Data Exchange Manager. This value is specified in seconds. <strong>Valid values:</strong> A value between 0 and ULONGMAX.</td>
</tr>
<tr>
<td>FTFP-MESSAGESIZE</td>
<td>S9(9) BINARY</td>
<td>Represents the ping message size. This value is specified in KB. <strong>Valid values:</strong> A value between 1 and the maximum message size of MQSeries in KB. This value is MQSeries configuration specific. This value can not exceed the “Maximum Message Length” attribute of the Queue Manager.</td>
</tr>
<tr>
<td>FTFP-PRIORITY</td>
<td>S9(9) BINARY</td>
<td>Represents the priority of the ping message. <strong>Valid values:</strong> 1 (highest) - 5 (lowest).</td>
</tr>
<tr>
<td>FTFP-KBYTESWRITTEN</td>
<td>S9(9) BINARY</td>
<td>Represents the actual number of bytes written to MQSeries for the ping message. <strong>Valid values:</strong> A value between 1 and the maximum message size supported by MQSeries in bytes.</td>
</tr>
<tr>
<td>FTFP-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

**FTFPING-INFO**

**Data Structure**

```
01 FTFPING-INFO SYNC RIGHT.
   05 FTFP-LQM PIC X(48).
   05 FILLER PIC X VALUE X'00'.
   05 FTFP-OQM PIC X(48).
   05 FILLER PIC X VALUE X'00'.
   05 FTFP-SQM PIC X(48).
   05 FILLER PIC X VALUE X'00'.
   05 FTFP-DQM PIC X(48).
   05 FILLER PIC X VALUE X'00'.
   05 FTFP-CFILE PIC X(256).
   05 FILLER PIC X VALUE X'00'.
   05 FTFP-TIMEOUT PIC S9(9) BINARY VALUE X'00'.
   05 FTFP-MESSAGESIZE PIC S9(9) BINARY VALUE 0.
   05 FTFP-PRIORITY PIC S9(9) BINARY VALUE 5.
   05 FTFP-KBYTESWRITTEN PIC S9(9) BINARY VALUE 0.
   05 FTFP-CQ PIC X(48).
   05 FILLER PIC X VALUE X'00'.
```

**Related Functions**

- FTFPING (page 283)
FTF-REQUEST-MESSAGE-INFO

The FTF-REQUEST-MESSAGE-INFO data structure is quite large and it has been presented here in parts for simplicity of understanding. The parts are as follows:

<table>
<thead>
<tr>
<th>Parts of FTF-REQUEST-MESSAGE-INFO</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-QMGRSINFO</td>
<td>303</td>
</tr>
<tr>
<td>FTF-STATUS-DETAIL</td>
<td>325</td>
</tr>
<tr>
<td>FTF-TARGET-FILE-INFO</td>
<td>308</td>
</tr>
<tr>
<td>FTF-JOB-INFO</td>
<td>313</td>
</tr>
<tr>
<td>FTF-USER-INFO</td>
<td>317</td>
</tr>
<tr>
<td>FTFAS400-FILE-INFO</td>
<td>320</td>
</tr>
</tbody>
</table>

Related Functions

The FTF-REQUEST-MESSAGE-INFO data structure and all its components are used in the following function:

- FTFREQ (page 286)

FTF-QMGRSINFO

Description

This data structure determines the queue managers used in a file-transfer request.
**Data Elements**

The following table lists and describes the FTF-QMGRSINFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFQ-LQM</td>
<td>X(48)</td>
<td>Contains the name of the local queue manager to which the FTFREQ API has a connection. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFQ-OQM</td>
<td>X(97)</td>
<td>Represents the originating queue manager on which the Tivoli Data Exchange data-transfer request originates. If this value is omitted, the Tivoli Data Exchange subsystem uses the local queue manager value for the originating queue manager.</td>
</tr>
<tr>
<td>FTFQ-SQM</td>
<td>X(97)</td>
<td>Represents the queue manager from which the source file is sent. The Tivoli Data Exchange Sender must be running on this node.</td>
</tr>
<tr>
<td>FTFQ-DQM</td>
<td>X(97)</td>
<td>Represents the queue manager where the destination file is received. The Tivoli Data Exchange Receiver must be running on this node.</td>
</tr>
</tbody>
</table>
FTF-SOURCE-FILE-INFO

Description

This data structure lists all the information required for describing the source file, its location, attributes, and delivery rules.

Data Elements

The following table lists and describes the FTF-SOURCE-FILE-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFS-SPATH-POINTER</td>
<td>POINTER</td>
<td>Represents a pointer to a variable (located in FTF-REQUEST-FIELDS) that contains the source filename and its fully qualified path. This path must be syntactically correct for the source platform that is accessing the file.</td>
</tr>
<tr>
<td>FTFS-IS-STAGED</td>
<td>9(4) BINARY</td>
<td>Represents the rules governing the staging queue. Upon request, the source file, once read, can be archived in a staging queue until it has been transmitted to each of its destinations. <strong>Valid values:</strong> 1 (staging is required), 0 (staging is not required).</td>
</tr>
</tbody>
</table>
### FTF-REQUEST-MESSAGE-INFO

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-REQUEST-MESSAGE-INFO</td>
<td></td>
<td>Represents the type of data transfer requested. For text files, no data conversion takes place when moving across dissimilar platforms. For binary files, data is translated on the receiving platform according to the local code page. <strong>Valid values:</strong> FTF-FILE-TYPE-BINARY X’01’. FTF-FILE-TYPE-TEXT X’02’.</td>
</tr>
<tr>
<td>FTF-IS-STAGEPERS</td>
<td>9(4) BINARY</td>
<td>If staging is enabled for a given request, you can choose to specify whether the data in the stage area is persistent or nonpersistent. Nonpersistent data does not survive a system restart. Therefore, if the data transfer request requires recovery and the staged data is not present, it will be required to re-access the original file from disk (if it has not changed). <strong>Valid values:</strong> 1 (stage is persistent), 0 (stage is nonpersistent).</td>
</tr>
<tr>
<td>FTF-IS-DATA-PERS</td>
<td>9(4) BINARY</td>
<td>Represents the rules governing the persistence of the message components of the file that is being transferred. This is a performance option which defaults to non-persistent. <strong>Valid values:</strong> 1 (persistent), 0 (nonpersistent).</td>
</tr>
<tr>
<td>FTF-IS-COMpressed</td>
<td>9(4) BINARY</td>
<td>Represents the rules governing the compression options. The data can be compressed or not compressed. If data is compressed, the data conversion exits must be installed at the source and target nodes for compression and decompression to occur properly. <strong>Valid values:</strong> 1 (compressed), 0 (uncompressed).</td>
</tr>
<tr>
<td>FTF-IS-DELETE</td>
<td>9(4) BINARY</td>
<td>Used internally by Tivoli Data Exchange.</td>
</tr>
<tr>
<td>FTF-FILE-TYPE-INFO</td>
<td>X</td>
<td>Represents the type of data transfer requested. For text files, no data conversion takes place when moving across dissimilar platforms. For binary files, data is translated on the receiving platform according to the local code page. <strong>Valid values:</strong> FTF-FILE-TYPE-BINARY X’01’. FTF-FILE-TYPE-TEXT X’02’.</td>
</tr>
</tbody>
</table>
| FTF-RECORD-PADDING  | X             | Enumerated data type that determines whether padding takes place. **Valid values:** FTF-PAD X’01’.
FTF-NO-PAD X’02’. |
| FTF-PAD-CHARACTER   | X             | Contains the character used for padding.                                                                                                                                                                |
FTF-REQUEST-MESSAGE-INFO

FTFS-SType X (26) 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 Tivoli Data Exchange configuration file.

FTFS-STYPE S9(9) BINARY Determines the length in bytes of the *pstypeData data element.

FTFS-SType-INFO-POINTER POINTER Pointer to the source data that is not contained in a file.

FTFS-BUFFNO S9(9) BINARY Determines the number of buffers to be allocated for a data transfer on the OS/390 platform.
**COBOL Data Structures**

**FTF-REQUEST-MESSAGE-INFO**

### Data Structure

<table>
<thead>
<tr>
<th>* FTF REQUEST SOURCE FILE INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>* FTF SOURCE-FILE-INFO.</td>
</tr>
<tr>
<td>05 FTF-SOURCE-FILE-INFO.</td>
</tr>
<tr>
<td>10 FTF-SPATH-POINTER USAGE IS POINTER.</td>
</tr>
<tr>
<td>10 FTF-IS-STAGED PIC 9(4) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-IS-STAGEPERS PIC 9(4) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-IS-DATA-PERS PIC 9(4) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-IS-COMPRessed PIC 9(4) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-IS-DELETE PIC 9(4) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-IS-FILE-TYPE-INFO PIC X VALUE X’01’.</td>
</tr>
<tr>
<td>88 FTF-FILE-TYPE-BINARY VALUE X’01’.</td>
</tr>
<tr>
<td>88 FTF-FILE-TYPE-TEXT VALUE X’02’.</td>
</tr>
<tr>
<td>10 FTF-RECORD-PADDING PIC X VALUE X’01’.</td>
</tr>
<tr>
<td>88 FTF-PAD VALUE X’01’.</td>
</tr>
<tr>
<td>88 FTF-NO-PAD VALUE X’02’.</td>
</tr>
<tr>
<td>10 FTF-PAD-CHARACTER PIC X VALUE SPACES.</td>
</tr>
<tr>
<td>10 FTF-SType PIC X(26) VALUE SPACES.</td>
</tr>
<tr>
<td>10 FILLER PIC X VALUE SPACES.</td>
</tr>
<tr>
<td>10 FTF-C-SType PIC S9(9) BINARY VALUE 0.</td>
</tr>
<tr>
<td>10 FTF-SType-INFO-POINTER USAGE IS POINTER.</td>
</tr>
<tr>
<td>10 FTF-BUFFNO PIC S9(9) BINARY VALUE 0.</td>
</tr>
</tbody>
</table>

---

**FTF-TARGET-FILE-INFO**

### Description

This data structure passes information about the current data transfer’s target file to the exit module.
### Data Elements

The following table lists and describes the FTF-TARGET-FILE-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-DPATH-POINTER</td>
<td>POINTER</td>
<td>Represents a pointer to a variable (located in FTF-REQUEST-FIELDS) that contains the filename and its fully qualified path. This path must be syntactically correct for the target platform that will be depositing the file.</td>
</tr>
<tr>
<td>FTF-IS-COMPRESSED</td>
<td>9(4) BINARY</td>
<td>Represents a flag that describes the inbound data as compressed or uncompressed. <strong>Valid values:</strong> 1 (compress), 0 (uncompress).</td>
</tr>
</tbody>
</table>
| FTF-FILE-TYPE-INFO    | X             | Represents the binary or text format of the data as it is transmitted. If binary is the chosen option, no conversion or formatting takes place. If text is the chosen option, code page translation will occur. This includes ASCII to EBCDIC translations as well as language translations. **Valid values:**
  - FTF-FILE-TYPE-BINARY X’01’.
  - FTF-FILE-TYPE-TEXT X’02’.
| FTF-FILE-MODE-INFO    | X             | Represents how the data is processed at the destination when data is deposited at the target. **Valid values:**
  - FTF-FILE-MODE-CREATE X’01’.
  - FTF-FILE-MODE-APPEND X’02’.
  - FTF-FILE-MODE-NOREPLACE X’03’.
| FTF-FILE-ORG-INFO     | X             | Specifies the file organization on OS/390. **Valid values:** PS (Physical Sequential), PDS (Partitioned Data Set)
  - FTF-FILE-ORG-PARTITIONED X’01’.
  - FTF-FILE-ORG-SEQUENTIAL X’02’.
<p>| Note: See the following paragraph, <em>OS/390 File Allocation Items.</em> |
| FTF-DIRECTORY-BLKS    | S9(9) BINARY  | Represents the number of directory blocks to be allocated on OS/390. <strong>Note:</strong> See the following paragraph, <em>OS/390 File Allocation Items.</em>                             |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFT-RECORD-FORMAT</td>
<td>X</td>
<td>Specifies the record format on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTF-RECFM-FIXED X‘01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTF-RECFM-VARIABLE X‘02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTF-RECFM-FIXED-BLOCKED X‘03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTF-RECFM-VARIABLE-BLOCKED X‘04’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-RECORD-LENGTH</td>
<td>S9(9) BINARY</td>
<td>Specifies the logical record length on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong> 1 - 32760.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-BLOCK-SIZE</td>
<td>S9(9) BINARY</td>
<td>Specifies the block size on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For fixed block (FB), the block size value must be a multiple of the logical record length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For variable block (VB), the blksize value must be at least four bytes greater than the logical record length.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong> 0-32760</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-UNIT-NAME</td>
<td>X(8)</td>
<td>Specifies the unit name on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-VOL-SER</td>
<td>X(6)</td>
<td>Specifies the volume serial number on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-ALLOCATION</td>
<td>X</td>
<td>Specifies the allocation unit to use on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Valid values:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTFT-ALLOCATION-UNIT-CYL X‘01’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTFT-ALLOCATION-UNIT-BLK X‘02’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTFT-ALLOCATION-UNIT-TRK X‘03’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
<tr>
<td>FTFT-PRIMARY-ALLOC</td>
<td>S9(9) BINARY</td>
<td>Specifies the primary allocation unit on OS/390.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> See the following paragraph, OS/390 File Allocation Items.</td>
</tr>
</tbody>
</table>
## COBOL Data Structures

### FTF-REQUEST-MESSAGE-INFO

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFT-SECOND-ALLOC</td>
<td>S9(9) BINARY</td>
<td>Specifies the secondary allocation unit on OS/390. Note: See the following paragraph, <em>OS/390 File Allocation Items</em>.</td>
</tr>
</tbody>
</table>
| FTFT-TEXT-WRAP-INFO | X             | Determines the treatment of the source data record if it is longer than the target data set. Wrap, Truncate, and Fail are supported. Applies only to text-mode transfers. **Valid values:**
|                     |               | FTFT-TEXT-WRAP X’01’.
|                     |               | FTFT-FAIL X’02’.
|                     |               | FTFT-TRUNC X’03’.
|                     |               | Note: See the following paragraph, *OS/390 File Allocation Items*. |
| FTFT-CREATE-DIR     | 9(4) BINARY   | Represents the rules governing the dynamic creation of the directory in the fully qualified path of the target file. **Valid values:** 1 (create directory), 0 (do not create a directory). |
| FTFT-ISDATA-PERSIST | 9(4) BINARY   | Represents the rules governing data persistence. The file data that is sent as MQSeries messages can be persistent or nonpersistent. Nonpersistent data does not survive a system restart. **Valid values:** 1 (persistent), 0 (not persistent). |
| FTFT-MODEL-DATASET  | X(44)         | Used to define a data set for GDG allocation. This data set contains definitions of the attributes group such as record length, block size, the maximum number of revisions that will be supported, and many others. These definitions form the basis for creating a GDG during processing **Valid values:** up to 44 characters |
| FTFT-DTYPE          | X(26)         | 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 Tivoli Data Exchange configuration file. |
| FTFT-C-DTYPE        | S9(9) BINARY  | Determines the length in bytes of the *pdtypeData data element. |
### COBOL Data Structures

#### FTF-REQUEST-MESSAGE-INFO

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-TARGET-FILE-INFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTFT-DPATH-POINTER</td>
<td>POINTER</td>
<td>Contains the receiver data that will not be stored in a file.</td>
</tr>
<tr>
<td>FTFT-BUFFNO</td>
<td>S9(9) BINARY</td>
<td>Determines the number of buffers to be allocated for a data transfer on the OS/390 platform.</td>
</tr>
</tbody>
</table>

---

#### Data Structure

```cobol
05 FTF-TARGET-FILE-INFO.
  10 FTFT-DPATH-POINTER USAGE IS POINTER.
  10 FTFT-IS-COMRESSED PIC 9(4) BINARY VALUE 0.
  10 FTFT-FILE-TYPE-INFO PIC X VALUE '01'.
     88 FTFT-FILE-TYPE-BINARY VALUE '01'.
     88 FTFT-FILE-TYPE-TEXT VALUE '02'.
  10 FTFT-FILE-MODE-INFO PIC X VALUE '01'.
     88 FTFT-FILE-MODE-CREATE VALUE '01'.
     88 FTFT-FILE-MODE-APPEND VALUE '02'.
     88 FTFT-FILE-MODE-NOREPLACE VALUE '03'.
  10 FTFT-FILE-ORG-INFO PIC X VALUE '01'.
     88 FTFT-FILE-ORG-PARTITIONED VALUE '01'.
     88 FTFT-FILE-ORG-SEQUENTIAL VALUE '02'.
  10 FILLER PIC X(3) VALUE SPACES.
  10 FTFT-DIRECTORY-BLKS PIC S9(9) BINARY.
```

---

312  Tivoli Data Exchange Technical Reference
**FTF-REQUEST-MESSAGE-INFO**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFT-RECORD-FORMAT</td>
<td>PIC X VALUE X’01’</td>
</tr>
<tr>
<td>FTFT-RECFM-FIXED</td>
<td>VALUE X’01’</td>
</tr>
<tr>
<td>FTFT-RECFM-VARIABLE</td>
<td>VALUE X’02’</td>
</tr>
<tr>
<td>FTFT-RECFM-FIXED-BLOCKED</td>
<td>VALUE X’03’</td>
</tr>
<tr>
<td>FTFT-RECFM-VARIABLE-BLOCKED</td>
<td>VALUE X’04’</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X(3) VALUE SPACES</td>
</tr>
<tr>
<td>FTFT-RECORD-LENGTH</td>
<td>PIC 9(9) BINARY VALUE 0</td>
</tr>
<tr>
<td>FTFT-BLOCK-SIZE</td>
<td>PIC 9(9) BINARY VALUE 1</td>
</tr>
<tr>
<td>FTFT-UNIT-NAME</td>
<td>PIC X(8) VALUE SPACES</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X VALUE X’00’</td>
</tr>
<tr>
<td>FTFT-VOL-SER</td>
<td>PIC X(6) VALUE SPACES</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X VALUE X’00’</td>
</tr>
<tr>
<td>FTFT-ALLOCATION</td>
<td>PIC X VALUE X’03’</td>
</tr>
<tr>
<td>FTFT-ALLOCATION-UNIT-CYL</td>
<td>VALUE X’01’</td>
</tr>
<tr>
<td>FTFT-ALLOCATION-UNIT-BLK</td>
<td>VALUE X’02’</td>
</tr>
<tr>
<td>FTFT-ALLOCATION-UNIT-TRK</td>
<td>VALUE X’03’</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X(3) VALUE SPACES</td>
</tr>
<tr>
<td>FTFT-PRIMARY-ALLOC</td>
<td>PIC 9(9) BINARY VALUE 5</td>
</tr>
<tr>
<td>FTFT-SECOND-ALLOC</td>
<td>PIC 9(9) BINARY VALUE 2</td>
</tr>
<tr>
<td>FTFT-TEXT-WRAP-INFO</td>
<td>PIC X VALUE X’00’</td>
</tr>
<tr>
<td>FTFT-TEXT-WRAP</td>
<td>VALUE X’01’</td>
</tr>
<tr>
<td>FTFT-FAIL</td>
<td>VALUE X’02’</td>
</tr>
<tr>
<td>FTFT-TRUNC</td>
<td>VALUE X’03’</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X VALUE SPACES</td>
</tr>
<tr>
<td>FTFT-CREATE-DIR</td>
<td>PIC 9(4) BINARY VALUE 0</td>
</tr>
<tr>
<td>FTFT-ISDATA-PERSIST</td>
<td>PIC 9(4) BINARY VALUE 0</td>
</tr>
<tr>
<td>FTFT-MODEL-DATASET</td>
<td>PIC 9(44) VALUE SPACES</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X VALUE X’00’</td>
</tr>
<tr>
<td>FTFT-DTYPE</td>
<td>PIC 9(26) VALUE SPACES</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X(3) VALUE SPACES</td>
</tr>
<tr>
<td>FTFT-C-DTYPE</td>
<td>PIC 9(9) BINARY VALUE 0</td>
</tr>
<tr>
<td>FTFT-DTYPE-INFO-POINTER</td>
<td>USAGE IS POINTER</td>
</tr>
<tr>
<td>FTFT-BUFFNO</td>
<td>PIC 9(9) BINARY VALUE 0</td>
</tr>
</tbody>
</table>

**FTF-JOB-INFO**

**Description**

This data structure represents the rules governing the associated data-transfer request. It contains information about the transfer’s pool, priority, recovery capabilities, message size, and staging.
COBOL Data Structures

**FTF-REQUEST-MESSAGE-INFO**

**Data Elements**

The following table lists and describes the FTF-JOB-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFJ-POOL-NAME</td>
<td>X(48)</td>
<td>Represents the target pool name for the data associated with the data-transfer request. For more information about using pools, see the Tivoli Data Exchange User's Guide, “About Pools.”</td>
</tr>
<tr>
<td>FTFJ-PRIORITY</td>
<td>S9(9) BINARY</td>
<td>Represents the data-transfer request’s priority. <strong>Valid values:</strong> 1 (highest) - 5 (lowest).</td>
</tr>
<tr>
<td>FTFJ-RETRIES</td>
<td>S9(9) BINARY</td>
<td>Represents the number of retry attempts for the data-transfer request.</td>
</tr>
<tr>
<td>FTFJ-EXP-DATE-TIME</td>
<td>S9(9) BINARY</td>
<td>Represents the number of seconds between the current time and the time that the data-transfer request expires.</td>
</tr>
<tr>
<td>FTFJ-MQM-MSG-SIZE</td>
<td>S9(9) BINARY</td>
<td>Represents the size of each MQSeries message in the data-transfer request. This value is represented in bytes.</td>
</tr>
<tr>
<td>FTFJ-IS-TRUSTED</td>
<td>9(4) BINARY</td>
<td>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 Tivoli Data Exchange protection), Tivoli Data Exchange does not perform syncpoint control. In the event of a failure, the data-transfer request is terminated with a failing notification. <strong>Valid values:</strong> 1 (trusted), 0 (not trusted). <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>FTFJ-RESERVED</td>
<td>9(4) BINARY</td>
<td>Reserved. Do not use this data element.</td>
</tr>
<tr>
<td>FTFJ-IS-STAGE-ONLY</td>
<td>9(4) BINARY</td>
<td>Determinates whether the source information is to be sent only to a staging queue and not to another destination. <strong>Valid values:</strong> 1 (send only to a staging queue), any other value (do not send staging queue)</td>
</tr>
</tbody>
</table>
## COBOL Data Structures

### FTF-REQUEST-MESSAGE-INFO

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFJ-IS-STAGEDFTFID</td>
<td>9(4) BINARY</td>
<td>Determines that FTFS-SPATH data element in the FTF-REQUEST-MESSAGE-INFO data structure is a staged FTFID. This value should be set to false unless the file is being taken from the staging queue and is specified by an FTFID.</td>
</tr>
<tr>
<td>FTFJ-IS-STAGED-FILE</td>
<td>9(4) BINARY</td>
<td>Determines that FTFS-SPATH data element in the FTF-REQUEST-MESSAGE-INFO data structure is a staged filename. This value should be set to false unless the file is being taken from the staging queue and is specified by a filename.</td>
</tr>
<tr>
<td>FTFJ-CANCEL-MODE</td>
<td>X</td>
<td>Determines how preemptive cancellation should be handled.</td>
</tr>
<tr>
<td>FTFJ-DELETE-SOURCE</td>
<td>9(4)</td>
<td>Determines whether the source file will be deleted after the data transfer.</td>
</tr>
<tr>
<td>FTF-USER-FTFR-IDENTIFIERS</td>
<td></td>
<td>Designates the identifier fields.</td>
</tr>
<tr>
<td>FTFJ-ID1</td>
<td>X(50)</td>
<td>Contains the first text field that is associated with the message.</td>
</tr>
<tr>
<td>FTFJ-ID2</td>
<td>X(50)</td>
<td>Contains the second text field that is associated with the message.</td>
</tr>
<tr>
<td>FTFJ-ID3</td>
<td>X(50)</td>
<td>Contains the third text field that is associated with the message.</td>
</tr>
<tr>
<td>FTFJ-IMMEDIATE</td>
<td>9(4) BINARY</td>
<td>Indicates that the data transfer is to be treated as expedited rather than use the normal process.</td>
</tr>
<tr>
<td>FTFJ-XML-FORMAT</td>
<td>9(4) BINARY</td>
<td>Indicates that the data is in XML format.</td>
</tr>
</tbody>
</table>
### FTF-REQUEST-MESSAGE-INFO

#### COBOL Data Structures

**Data Structure**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 FTF-JOB-INFO.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 FTFJ-POOL-NAME</td>
<td>PIC X(48)</td>
<td>VALUE SPACES.</td>
</tr>
<tr>
<td>10 FILLER</td>
<td>PIC X</td>
<td>VALUE X’00’.</td>
</tr>
<tr>
<td>10 FILLER</td>
<td>PIC X(3)</td>
<td>VALUE SPACES.</td>
</tr>
<tr>
<td>10 FTFJ-PRIORITY</td>
<td>PIC S9(9) BINARY</td>
<td>VALUE 3.</td>
</tr>
<tr>
<td>10 FTFJ-RETRIES</td>
<td>PIC S9(9) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-EXP-DATE-TIME</td>
<td>PIC S9(9) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-MQM-MSG-SIZE</td>
<td>PIC S9(9) BINARY</td>
<td>VALUE 524288.</td>
</tr>
<tr>
<td>10 FTFJ-IS-TRUSTED</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-RESERVED</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 1.</td>
</tr>
<tr>
<td>10 FTFJ-IS-STAGE-ONLY</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-IS-STAGEDFTFID</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-IS-STAGED-FILE</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-CANCEL-MODE</td>
<td>PIC X</td>
<td>VALUE X’00’.</td>
</tr>
<tr>
<td>10 FTFJ-DELETE-SOURCE</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-ID1</td>
<td>PIC X(50)</td>
<td>VALUE LOW-VALUES.</td>
</tr>
<tr>
<td>10 FILLER</td>
<td>PIC X</td>
<td>VALUE X’00’.</td>
</tr>
<tr>
<td>10 FTFJ-ID2</td>
<td>PIC X(50)</td>
<td>VALUE LOW-VALUES.</td>
</tr>
<tr>
<td>10 FILLER</td>
<td>PIC X</td>
<td>VALUE X’00’.</td>
</tr>
<tr>
<td>10 FTFJ-ID3</td>
<td>PIC X(50)</td>
<td>VALUE LOW-VALUES.</td>
</tr>
<tr>
<td>10 FILLER</td>
<td>PIC X</td>
<td>VALUE X’00’.</td>
</tr>
<tr>
<td>10 FTFJ-IMMEDIATE</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
<tr>
<td>10 FTFJ-XML-FORMAT</td>
<td>PIC 9(4) BINARY</td>
<td>VALUE 0.</td>
</tr>
</tbody>
</table>
FTF-USER-INFO

Description

This data structure allows you to specify group and label information about the current data-transfer request.

Data Elements

The following table lists and describes the FTF-USER-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFU-GROUP-NAME</td>
<td>X(20)</td>
<td>Represents the name of the group to which this data-transfer request belongs.</td>
</tr>
<tr>
<td>FTFU-LABEL</td>
<td>X(20)</td>
<td>Represents a user-specified label that can be used in the Tivoli Data Exchange status queries. This value has no bearing on the internal processing of a data-transfer request and is only for output and query specifications.</td>
</tr>
<tr>
<td>REPLYQ</td>
<td>X(48)</td>
<td>Represents the queue to which reply messages are to be routed.</td>
</tr>
<tr>
<td>REPLYQMG</td>
<td>X(48)</td>
<td>Represents the queue manager to which reply messages are to be routed.</td>
</tr>
<tr>
<td>NOTIFYSTATUS</td>
<td>S9(4) BINARY</td>
<td>Defines when a notification message will be sent to the NotifyQueue. The notification is sent if the transaction’s status matches the status specified. If you specify this, you must also specify NOTIFYDATA and NOTIFYTYPE. Valid values: Success, Failure, Nonsuccess (includes failed, cancelled, expired)</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

*FTF-REQUEST-MESSAGE-INFO*

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTIFYTYPE</td>
<td>X(25)</td>
<td>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, you must also specify NOTIFYDATA and NOTIFYTYPE.</td>
</tr>
<tr>
<td>NOTIFYDATA-POINTER</td>
<td>POINTER</td>
<td>Pointer to the 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, you must also specify NOTIFYSTATUS and NOTIFYTYPE.</td>
</tr>
</tbody>
</table>

**Data Structure**

```
05 FTF-USER-INFO.
   10 FTFU-GROUP-NAME PIC X(20) VALUE SPACES.
   10 FILLER PIC X VALUE X'00'.
   10 FTFU-LABEL PIC X(20) VALUE LOW-VALUES.
   10 FILLER PIC X VALUE X'00'.
   10 REPLYQ PIC X(48) VALUE LOW-VALUES.
   10 FILLER PIC X VALUE X'00'.
   10 REPLYQMGR PIC X(48) VALUE LOW-VALUES.
   10 FILLER PIC X VALUE X'00'.
   10 NOTIFYSTATUS PIC S9(4) BINARY VALUE 0.
   10 NOTIFYTYPE PIC X(25) BINARY VALUE LOW-VALUES.
   10 NOTIFYDATA-POINTER USAGE IS POINTER.
```
<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-IS-REPLY</td>
<td>9(4) BINARY</td>
<td>Determines whether Tivoli Data Exchange should wait for a reply before returning control to the calling application.</td>
</tr>
<tr>
<td>FTF-TIMEOUT</td>
<td>9(9) BINARY</td>
<td>Determines the length of time (in seconds) the data-transfer request is to stay active before it completes.</td>
</tr>
<tr>
<td>FTF-REPLY-WAIT-TIME</td>
<td>9(9) BINARY</td>
<td>Determines the length of time (in seconds) to wait for a reply before control is returned to the calling application.</td>
</tr>
<tr>
<td>FTF-CONFIG-FILE-POINTER</td>
<td>POINTER</td>
<td>Pointer to a variable that contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTF-C-EXIT-INFO</td>
<td>S9(9) BINARY</td>
<td>Determines the number of exits invoked during this data-transfer request.</td>
</tr>
<tr>
<td>FTF-EXIT-INFO-POINTER</td>
<td>POINTER</td>
<td>Pointer to a variable that contains the name of a data structure that governs how user exits are invoked and executed.</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

*FTF-REQUEST-MESSAGE-INFO*

**FTFAS400-FILE-INFO**

**Description**

This data structure contains information about target files that will be written to the AS/400 platform.

**Data Elements**

The following lists and describes the FTF-AS400-FILE-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFAS-CCSID</td>
<td>X(5)</td>
<td>Contains the CCSID value used as an identifier for the data-transfer request. If no CCSID is specified, Tivoli Data Exchange uses the job’s CCSID.</td>
</tr>
<tr>
<td>FTFAS-RECORD-LENGTH</td>
<td>S9(9) BINARY</td>
<td>Determines the logical record length for the AS/400 target file. <strong>Valid values:</strong> 1-32760</td>
</tr>
<tr>
<td>FTFAS-FTF-FILE-TYPE</td>
<td>X</td>
<td>Contains a hexadecimal value that represents the type of file being written to the AS/400 target. <strong>Valid values:</strong> FTF_BINARY, FTF_TEXT</td>
</tr>
<tr>
<td>FTFAS-FILE-TYPE-400</td>
<td>X</td>
<td>Contains a hexadecimal value that represents the AS/400 file type for the target file. <strong>Valid values:</strong> FTF-BAD X’00’, FTF-TYPE-SAVE X’01’, FTF-TYPE-SRC X’02’, FTF-TYPE-IFS X’03’, FTF-TYPE-QSYSLIB X’04’, FTF-TYPE-PHYSICAL X’05’, FTF-TYPE-LOGICAL X’06’, FTF-TYPE-DATABASE X’07’</td>
</tr>
<tr>
<td>FTFAS-CREATE-LIBRARY</td>
<td>9(4) BINARY</td>
<td>Determines whether Tivoli Data Exchange should create the specified library if it does not already exist.</td>
</tr>
<tr>
<td>FTFAS-LIB-ASP</td>
<td>S9(9) BINARY</td>
<td>Specifies the library ASP for a library created as part of the data-transfer request.</td>
</tr>
<tr>
<td>FTFAS-FILE-ASP</td>
<td>S9(9) BINARY</td>
<td>Specifies the file ASP for a file created as part of the data-transfer request.</td>
</tr>
</tbody>
</table>
### Data Structure

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFA-LIB-TEXT</td>
<td>X(50)</td>
<td>Specifies the description for a library created as part of the data-transfer request.</td>
</tr>
<tr>
<td>FTFA-FILE-TEXT</td>
<td>X(50)</td>
<td>Specifies the description for a file created as part of the data-transfer request.</td>
</tr>
</tbody>
</table>
### FTF ID

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-TFTFID</td>
<td>X(37)</td>
<td>Contains the FTFID that identifies the current data-transfer request. This value is generated as part of the data-transfer request.</td>
</tr>
<tr>
<td>FTF-CQ-POINTER</td>
<td>POINTER</td>
<td>Contains a pointer to a variable (located in FTF-REQUEST-FIELDS) that contains the name of the configuration queue used to store configuration file information. If you are using a configuration queue to store configuration information, you must populate this data element.</td>
</tr>
</tbody>
</table>

### FTF REQUEST FIELDS

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFS-SPATH</td>
<td>X(256)</td>
<td>Contains the fully qualified path and filename of the source file.</td>
</tr>
<tr>
<td>FTFT-DPATH</td>
<td>X(256)</td>
<td>Contains the fully qualified path and filename of the destination file.</td>
</tr>
<tr>
<td>FTF-CONFIG-FILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTF-CONFIG-QUEUE</td>
<td>X(256)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
</tbody>
</table>

### Data Structure

```cobol
*------------------------------------------------------------------
* FTF REQUEST FIELDS
*------------------------------------------------------------------
01 FTFS-SPATH         PIC X(256) VALUE SPACES.
01 FTFT-DPATH         PIC X(256) VALUE SPACES.
01 FTF-CONFIG-FILE    PIC X(256) VALUE SPACES.
01 FTF-CONFIG-QUEUE   PIC X(256) VALUE SPACES.
```
FTF-STATSUM-INFO

Description

This data structure contains the information that determines what summary status information is returned by the FTFSL API.

Data Elements

The following table lists and describes the FTF-STATSUM-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFSUM-LQM</td>
<td>X(48)</td>
<td>Contains the name of the local queue manager to which the FTFSL API has a connection. You must populate this data element on OS/390 platforms.</td>
</tr>
<tr>
<td>FTFSUM-CONFIG-FILE</td>
<td>X(256)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTFSUM-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
</tbody>
</table>
**Data Structure**

```
***********************************************************
* FTFSTATUSSUMMARY FTF-SUMMARY-INFO STRUCTURE
***********************************************************
*
01 FTF-STATSUM-INFO.
  05 FTFSUM-LQM          PIC X(48)         VALUE SPACES.
  05 FILLER             PIC X             VALUE X’00’.
  05 FTFSUM-CONFIG-FILE PIC X(256)        VALUE SPACES.
  05 FILLER             PIC X             VALUE X’00’.
  05 FTFSUM-CQ          PIC X(48)          VALUE SPACES.
  05 FILLER             PIC X             VALUE X’00’.
```

**Related Functions**

- FTFSL (page 289)
FTF-STATUS-DETAIL

Description

This 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 FTF-STATUS-DETAIL data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SD-FTFID</td>
<td>X(37)</td>
<td>Represents the unique FTFID that correlates with the data-transfer request for which the detailed status was requested.</td>
</tr>
<tr>
<td>FTF-SD-TIME-STAMP</td>
<td>S9(9) BINARY</td>
<td>Represents the date and time at which the data-transfer request was initiated. The FTFTS data type is defined in the copybook file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>FTF-SD-LOCAL-QMGR</td>
<td>X(25)</td>
<td>Contains the name of the queue manager to which the reporting component has a connection handle.</td>
</tr>
<tr>
<td>FTF-SD-COMPONENT</td>
<td>X</td>
<td>Represents the Tivoli Data Exchange component that issued the detailed status message. FTFComponent is an enumerated data type that is defined in the copybook file and represents the Tivoli Data Exchange components.</td>
</tr>
<tr>
<td>FTF-SD-STATUS</td>
<td>X</td>
<td>Represents the status associated with the current detailed message’s Tivoli Data Exchange component. The FTFStatus enumerated data type is defined in the copybook file and represents all possible statuses of the Tivoli Data Exchange subsystem.</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

*FTF-STATUS-DETAIL*

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SD-FTF-RC1</td>
<td>S9(9) BINARY</td>
<td>Represents the primary processing or return code. The last three digits correspond to the Tivoli Data Exchange return codes outlined in the <em>Tivoli Data Exchange Messages and Codes</em> manual and defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SD-FTF-RC2</td>
<td>S9(9) BINARY</td>
<td>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 Tivoli Data Exchange error occurs it may be a Tivoli Data Exchange return code.</td>
</tr>
<tr>
<td>FTF-SD-FTF-ERROR-MSG</td>
<td>X(1024)</td>
<td>Represents the message text associated with an error.</td>
</tr>
<tr>
<td>FTF-SD-CUR-MSG-NUMBER</td>
<td>S9(9) BINARY</td>
<td>Represents the current message number of the data messages created during the Tivoli Data Exchange data-transfer request. This field is populated on the messages with a status of PROCESSING that originated from the Tivoli Data Exchange Sender and the Tivoli Data Exchange Receiver.</td>
</tr>
<tr>
<td>FTF-SD-TOTAL-MSG-COUNT</td>
<td>S9(9) BINARY</td>
<td>Represents the number of data messages created during the Tivoli Data Exchange data-transfer request.</td>
</tr>
<tr>
<td>FTF-SD-CUSTOM-COMPONENT</td>
<td>X(256)</td>
<td>Can contain user-specified custom component information submitted to the Tivoli Data Exchange status subsystem.</td>
</tr>
<tr>
<td>FTF-SD-CUSTOM-STATUS-TP</td>
<td>X(256)</td>
<td>Can contain user-specified custom status information submitted to the Tivoli Data Exchange status subsystem.</td>
</tr>
</tbody>
</table>
**Data Structure**

```
***********************************************************
* FTFSTATUS DETAIL INFORMATION                          *
***********************************************************
*                                                        *
  01 FTF-STATUS-DETAIL.
      05 FTF-SD-FTFID PIC X(37).
      05 FILLER PIC X(3) VALUE SPACES.
      05 FTF-SD-TIME-STAMP PIC S9(9) BINARY.
      05 FTF-SD-LOCAL-QMGR PIC X(25).
      05 FTF-SD-COMPONENT PIC X.
      05 FTF-SD-STATUS PIC X.
      05 FILLER PIC X(3) VALUE SPACES.
      05 FTF-SD-FTF-RC1 PIC S9(9) BINARY.
      05 FTF-SD-FTF-RC2 PIC S9(9) BINARY.
      05 FTF-SD-ERROR-MSG PIC X(1024).
      05 FTF-SD-CUR-MSG-NUMBER PIC S9(9) BINARY.
      05 FTF-SD-TOTAL-MSG-COUNT PIC S9(9) BINARY.
      05 FTF-SD-CUSTOM-COMPONENT PIC X(256).
      05 FTF-SD-CUSTOM-STATUS-TP PIC X(256).
```

**Related Functions**

None

**Constant Definitions**

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-CPT-MANAGER</td>
<td>X(2)</td>
<td>Represents the Tivoli Data Exchange Manager.</td>
</tr>
<tr>
<td>FTF-CPT-SENDER</td>
<td>X(2)</td>
<td>Represents the Tivoli Data Exchange Sender.</td>
</tr>
<tr>
<td>FTF-CPT-RECEIVER</td>
<td>X(2)</td>
<td>Represents the Tivoli Data Exchange Receiver.</td>
</tr>
<tr>
<td>FTF-CPT-LOGGER</td>
<td>X(2)</td>
<td>Represents the log process.</td>
</tr>
<tr>
<td>FTF-CPT-ALL</td>
<td>X(2)</td>
<td>Represents all Tivoli Data Exchange component types.</td>
</tr>
</tbody>
</table>
**COBOL Data Structures**

*FTF-STATUS-DETAIL-LIST*

**FTF-STATUS-DETAIL-LIST**

**Description**

This data structure is for future use and represents the detailed status information that describes the complete status history for the data-transfer request associated with the FTFID that will be supplied to the FTFDL and FTFSDL APIs. It contains all status messages associated with the file-transfer request.

**Data Elements**

The following table lists and describes the FTF-STATUS-DETAIL-LIST data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SDL-ROWS</td>
<td>S9(9) BINARY</td>
<td>Represents the number of data-transfer request detail records that have been returned.</td>
</tr>
<tr>
<td>FTF-SDL-ROW-POINTER</td>
<td>POINTER</td>
<td>Represents a pointer to the first entry in the FTF-STATUS-DETAIL data structure which contains detail status records.</td>
</tr>
</tbody>
</table>

**Data Structure**

```
* FTFSTATUS DETAIL LIST
******************************************************************************
* 01 FTF-STATUS-DETAIL-LIST.
  05 FTF-SDL-ROWS   PIC S9(9) BINARY.
  05 FTF-SDL-ROW-POINTER USAGE IS POINTER.
```

**Related Functions**

For future use
**FTF-STATUS-SUMMARY**

**Description**

This data structure represents the summary status record for a data-transfer request. It contains the most recent status updates for each Tivoli Data Exchange component of the associated data-transfer request: the Tivoli Data Exchange Manager, the Tivoli Data Exchange Sender, and the Tivoli Data Exchange Receiver.

**Data Elements**

The following table lists and describes the FTF-STATUS-SUMMARY data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SS-FTFID</td>
<td>X(37)</td>
<td>Represents the unique FTFID that correlates with the data-transfer request for which the summary status was requested.</td>
</tr>
<tr>
<td>FTF-SS-TIME-STAMP</td>
<td>S9(9) BINARY</td>
<td>Represents the date and time that the data-transfer request was initiated. The FTFTS data type is defined in the copybook file. The value returned is of type time_t returned from the time() function.</td>
</tr>
<tr>
<td>FTF-SS-FTF-CODE</td>
<td>S9(9) BINARY</td>
<td>Represents the primary processing or return code. The last three digits correspond to the Tivoli Data Exchange return codes that are outlined in the <em>Tivoli Data Exchange Messages and Codes</em> manual and defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-SRC</td>
<td>S9(9) BINARY</td>
<td>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 Tivoli Data Exchange error occurs it may be a Tivoli Data Exchange return code.</td>
</tr>
<tr>
<td>FTF-SS-LOCAL-QMGR</td>
<td>X(25)</td>
<td>Contains the name of the queue manager from which the data-transfer request was initiated.</td>
</tr>
<tr>
<td>FTF-SS-ORIG-QMGR</td>
<td>X(25)</td>
<td>Contains the name of the queue manager to which the data-transfer request’s Tivoli Data Exchange Manager is connected.</td>
</tr>
</tbody>
</table>
## Tivoli Data Exchange Technical Reference

### COBOL Data Structures

#### FTF-STATUS-SUMMARY

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SS-ORIG-QMGR-STATUS</td>
<td>X</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Manager for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-ORIG-QMGR-TIMES</td>
<td>S9(9) BINARY</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Manager. The FTFTS data type is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-SOURCE-QMGR</td>
<td>X(25)</td>
<td>Contains the name of the queue manager where the source file was processed by the Tivoli Data Exchange Sender for the data-transfer request associated with the FTFID.</td>
</tr>
<tr>
<td>FTF-SS-SOURCE-QMGR-STAT</td>
<td>X</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Sender for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-SOURCE-QMGR-TIME</td>
<td>S9(9) BINARY</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Sender. The FTFTS data type is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-SOURCE-FILE-NAME</td>
<td>X(255)</td>
<td>Contains the fully qualified path and filename of the source file used in the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-TARGET-QMGR</td>
<td>X(25)</td>
<td>Contains the name of the queue manager to which data is sent. The Tivoli Data Exchange Receiver is attached to this queue manager.</td>
</tr>
<tr>
<td>FTF-SS-TARGET-QMGR-STAT</td>
<td>X</td>
<td>Contains the most recent status update from the Tivoli Data Exchange Receiver for the data-transfer request associated with the FTFID. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the copybook file.</td>
</tr>
<tr>
<td>Name</td>
<td>COBOL Picture</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FTF-SS-TARGET-QMGR-TIME</td>
<td>S9(9) BINARY</td>
<td>Contains the date and time of the most recent status update from the Tivoli Data Exchange Receiver. The FTFTS data type is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-TARGET-FILE-NAME</td>
<td>X(255)</td>
<td>Contains the fully qualified path and filename of the target file used in the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-REQ-QMGR-STATUS</td>
<td>X</td>
<td>Contains the most recent status update from the component that requested the data-transfer. The FTFStatus data type represents all possible statuses in the Tivoli Data Exchange subsystem. It is an enumerated data type that is defined in the copybook file.</td>
</tr>
<tr>
<td>FTF-SS-REQ-QMGR-TIMES</td>
<td>S9(9) BINARY</td>
<td>Contains the date and time of the status update from the Tivoli Data Exchange transaction requesting component. The FTFTS data type is defined in the ftfc.h header file.</td>
</tr>
<tr>
<td>FTF-SS-GROUP-NAME</td>
<td>X(25)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>FTF-SS-LABEL</td>
<td>X(25)</td>
<td>Represents the user-defined label that is associated with the data-transfer request.</td>
</tr>
<tr>
<td>FTF-USER-IDENTIFIERS</td>
<td></td>
<td>Designates the identifier fields.</td>
</tr>
<tr>
<td>FTF-SS-ID1</td>
<td>X(50)</td>
<td>Contains the first text field that is associated with the message.</td>
</tr>
<tr>
<td>FTF-SS-ID2</td>
<td>X(50)</td>
<td>Contains the second text field that is associated with the message.</td>
</tr>
<tr>
<td>FTF-SS-ID3</td>
<td>X(50)</td>
<td>Contains the third text field that is associated with the message.</td>
</tr>
<tr>
<td>FTF-SS-PRIORITY</td>
<td>S9(9) BINARY</td>
<td>Contains the priority code associated with the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-PERSISTENT</td>
<td>S9(9) BINARY</td>
<td>Contains the persistence code associated with the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-BYTES-SENT</td>
<td>S9(9) BINARY</td>
<td>Contains a count of the number bytes of data in the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-NO-MSGS</td>
<td>S9(9) BINARY</td>
<td>Contains a count of the number of message that make up the data-transfer request.</td>
</tr>
<tr>
<td>FTF-SS-TRANS-TIME</td>
<td>S9(9) BINARY</td>
<td>Contains the time, in seconds, that has elapsed since the data-transfer request has begun.</td>
</tr>
</tbody>
</table>
**Data Structure**

```
01 FTF-STATUS-SUMMARY.
  05 FTF-SS-FTFID PIC X(37).
  05 FILLER PIC X(3) VALUE SPACES.
  05 FTF-SS-TIME-STAMP PIC S9(9) BINARY.
  05 FTF-SS-FTF-CODE PIC S9(9) BINARY.
  05 FTF-SS-SRC PIC S9(9) BINARY.
  05 FTF-SS-LOCAL-QMGR PIC X(25).
  05 FTF-SS-ORIG-QMGR PIC X(25).
  05 FTF-SS-ORIG-QMGR-STATUS PIC X.
  05 FILLER PIC X VALUE SPACES.
  05 FTF-SS-ORIG-QMGR-TIMES PIC S9(9) BINARY.
  05 FTF-SS-SOURCE-QMGR PIC X(25).
  05 FTF-SS-SOURCE-QMGR-STAT PIC X.
  05 FILLER PIC XX VALUE SPACES.
  05 FTF-SS-SOURCE-QMGR-TIME PIC S9(9) BINARY.
  05 FTF-SS-SOURCE-FILE-NAME PIC X(255).
  05 FTF-SS-TARGET-QMGR PIC X(25).
  05 FTF-SS-TARGET-QMGR-STAT PIC X.
  05 FILLER PIC X(3) VALUE SPACES.
  05 FTF-SS-TARGET-QMGR-TIME PIC S9(9) BINARY.
  05 FTF-SS-TARGET-FILE-NAME PIC X(255).
  05 FTF-SS-REQ-QMGR-STATUS PIC X.
  05 FTF-SS-REQ-QMGR-TIMES PIC S9(9) BINARY.
  05 FTF-SS-REQUESTER-NAME PIC X(25).
  05 FTF-SS-REQUESTER-QMGR PIC X(25).
  05 FILLER PIC XX VALUE SPACES.
  05 FTF-USER-IDENTIFIERS.
    10 FTF-SS-ID1 PIC X(50) VALUE LOW-VALUES.
    10 FTF-SS-ID2 PIC X(50) VALUE LOW-VALUES.
    10 FTF-SS-ID3 PIC X(50) VALUE LOW-VALUES.
    05 FTF-SS-PRIORITY PIC S9(9) BINARY.
    05 FTF-SS-PERSISTENT PIC S9(9) BINARY.
    05 FTF-SS-BYTES-SENT PIC S9(9) BINARY.
    05 FTF-SS-NO-MSGS PIC S9(9) BINARY.
    05 FTF-SS-TRANS-TIME PIC S9(9) BINARY.
```

**Related Functions**

- FTFSL (page 289)
**FTF-STATUS-SUMMARY-FILTER**

**Description**

This data structure represents the filter information required for the FTFSL API. The FTFSL API retrieves the most recent status of the Tivoli Data Exchange components for all data-transfer requests that meet the filter requirements defined in this data structure.

Before you populate this data structure, initialize its elements to null values.

**Data Elements**

The following table lists and describes the FTF-STATUS-SUMMARY-FILTER data structure’s elements.

<table>
<thead>
<tr>
<th>Elements</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSF-FTF-ID</td>
<td>X(37)</td>
<td>Represents the unique FTFID that correlates with the file-transfer request to retrieve the current summary status.</td>
</tr>
<tr>
<td>FSF-IGNORE-ACTIVE</td>
<td>S9(9) BINARY</td>
<td>Represents a flag that tells the API to ignore file-transfer request statuses with a type of “Active”. If this data element is set to True, the API does not return the status for data-transfer requests that have a status of “Active”. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>FSF-IGNORE-COMPLETE</td>
<td>S9(9) BINARY</td>
<td>Represents a flag that tells the API to ignore data-transfer request statuses with a type of “Complete”. If this data element is set to True, the API does not return the status of data-transfer requests that have a status of “Complete”. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
</tbody>
</table>
## COBOL Data Structures

### FTF-STATUS-SUMMARY-FILTER

<table>
<thead>
<tr>
<th>Elements</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSF-IGNORE-FAILED</td>
<td>S9(9)</td>
<td>Represents a flag that tells the API to ignore data-transfer request statuses with a type of “Failed”. If this data element is set to True, the API does not return the status of data-transfer requests that have a status of “Failed”. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>FSF-IGNORE-CANCELLED</td>
<td>S9(9)</td>
<td>Represents a flag that tells the API to ignore data-transfer request statuses with a type of “Canceled”. If this data element is set to True, the API does not return the status of data-transfer requests that have a status of “Canceled”. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>FSF-IGNORE-EXPIRED</td>
<td>S9(9)</td>
<td>Represents a flag that tells the API to ignore data-transfer request statuses with a type of “Expired”. If this data structure is set to True, the API does not return the status of data-transfer requests that have a status of “Expired”. <strong>Valid values:</strong> 1 (ignore this status type), 0 (include this status type).</td>
</tr>
<tr>
<td>FSF-LOCAL-QUEUE-MGR</td>
<td>X(25)</td>
<td>Contains the name of the local queue manager for which status summary information is returned.</td>
</tr>
<tr>
<td>FSF-CASE-SENSITIVE</td>
<td>9(4) BINARY</td>
<td>Represents a flag that tells the FTFSL API to ignore the case of the source filename and the target filename data elements.</td>
</tr>
<tr>
<td>FSF-ORIGIN-QUEUE-MGR</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the name of the oqm. (The Tivoli Data Exchange Manager is connected to the oqm.) Only records with the specified oqm are included in the returned status information.</td>
</tr>
<tr>
<td>FSF-ORIGIN-START-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the earliest data-transfer start date and time included in the returned status information. The FTFSL API accepts partial date and time input. <strong>Note:</strong> This field is mandatory. <strong>Format:</strong> YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>Elements</td>
<td>COBOL Picture</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FSF-ORIGIN-END-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the latest data-transfer start date and time included in the returned status information. The FTFSL API accepts partial date and time input. Format: YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>FSF-SOURCE-QUEUE-MGR</td>
<td>X(25)</td>
<td>Contains the name of the source queue manager (sqm). (The Tivoli Data Exchange Sender is connected to the source queue manager.) Only records with the specified sqm are included in the returned status information.</td>
</tr>
<tr>
<td>FSF-SOURCE-START-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the earliest time and date at which the data-transfer request started at the Tivoli Data Exchange Sender. Only records with a value greater than the value stored in this data element are included in the returned status information. Format: YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>FSF-SOURCE-END-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the latest time and date at which the data-transfer request started at the Tivoli Data Exchange Sender. Only records with a value less than the value stored in this data element are included in the returned status information. Format: YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>FSF-SOURCE-FILE-NAME</td>
<td>X(256)</td>
<td>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.</td>
</tr>
<tr>
<td>FSF-TARGET-QUEUE-MGR</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the name of the destination queue manager (dqm). (The Tivoli Data Exchange Receiver is connected to the destination queue manager.) Only records with the specified dqm are included in the returned status information.</td>
</tr>
</tbody>
</table>

Tivoli Data Exchange Technical Reference 335
### COBOL Data Structures

*FTF-STATUS-SUMMARY-FILTER*

<table>
<thead>
<tr>
<th>Elements</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-TARGET-START-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the earliest time and date at which the data-transfer request started at the Tivoli Data Exchange Receiver. Only records with a value greater than the value stored in this data element are included in the returned status information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Format:</strong> YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>FTF-TARGET-END-TIME</td>
<td>X(25)</td>
<td>Pointer to a variable that contains the latest time and date at which the data-transfer request started at the Tivoli Data Exchange Receiver. Only records with a value less than the value stored in this data element are included in the returned status information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Format:</strong> YYYYMMDDhhmmss</td>
</tr>
<tr>
<td>FTF-TARGET-FILE-NAME</td>
<td>X(256)</td>
<td>Pointer 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.</td>
</tr>
<tr>
<td>FSF-GROUP-NAME</td>
<td>X(25)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>FSF-LABEL</td>
<td>X(25)</td>
<td>Contains the user-defined label for which status information is returned.</td>
</tr>
</tbody>
</table>
**Data Structure**

```cobol
01 FTF-STATUS-SUMMARY-FILTER.
   05 FSF-FTF-ID PIC X(37) VALUE SPACES.
   05 FILLER PIC X(3) VALUE SPACES.
   05 FSF-IGNORE-ACTIVE PIC S9(9) BINARY.
   05 FSF-IGNORE-COMPLETE PIC S9(9) BINARY.
   05 FSF-IGNORE-FAILED PIC S9(9) BINARY.
   05 FSF-IGNORE-CANCELLED PIC S9(9) BINARY.
   05 FSF-IGNORE-EXPIRED PIC S9(9) BINARY.
   05 FSF-LOCAL-QUEUE-MGR PIC X(25).
   05 FILLER PIC X VALUE SPACES.
   05 FSF-CASE-SENSITIVE PIC 9(4) BINARY.
   05 FSF-ORIGIN-QUEUE-MGR PIC X(25).
   05 FSF-ORIGIN-START-TIME PIC X(25).
   05 FSF-ORIGIN-END-TIME PIC X(25).
   05 FSF-SOURCE-QUEUE-MGR PIC X(25).
   05 FSF-SOURCE-START-TIME PIC X(25).
   05 FSF-SOURCE-END-TIME PIC X(25).
   05 FSF-SOURCE-FILE-NAME PIC X(256).
   05 FSF-TARGET-QUEUE-MGR PIC X(25).
   05 FSF-TARGET-START-TIME PIC X(25).
   05 FSF-TARGET-END-TIME PIC X(25).
   05 FSF-TARGET-FILE-NAME PIC X(256).
   05 FSF-GROUP-NAME PIC X(25).
   05 FSF-LABEL PIC X(25).
   05 FILLER PIC X(3) VALUE SPACES.
```

**Related Functions**

- FTFSL (page 289)
COBOL Data Structures
FTF-STATUS-SUMMARY-LIST

FTF-STATUS-SUMMARY-LIST

Description

The data structure represents data-transfer requests summary status information. It contains a summary record for each data-transfer request that matches the filter criteria. Each summary record contains the most recent status update from each Tivoli Data Exchange component.

Data Elements

The following data elements are found in the FTF-STATUS-SUMMARY-LIST data structure.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SSL-ROWS</td>
<td>S9(9)</td>
<td>Contains the number of returned data-transfer request summary records contained in the FTF-STATUS-SUMMARY-LIST.</td>
</tr>
<tr>
<td></td>
<td>BINARY</td>
<td></td>
</tr>
<tr>
<td>FTF-SSL-ROW-POINTER</td>
<td>POINTER</td>
<td>Contains a pointer to the first entry in the FTF-STATUS-SUMMARY-LIST data structure.</td>
</tr>
</tbody>
</table>

Data Structure

```
**********************************************************************
* FTFSTATUS SUMMARY LIST
**********************************************************************
*     01 FTF-STATUS-SUMMARY-LIST.
      05 FTF-SSL-ROWS PIC S9(9) BINARY.
      05 FTF-SSL-ROW-POINTER USAGE IS POINTER.
```

Related Functions

- FTFSL (page 289)
FTF-SHUTDOWN-INFO

Description

This data structure represents the input information required for the FTFEND API. The FTFEND API terminates specified Tivoli Data Exchange components and reports which components have actually been terminated. This input information includes the components and the target node for which the request is destined.

Data Elements

The following table lists and describes the FTF-SHUTDOWN-INFO data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SHUT-INFO-LQM</td>
<td>X(48)</td>
<td>Contains the name of the local queue manager to which the FTFEND API has a connection handle.</td>
</tr>
<tr>
<td>FTF-SHUT-INFO-NODE</td>
<td>X(48)</td>
<td>Contains the name of the queue manager to which the Tivoli Data Exchange Receiver is connected.</td>
</tr>
<tr>
<td>FTF-SHUT-INFO-COMPONENT</td>
<td>X(4)</td>
<td>Represents the component or components to which the FTFEND API will direct the shutdown request. This can be a single component or each of the components at a specified node. Valid Values: SHUTDOWN-MANAGER X’00000001’ SHUTDOWN-SENDER X’00000010’ SHUTDOWN-RECEIVER X’00000100’ SHUTDOWN-LOGGER X’00001000’ SHUTDOWN-ALL X’00001111’</td>
</tr>
<tr>
<td>FTF-SHUT-INFO-IMMEDIATE</td>
<td>9(4) BINARY</td>
<td>Determines whether Tivoli Data Exchange 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). Valid values: 1 (immediate), 0 (quiesce)</td>
</tr>
</tbody>
</table>
### COBOL Data Structures

**FTF-SHUTDOWN-INFO**

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SHUT-INFO-TIMEOUT</td>
<td>9(9) BINARY</td>
<td>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 Tivoli Data Exchange shutdown API returns the status. Otherwise, a timeout condition occurs. <strong>Valid values:</strong> 0 to ULONGMAX</td>
</tr>
<tr>
<td>FTF-SHUT-INFO-CONFIG</td>
<td>X(257)</td>
<td>Contains the fully qualified path and name of the Tivoli Data Exchange configuration file. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
<tr>
<td>FTF-SHUT-INFO-CQ</td>
<td>X(48)</td>
<td>The fully qualified name of the configuration queue. Unless you are using a configuration queue to store configuration information, this parameter is mandatory.</td>
</tr>
</tbody>
</table>
### Data Structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SHUTDOWN-INFO-LQM</td>
<td>PIC X(48)</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X</td>
<td>VALUE X'00'</td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X(2)</td>
<td>VALUE SPACES</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-NODE</td>
<td>PIC X(48)</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X</td>
<td>VALUE X'00'</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-COMPONENT</td>
<td>PIC X(4)</td>
<td></td>
</tr>
<tr>
<td>88 SHUTDOWN-MANAGER</td>
<td>VALUE X'00000001'</td>
<td></td>
</tr>
<tr>
<td>88 SHUTDOWN-SENDER</td>
<td>VALUE X'00000010'</td>
<td></td>
</tr>
<tr>
<td>88 SHUTDOWN-RECEIVER</td>
<td>VALUE X'00000100'</td>
<td></td>
</tr>
<tr>
<td>88 SHUTDOWN-LOGGER</td>
<td>VALUE X'00001000'</td>
<td></td>
</tr>
<tr>
<td>88 SHUTDOWN-ALL</td>
<td>VALUE X'00001111'</td>
<td></td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-IMMEDIATE</td>
<td>PIC 9(4) BINARY</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X</td>
<td>VALUE SPACES</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-TIMEOUT</td>
<td>PIC 9(9) BINARY</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X(2)</td>
<td>VALUE SPACES</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-CONFIG</td>
<td>PIC X(257)</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X</td>
<td>VALUE SPACES</td>
</tr>
<tr>
<td>FTF-SHUTDOWN-INFO-CQ</td>
<td>PIC X(48)</td>
<td></td>
</tr>
<tr>
<td>FILLER</td>
<td>PIC X</td>
<td>VALUE X'00'</td>
</tr>
</tbody>
</table>

### Related Functions

- FTFEND (page 280)
COBOL Data Structures

**FTF-SHUTDOWN-REPLY**

**Description**

This data structure represents the output information for the FTFEND API. The FTFEND API ends any or all of the Tivoli Data Exchange components and tracks which components have actually ended. This output information includes elements that represent replies and number of instances that should end when requested.

**Data Elements**

The following table lists and describes the FTF-SHUTDOWN-REPLY data structure’s elements.

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SR-MANAGER-REPLY</td>
<td>S9(9)</td>
<td>BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates whether the Tivoli Data Exchange Manager replied to the shutdown request. This data element is set to True if the Tivoli Data Exchange Manager has replied and False if it has not. <strong>Valid values:</strong> 1 (Tivoli Data Exchange Manager has replied), 0 (Tivoli Data Exchange Manager has not replied).</td>
</tr>
<tr>
<td>FTF-SR-SENDER-COUNT</td>
<td>S9(9)</td>
<td>BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates the number of Tivoli Data Exchange Senders that should respond to the shutdown request. This value is determined by the number of instances specified in the Tivoli Data Exchange configuration file. <strong>Valid values:</strong> 0 to ULONGMAX</td>
</tr>
<tr>
<td>FTF-SR-SENDER-REPLIES</td>
<td>S9(9)</td>
<td>BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates the number of Tivoli Data Exchange Senders that have replied to the shutdown request.</td>
</tr>
<tr>
<td>FTF-SR-RECVR-COUNT</td>
<td>S9(9)</td>
<td>BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates the number of Tivoli Data Exchange Receivers that should respond to the shutdown request. This value is determined by the number of instances specified in the Tivoli Data Exchange configuration file. <strong>Valid values:</strong> 0 to ULONGMAX</td>
</tr>
</tbody>
</table>
**FTF-SHUTDOWN-REPLY**

<table>
<thead>
<tr>
<th>Name</th>
<th>COBOL Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF-SR-RECVR-REPLIES</td>
<td>S9(9) BINARY</td>
<td>Indicates the number of Tivoli Data Exchange Receivers that have replied to the shutdown request.</td>
</tr>
<tr>
<td>FTF-SR-LOGGER-COUNT</td>
<td>S9(9) BINARY</td>
<td>Indicates the number of FTFLOG processes that should respond to the shutdown request. <strong>Valid values:</strong> 0 to ULONGMAX</td>
</tr>
<tr>
<td>FTF-SR-LOGGER-REPLIES</td>
<td>S9(9) BINARY</td>
<td>Indicates the number of FTFLOG processes that have replied to the shutdown request.</td>
</tr>
</tbody>
</table>

**Data Structure**

```cobol
* FTF-SHUTDOWNREPLY STRUCTURE
* 01 FTF-SHUTDOWN-REPLY.
  05 FTF-SR-MANAGER-REPLY PIC S9(9) BINARY.
  05 FTF-SR-SENDER-COUNT PIC S9(9) BINARY.
  05 FTF-SR-SENDER-REPLIES PIC S9(9) BINARY.
  05 FTF-SR-RECVR-COUNT PIC S9(9) BINARY.
  05 FTF-SR-RECVR-REPLIES PIC S9(9) BINARY.
  05 FTF-SR-LOGGER-COUNT PIC S9(9) BINARY.
  05 FTF-SR-LOGGER-REPLIES PIC S9(9) BINARY.
```

**Related Functions**

- FTFEND (page 280)
COBOL Data Structures

FTF-SHUTDOWN-REPLY
This chapter describes the security used by Tivoli Data Exchange and MQSeries. It contains the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>346</td>
</tr>
<tr>
<td>MQSeries Security Enabling Interface</td>
<td>347</td>
</tr>
<tr>
<td>Enabling Security for Tivoli Data Exchange</td>
<td>349</td>
</tr>
</tbody>
</table>

**Assumptions**

This chapter makes the following assumptions:

- You have a working knowledge of Tivoli Data Exchange queues, queue managers, and data transfers.
- You have an understanding of MQSeries security procedures.
Introduction

This chapter discusses the security considerations required when running Tivoli Data Exchange 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.

Tivoli Data Exchange 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. Tivoli Data Exchange 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.

Tivoli Data Exchange 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 Tivoli Data Exchange or from remote platforms. As such, the Tivoli Data Exchange 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. Tivoli Data Exchange 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 Tivoli Data Exchange’s interface to the external security manager through the OS/390 Security Access Facility, also known as the SAF interface.

Modes of Security

Tivoli Data Exchange may be run in one of three modes:

No Security

No security enabled. This means that any user can access any file that Tivoli Data Exchange can access.

Level One Security

Security is enabled either for the MQSeries objects that are used by Tivoli Data Exchange 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 Tivoli Data Exchange for users. It uses the access controls supported by MQSeries for the Tivoli Data Exchange 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 Tivoli Data Exchange objects. If so, then Tivoli Data Exchange 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, while OS/2 has no security authentication in place. 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

Tivoli Data Exchange uses the Alternate User Authority facility, when enabled, to ensure that a user can access the Tivoli Data Exchange objects. This feature enables MQSeries on behalf of Tivoli Data Exchange to validate whether a user (as defined in the MQMD of the control message) has access to Tivoli Data Exchange objects. If the userID of the inbound request to the Tivoli Data Exchange is authorized to the Tivoli Data Exchange objects, the request is allowed to continue. If not, the request fails with a Tivoli Data Exchange return code 720 (FTFRCE_SECURITY_ERROR).

Tivoli Data Exchange External Security Manager

The Sender and Receiver Tivoli Data Exchange 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 Tivoli Data Exchange component accesses the file on behalf of the user and the request proceeds. If not, then the request fails with a Tivoli Data Exchange 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 Tivoli Data Exchange, the customer must enable this feature at the startup of Tivoli Data Exchange. This is done by setting the parameter OAMSecurity=On in the configuration file for Tivoli Data Exchange. By default, OAMSecurity is disabled. The customer must enable access to Alternate User Authority checking facilities to Tivoli Data Exchange using the appropriate facilities on the platform that Tivoli Data Exchange is running on. Please refer to the MQSeries System Management Guide for the platform MQSeries and Tivoli Data Exchange 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 Tivoli Data Exchange components must have access to all of the queues that Tivoli Data Exchange uses.

When OAMSecurity is enabled, the Requesting User must have access to the following queues:

- FTFRCV:CONTROL
- FTFRCV:SYNC
- FTFRCV:STAGE
Tivoli Data Exchange and MQSeries Security

Enabling Security for Tivoli Data Exchange

- FTFRCV.SYSTEM
- 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 Tivoli Data Exchange 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 Tivoli Data Exchange, 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
```
To display the access for user “USER1” on queue FTFDATA, use the following MQSeries command:

```
dspmqaut -m <QMGR> -n FTFDATA -t queue -p USER1
```

For OS/390, the following must be performed for the userID under which the Tivoli Data Exchange address space executes:

```
permit QMGR.FTF.** class(MQUEUE) 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(MQUEUE) id(user-id) access(update)
permit QMGR.FTFSDR.CONTROL class(MQUEUE) id(user-id) access(update)
permit QMGR.FTFRCV.CONTROL class(MQUEUE) id(user-id) access(update)
permit QMGR.FTFSTAT.CONTROL class(MQUEUE) id(user-id) access(update)
permit QMGR.FTFSTAT.DETAIL class(MQUEUE) id(user-id) access(update)
permit QMGR.FTFDATA.** class(MQUEUE) 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 Tivoli Data Exchange security is disabled and this statement should be used:

```
permit QMGR.FTFMGR.CONTROL class(MQUEUE) id(user-id) access(update)
```
Enabling Security for Tivoli Data Exchange

**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 Tivoli Data Exchange security is disabled and this statement should be used.

```bash
setmqaut -m QMGR -n FTFMGR.CONTROL -g anyusergroup +put +get
```

With altuser, the setting for the user changes as follows:

```bash
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 FTFRCSV.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.DETAIL -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 Tivoli Data Exchange 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.
chapter 10

XML Integration

This chapter describes the characteristics of the XML files that can be submitted to the Tivoli Data Exchange Manager to initiate Tivoli Data Exchange functions.

The chapter includes the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>353</td>
</tr>
<tr>
<td>Implementation Details</td>
<td>355</td>
</tr>
<tr>
<td>Return Objects</td>
<td>358</td>
</tr>
<tr>
<td>XML Element Names</td>
<td>359</td>
</tr>
<tr>
<td>Examples</td>
<td>372</td>
</tr>
<tr>
<td>Sample DTD</td>
<td>374</td>
</tr>
</tbody>
</table>

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.

Overview

The XML Integration allows a user/application to issue Tivoli Data Exchange commands using eXtensible Markup Language (XML) as the parameter input mechanism. Currently Tivoli Data Exchange 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 Tivoli Data Exchange
**XML Integration**

*Overview*

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 Tivoli Data Exchange 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
<?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
XML Integration
Implementation Details

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.

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFIdObject</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>FTFId</td>
<td>FTF Identifier</td>
</tr>
<tr>
<td>FTFDataObject</td>
<td>Data Object containing input data</td>
</tr>
</tbody>
</table>

**FTFReturnObject**

The success or failure of Tivoli Data Exchange to perform the requested action is indicated with an embedded FTFReturnObject. The format of the FTFReturnObject mimics the Tivoli Data Exchange 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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC1</td>
<td>Level 1 return code or 0 if successful</td>
</tr>
<tr>
<td>RC2</td>
<td>Level 2 return code or 0 if successful</td>
</tr>
<tr>
<td>FTFId</td>
<td>FTF identifier for this transaction</td>
</tr>
<tr>
<td>FTFErrMsg</td>
<td>Error text if an error occurred, otherwise blank</td>
</tr>
</tbody>
</table>
**XML Integration**

*Implementation Details*

**FTFReplyObject**

This object is returned, with the original transaction input when the replyQ and replyQMgr options are specified with a request transaction.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFId</td>
<td>FTF identifier for this transaction</td>
</tr>
<tr>
<td>TransferStatus</td>
<td>Status of transfer</td>
</tr>
</tbody>
</table>

**FTFNotifyObject**

This object is returned with the input data when the notify option is specified with a request transaction.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTFId</td>
<td>FTF identifier for this transaction</td>
</tr>
<tr>
<td>TransferStatus</td>
<td>Status of transfer</td>
</tr>
<tr>
<td>NotifyStatus</td>
<td>Status that was matched</td>
</tr>
<tr>
<td>NotifyType</td>
<td>User-defined method</td>
</tr>
<tr>
<td>NotifyData</td>
<td>User-defined data</td>
</tr>
</tbody>
</table>
XML Integration

Return Objects

**FTFPingInfo**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lqm</td>
<td>Local queue manager</td>
</tr>
<tr>
<td>Oqm</td>
<td>Originating queue manager</td>
</tr>
<tr>
<td>Sqm</td>
<td>Source queue manager</td>
</tr>
<tr>
<td>Dqm</td>
<td>Destination queue manager</td>
</tr>
<tr>
<td>Difftime</td>
<td>Time difference from start to finish</td>
</tr>
<tr>
<td>BytesWritten</td>
<td>Number of bytes written</td>
</tr>
</tbody>
</table>

**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.

```xml
<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.

```xml
<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.

```xml
<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 Tivoli Data Exchange. All following tags are placed within the FTFInputData root element.
**XML Integration**

**XML Element Names**

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:

```xml
<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:

```xml
<FromStage/>
<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 Tivoli Data Exchange 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
**XML Integration**

**XML Element Names**

- `<BufNo>` – Allows you to specify the number of internal buffers that are to be used when processing data transfers. The throughput of a Tivoli Data Exchange 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 Tivoli Data Exchange configuration file. **Valid values:** On, Off

- `<CFile>` – Contains the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange compression algorithm.

- `<CQ>` – Displays the queue from which the configuration information is to be retrieved for this Tivoli Data Exchange instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points Tivoli Data Exchange 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 Tivoli Data Exchange 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, Tivoli Data Exchange 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.
XML Integration
XML Element Names

- `<Dqm>` – Designates the destination queue manager. It is the queue manager on which the Tivoli Data Exchange 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 Tivoli Data Exchange 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 FTFRC1_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 Tivoli Data Exchange creates for a data-transfer request. **Valid values:** 1-16

- `<FileTxt>` – Specifies the file description for a library that Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange transfer. Using FTF immediate, the Tivoli Data Exchange 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 Tivoli Data Exchange data transfers require that the Receiver have all related data messages before writing the data to the disk. For example, using Tivoli Data Exchange 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.
**XML Integration**

**XML Element Names**

- `<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 Tivoli Data Exchange creates for a data-transfer request. **Valid values:** 1-16

- `<LibTxt>` – Specifies the library description for a library that Tivoli Data Exchange 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

<table>
<thead>
<tr>
<th>RecFmt Value</th>
<th>LRecl and BlkSize Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>LRecl must be equal to BlkSize</td>
</tr>
<tr>
<td>FB</td>
<td>If BlkSize does not equal 0 then BlkSize must be a multiple of LRecl</td>
</tr>
<tr>
<td>V</td>
<td>Blksize must be equal to LRecl+4. This will allow for the block descriptor word.</td>
</tr>
<tr>
<td>VB</td>
<td>If BlkSize does not equal 0, then LRecl must be no more than BlkSize–4. This will allow for the block descriptor word</td>
</tr>
</tbody>
</table>

- `<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.
XML Integration
XML Element Names

- **<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 Tivoli Data Exchange Manager operates. If this value is not specified, the Oqm is given the same value as the Lqm.
**XML Integration**

**XML Element Names**

- `<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 Tivoli Data Exchange Sender to the Tivoli Data Exchange Receiver. If this value is not specified, the default pool specified in the Tivoli Data Exchange configuration file is used. For this option to function, the specified pool must be defined in the Tivoli Data Exchange 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. Tivoli Data Exchange 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)
XML Element Names

- **<RecWrap>** – Indicates how records will be processed when they reach the target file and the records are longer than the target record length. Tivoli Data Exchange 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 Tivoli Data Exchange 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.
**XML Integration**

**XML Element Names**

- `<StagePersist>` – Specifies that the messages in the staging area are persistent. If you specify this argument, the messages still exist after a system or Tivoli Data Exchange reboot or shutdown. If you select this argument, you increase the recovery ability of Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange reboot or shutdown. If you select this argument, you increase the recovery ability of Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange Manager to indicate whether a request has succeeded or failed. If the Tivoli Data Exchange 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 Tivoli Data Exchange ping. The RequestType element value must be set to “Ping” for this type of transaction.
**XML Integration**

**XML Element Names**

- **<CFile>** – Can contain the fully qualified path and filename for the Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange subsystem by the processing Tivoli Data Exchange 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
XML Integration

XML Element Names

- `<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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange instance on this node. On OS/390 platforms, if no CFile value is specified, this value must be specified. The CQ argument points Tivoli Data Exchange 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.
**XML Integration**

**XML Element Names**

- `<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 Tivoli Data Exchange 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
XML Integration
Examples

Examples

Request

```xml
<?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>
  <LRclin>value</LRclin>
  <A1cUnit>value</A1cUnit>
  <Secondary>value</Secondary>
</FTFInputData>
```
**XML Integration Examples**

**Ping**

```xml
<?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
<?xml version='1.0'?>
<FTFInputData>
  <RequestType>Cancel</RequestType>
  <FTFId>value</FTFId>
  <Lqm>value</Lqm>
  <Oqm>value</Oqm>
  <CFile>value</CFile>
</FTFInputData>
```
Sample DTD

<!ELEMENT FTFInputData

<!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 LRec1 (#PCDATA)>  
<!ELEMENT B1kSize (#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:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>377</td>
</tr>
<tr>
<td>Multi-File Connector Installation</td>
<td>378</td>
</tr>
<tr>
<td>Connector Entry Points</td>
<td>379</td>
</tr>
<tr>
<td>Invoking the Multi-File Connector</td>
<td>380</td>
</tr>
<tr>
<td>XML Configuration File</td>
<td>386</td>
</tr>
<tr>
<td>Error Conditions</td>
<td>388</td>
</tr>
</tbody>
</table>

Assumptions

For the Tivoli Data Exchange Multi-File Connector to function, Tivoli Data Exchange 1.1.0 or higher and MQSeries must be installed.

Introduction

The Multi-File Connector allows Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data
The Multi-File Connector supports the following operating systems:

- OS/390
- Windows NT/2000
- Solaris
- AIX
- HP-UX
- OS/2

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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange 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 Tivoli Data Exchange to search the SYSPROC or library path for a file with this name.

There are five distinct entry points in the library.

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATAEX</td>
<td>This entry point is invoked at the source (exit 9) to extract the local data into Tivoli Data Exchange messages.</td>
</tr>
<tr>
<td>DATALOAD</td>
<td>This entry pairs with DATAEX at the target host, and is invoked in exit 10. It takes the messages from Tivoli Data Exchange and translates them into data stores on the local file system.</td>
</tr>
<tr>
<td>MULTIEX</td>
<td>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 spath. This must be a valid text file with appropriate syntax.</td>
</tr>
<tr>
<td>MULTLOAD</td>
<td>The MULTLOAD entry point couples with the MULTIEX entry for the target host and is specified in exit 10.</td>
</tr>
<tr>
<td>USAGE</td>
<td>This entry point displays release, version, and build from the portal.h 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.</td>
</tr>
</tbody>
</table>
Multi-File Connector
Invoking the Multi-File Connector

Invoking the Multi-File Connector

All data transfers are binary and they must contain the –type BINARY option. 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.

The following example shows a transfer of a binary file from a Windows NT 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 $DFLTQM \n-type BINARY \n-dqm MQA1 \n-spath C:/FTF/TEST/LANE1.JPG \n-dpath DEV$XK.FTFCMF.TEST.NT.JPEG \n-org PS \n-recfmt VB \n-lrecl 32760 \n-bksize 32760 \n-volser USER01 \n-unit SYSDA \n-exit 9 -exitdll FTFCMF -exitentry DATAEX \n-exit 10 -exitdll FTFCMF -exitentry DATALOAD
```

On Windows NT, 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.
**Invoking the Multi-File Connector**

**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 Windows NT platform, the last file transferred overwrites the first file transferred on the Windows NT 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 Tivoli Data Exchange Manager, Sender, and Receiver component to effect the changes.

**Output**

Transfer information is logged to the Tivoli Data Exchange status system. These can be viewed using the -format detail of the FTSTAT command line or through the Tivoli Data Exchange 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), "DEVXXK.HTFCMF.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.

```bash
```
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.

```plaintext
000001 //IDCAMSK JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID,RESTART=*  
000002 //STEP1 EXEC PGM=IDCAMS 
000003 /* 
000004 //SYSPRINT DD SYSOUT=* 
000005 //SYSIN DD * 
000006 DELETE - 
000007 (DEV$XK.FTFCMF.KSDS) PURGE  
000008 
000009 IF LASTCC < 12 - 
000010 THEN - 
000011 DEFINE CLUSTER - 
000012 (NAME(DEV$XK.FTFCMF.KSDS) -  
000013 VOLUMES(TECH01) -  
000014 RECORDS(50 25)) -  
000015 DATA - 
000016 (NAME(DEV$XK.FTFCMF.KSDS.DATA) -  
000017 KEYS(24 0) -  
000018 RECORDSIZE(3883 3883) -  
000019 FREESPACE(20 10) -  
000020 BUFFERSPACE(250 ) -  
000021 INDEX - 
000022 (NAME(DEV$XK.FTFCMF.KSDS.INDX) IMBED) 
000023 //
```
Invoking the Multi-File Connector

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
-SPATH /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).

```xml
<!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 Windows NT to OS/390:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<filelist>
  <!--Transfer a PS specifying some DCB information.-->
  <file filename="C:/FTF/TEST/small.txt">
    <dcb-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">
        <dcb-info secondaryspace="9" lrecl="3200"
          blksize="3200"
          primaryspace="7"
          DSORG="PO" VOLSER="USER04" DEVICE="SYSDA"/>
        <destfile="FTF.FTFCMF.TEST.TEXT.XMLMULTI.PDS">
          </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 Tivoli Data Exchange
2. Tivoli Data Exchange itself

Information on error conditions for MQSeries and Tivoli Data Exchange 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.

<table>
<thead>
<tr>
<th>Numeric Code</th>
<th>Error Message Text</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>In dataload of ftfcmf, exit invoked with bad rc returncodeNumber</td>
<td>DATALOAD() must be invoked with pExit-&gt;rc == 0.</td>
</tr>
<tr>
<td>1029</td>
<td>Unable to open file filename</td>
<td></td>
</tr>
<tr>
<td>1030</td>
<td>DATAEX(): error from getStatus on file filename</td>
<td></td>
</tr>
<tr>
<td>1035</td>
<td>Unable to open file filename</td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td>FTFCMF::DATAEX():Unexpected return from PutPortalHeader(), error=errorCode</td>
<td></td>
</tr>
<tr>
<td>1052</td>
<td>DATAEX():Unexpected return from CreateFileNameList(), error=errorCode</td>
<td></td>
</tr>
<tr>
<td>1053</td>
<td>FTFCMF::DATAEX():Unexpected return from PutFileHeader(), error=errorCode</td>
<td></td>
</tr>
<tr>
<td>1055</td>
<td>Unable to complete request, error=errorCode</td>
<td>If fileType==PDS then PutFileTrailer(). Else MVSToFTF() failed.</td>
</tr>
<tr>
<td>1060</td>
<td>Unexpected return from PutPortalHeader(), error=errorCode</td>
<td>Expected CONNECTOR__SUCCESS.</td>
</tr>
<tr>
<td>1065</td>
<td>Unable to complete request, error=errorCode</td>
<td>If fileType==PDS then PutFileTrailer(). Else MVSToFTF() failed.</td>
</tr>
<tr>
<td>1066</td>
<td>Unexpected status from PutFileHeader(), error=errorCode</td>
<td>Expected CONNECTOR__SUCCESS</td>
</tr>
</tbody>
</table>
## Error Conditions

<table>
<thead>
<tr>
<th>Numeric Code</th>
<th>Error Message Text</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1075</td>
<td>Unexpected status from <code>GetPortalHeader()</code>, <code>error=errorCode</code></td>
<td>Issued from Dataload entry point</td>
</tr>
<tr>
<td>1076</td>
<td>Unexpected return from <code>FTFToMVS</code>, <code>error=errorCode</code></td>
<td></td>
</tr>
<tr>
<td>1080</td>
<td>Unexpected status from <code>GetPortalHeader()</code>, <code>error=errorCode</code></td>
<td>Issued from the Multiload entry point</td>
</tr>
<tr>
<td>1081</td>
<td>Unable to read a record from the local file, <code>fileName</code></td>
<td></td>
</tr>
<tr>
<td>1082</td>
<td>Unable to read a record from MVS</td>
<td></td>
</tr>
<tr>
<td>1083</td>
<td>Unable to read a record from PS</td>
<td></td>
</tr>
<tr>
<td>1085</td>
<td>Unexpected status from <code>FTFToMVS()</code>, <code>error=errorCode</code></td>
<td></td>
</tr>
<tr>
<td>1086</td>
<td>Unexpected status from <code>GetFileHeader()</code>, <code>error=errorCode</code></td>
<td>Issued from the Multiload entry point</td>
</tr>
<tr>
<td>1175</td>
<td>Unexpected status from <code>GetFileHeader()</code>, <code>error=errorCode</code></td>
<td>Issued from the Dataload entry point</td>
</tr>
<tr>
<td>1220</td>
<td><code>DATAEX()</code>: code passed to exit entry (<code>errorCode</code>)</td>
<td><code>DATAEX()</code> must be invoked with <code>pExit-&gt;rc == 0</code></td>
</tr>
<tr>
<td>1221</td>
<td><code>FTFToMVS()</code>: Unexpected result from <code>WriteRecord()</code></td>
<td></td>
</tr>
<tr>
<td>1231</td>
<td>Unable to allocate memory</td>
<td></td>
</tr>
<tr>
<td>1293</td>
<td><code>DATAEX()</code>: failure returned from <code>PutPortalHeader()</code></td>
<td></td>
</tr>
<tr>
<td>1294</td>
<td><code>DATAEX()</code>: unable to determine file type for <code>fileName</code></td>
<td></td>
</tr>
<tr>
<td>1346</td>
<td><code>ReadRecordFTF</code>: FTFPortalReceive error (<code>errorCode1</code>) (<code>errorCode2</code>)</td>
<td></td>
</tr>
<tr>
<td>1349</td>
<td><code>GetFileHeader()</code>: FTFPortalReceive error (<code>errorCode1</code>) (<code>errorCode2</code>)</td>
<td>Call to <code>FTFPortalReceive()</code> in function <code>GetFileHeader()</code> failed</td>
</tr>
<tr>
<td>1500</td>
<td>Unable to perform <code>FTFPortalSend</code> (<code>errorCode</code>)</td>
<td></td>
</tr>
</tbody>
</table>
### Multi-File Connector

#### Error Conditions

<table>
<thead>
<tr>
<th>Numeric Code</th>
<th>Error Message Text</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1504</td>
<td>Unable to perform FTFPortalSend for RDW. rc1: (errorCode)</td>
<td></td>
</tr>
<tr>
<td>1510</td>
<td>Tried to write (number) bytes, actually wrote (number) bytes</td>
<td></td>
</tr>
<tr>
<td>1511</td>
<td>WriteRecordFTF(): called with short recordLength</td>
<td></td>
</tr>
<tr>
<td>1514</td>
<td>Tried to write RDW for (number) bytes, actually wrote (number) bytes</td>
<td></td>
</tr>
<tr>
<td>1530</td>
<td>PutPortalHeader(): Error from FTFPortalSend. rc1: (number)</td>
<td></td>
</tr>
<tr>
<td>1531</td>
<td>PutPortalHeader(): Tried to write (number) bytes, actually wrote (number) bytes</td>
<td></td>
</tr>
<tr>
<td>1533</td>
<td>PutFileHeader(): Error from FTFPortalSend. rc1: (errorCode)</td>
<td></td>
</tr>
<tr>
<td>1534</td>
<td>PutFileHeader(): Tried to write (number) bytes, actually wrote (number) bytes</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>FTFCMF::FTFToMVS(): Unexpected status from ReadRecordFTF, error errorCode</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Unable to open file (fileName)</td>
<td></td>
</tr>
<tr>
<td>2100</td>
<td>MULTIEX() called with exit (exitNumber) rather than (9)</td>
<td></td>
</tr>
<tr>
<td>2131</td>
<td>DATAEX(): empty file list returned</td>
<td>Empty file list returned by CreateFileNameList()</td>
</tr>
<tr>
<td>3020</td>
<td>FTFCMF::GetFileHeader(): Read file header length numberBytes not same as expected numberBytes</td>
<td></td>
</tr>
<tr>
<td>3021</td>
<td>PMVS00::GetFileHeader(): found invalid file header type: headerType</td>
<td></td>
</tr>
<tr>
<td>3050</td>
<td>Line is too big</td>
<td>Line is too big for the length specified in the call to getLine()</td>
</tr>
<tr>
<td>3109</td>
<td>GetPortalHeader(): header read (headerType) expected (headerType)</td>
<td></td>
</tr>
<tr>
<td>3111</td>
<td>GetPortalHeader(): invalid portal header type (headerType)</td>
<td>Expect type CONNECTOR_HDR_REC</td>
</tr>
</tbody>
</table>
## Multi-File Connector

### Error Conditions

<table>
<thead>
<tr>
<th>Numeric Code</th>
<th>Error Message Text</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>stat() failed for fileName. rc = returnCode</td>
<td>Stat() call for this filename failed</td>
</tr>
<tr>
<td>4002</td>
<td>malloc failed for numberBytes bytes\n</td>
<td></td>
</tr>
<tr>
<td>4200</td>
<td>Error opening file for output (fileName) type (fileType): fileName</td>
<td></td>
</tr>
<tr>
<td>4211</td>
<td>Unexpected status from PL_CreateDirectory for (fileName)</td>
<td>Status returned by PL_CreateDirectory() was CONNECTOR__FAILED</td>
</tr>
<tr>
<td>4213</td>
<td>error opening File (fileName); fileName</td>
<td>An Open Systems file</td>
</tr>
<tr>
<td>4219</td>
<td>error opening File (fileName); fileName</td>
<td>General file open failues</td>
</tr>
<tr>
<td>4811</td>
<td>Failed to open file (fileName); fileName</td>
<td>An Open Systems file or a PDS member</td>
</tr>
<tr>
<td>4812</td>
<td>Failed to open file (fileName); fileName</td>
<td>Type of file unknown</td>
</tr>
<tr>
<td>4813</td>
<td>Unhandled file type for file (fileName) type: fileType</td>
<td></td>
</tr>
<tr>
<td>4814</td>
<td>Translated unknown file type</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>Tried to read numberBytes bytes, only read numberBytes bytes</td>
<td></td>
</tr>
<tr>
<td>6020</td>
<td>GetAttribute(): probable parsing error with (errorCode)</td>
<td>Check to prevent continuous looping</td>
</tr>
<tr>
<td>6030</td>
<td>ParseFileList(): no valid filename tag found</td>
<td>No valid filename was found in the file list</td>
</tr>
<tr>
<td>6422</td>
<td>malloc failed for numberBytes bytes\n</td>
<td></td>
</tr>
<tr>
<td>9000</td>
<td>ParseFileList(): Unable to open file (fileName), errno: errorNumber, msg:messageNumber</td>
<td></td>
</tr>
<tr>
<td>9243</td>
<td>ParseFileList(): unable to determine file type for (fileType)</td>
<td></td>
</tr>
</tbody>
</table>
This connector is composed of two connectors, File To Message and Message To File, for two types of processing. The File To Message connector reads an input file and places the output as messages in a queue. The Message To File connector takes messages from a queue as input and outputs to a file. For ease of reference, the connector is called File To Message for the remainder of this chapter. Any difference in processing between the two is pointed out specifically (e.g., exitdata modifiers).

This chapter contains the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>File To Message Connector Installation</td>
<td>394</td>
</tr>
<tr>
<td>Arguments</td>
<td>394</td>
</tr>
<tr>
<td>Examples</td>
<td>398</td>
</tr>
<tr>
<td>Error Messages</td>
<td>399</td>
</tr>
</tbody>
</table>

**Assumptions**

For the File To Message Connector to function, Tivoli Data Exchange 1.1.0 or higher and MQSeries must be installed.
The File To Message Connector

Introduction

The File To Message Connector allows you to transfer files to Tivoli Data Exchange/ MQSeries messages or move messages to data files. This connector is invoked through standard Tivoli Data Exchange exits.

The File To Message Connector supports the following operating systems:

- OS/390
- Windows NT/2000
- Solaris
- AIX
- HP-UX

File To Message Connector Installation

The File To Message Connector is distributed and installed with the base Tivoli Data Exchange product.

See 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/Message to File connector is invoked using the standard Tivoli Data Exchange interface. Entries can be entered through the command by supplying an FTF command or by using one of the other interfaces. In each case you need to supply the values that are described below.

spath and dpath

The source path, spath, specifies the input to the connector. The spath specifies either a file or a queue. If File to Message, the source is a file. If Message to File, the source is a queue. The destination path, dpath, specifies the destination of the file or queue. For File to Message, the dpath is a queue. For Message to File, the dpath is a file.
**The File To Message Connector**

**Arguments**

**Exitdll**

The exitdll argument names the dynamic load library (DLL) that is to be used for the exit. The value for the exitdll argument for the File to Message/Message to File connector is FTFCF2M.

**Exit Entry Points**

The File To Message/Message To File dynamic load library (DLL) contains two entry points. The difference between the two entry points is that one is used to read an input file and output messages to a queue and the other to read messages from a queue and output to a file:

<table>
<thead>
<tr>
<th>Entry Points</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataExtract</td>
<td>This entry point is used to extract messages from a queue and output them to a file.</td>
</tr>
<tr>
<td>DataLoad</td>
<td>This entry point is used to read an input file and output messages to a queue.</td>
</tr>
</tbody>
</table>

**Exitdata Modifiers**

The exitdata field can be populated with modifiers, which change the behavior of the connector. When specified, modifiers should be separated by commas. The exitdata modifiers are described in the following table:

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Connector</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH=xxx</td>
<td>File to Message</td>
<td>Specifies the number of messages put to the output_message_queue before an MQ commit is executed. With Message to File, this modifier specifies the number of messages retrieved from the input_message_queue. <strong>Default: 100</strong></td>
</tr>
<tr>
<td>BROWSE</td>
<td>Message to File</td>
<td>Specifies that a nondestructive read is to be performed of the queue specified by spath. The contents of the queue are unaffected. If BROWSE is not specified, records are removed from the queue being read.</td>
</tr>
</tbody>
</table>
# Arguments

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Connector</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| CR CRLF LF NOCR | File to Message | Specifies 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 terminator(s).
  - CR - Carriage Return
  - LF - Line Feed
  - CRLF - Carriage Return and Line Feed
  - NOCR - No Carriage Return
  Default: LF |
| DELDLM    | File to Message    | Specifies that the data starting delimiter described in the STARTDLM parameter is to be deleted from the data when it is stored in the target queue. |
| ENDDLM=xxx | File to Message    | Specifies the ending delimiter for the data being transferred. Data is copied up to and including the line containing the ENDDLM delimiter. If this parameter is not specified, data is copied to the target until the end of file. |
| FIXED     | File to Message    | Specifies that the connector should break up the input file stream based on the value in RECSIZE.                                            |
| LOGFILE   | File to Message    | Specifies the default log file to be used by the connector. The fully qualified filename should be specified. If the file does not exist, it is created at runtime. |
| MSGID=msgid | Message to File    | Specifies all 24 characters of the message ID of the message that is to be read from the queue specified by spath. If this parameter is not specified, all messages, regardless of message ID, are read. |
| NOT_PERSISTENT | File to Message | Specifies that messages that are put into the output_message_queue are not persistent. Default: persistent |
| PERSISTENT | File to Message    | Specifies that messages that are put to the output_message_queue are persistent. Default: persistent |
## Arguments

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Connector</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECSIZE=x</td>
<td>File to Message</td>
<td>In FIXED mode, this specifies the record length. Any data in an input stream that occurs after this value is truncated. In SINGLE mode, the RECSIZE value is ignored and the entire input file stream is processed.</td>
</tr>
<tr>
<td></td>
<td>Message to File</td>
<td>This function is for the File to Message connector only, but it is included in the Message to File connector for backwards compatibility with previous connectors. Start (STARTDLM) and end (ENDDLM) delimiters for the RECSIZE parameter can be specified. If the incoming record size is less than specified, the record is padded with blanks; if greater than specified, the record is truncated.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>File to Message</td>
<td>Places the entire input file stream in a single message (not valid with FIXED).</td>
</tr>
<tr>
<td>SINGLE</td>
<td>Message to File</td>
<td>Specifies that only one message is to be taken off the queue.</td>
</tr>
<tr>
<td>STARTDLM=xxx</td>
<td>File to Message</td>
<td>Specifies the starting delimiter for data being transferred. Data is copied starting with the character after the delimiter. If the STARTDLM delimiter is not provided, then data is copied starting at the beginning of the file.</td>
</tr>
<tr>
<td>SYNCQ=sync_Queue</td>
<td>File to Message</td>
<td>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. <strong>Default:</strong> FTF.CONNECTOR.SYNC</td>
</tr>
</tbody>
</table>
The File To Message Connector

Examples

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Connector</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT</td>
<td>Message to File</td>
<td>Specifies that Tivoli Data Exchange should wait if no messages are found on the queue. This argument is used in conjunction with the WAITTIME argument.</td>
</tr>
<tr>
<td>WAITTIME=</td>
<td>Message to File</td>
<td>Specifies the number of seconds Tivoli Data Exchange is to wait for a message to arrive on the queue. This argument works with the WAIT argument.</td>
</tr>
<tr>
<td>xxx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

File to Message

The File To Message connector can be invoked from the command line entering an FTF command or through any of the other interfaces. The example below invokes the DataLoad entry point of the File To Message connector.

Exit number 10 specifies to Tivoli Data Exchange that this is a connector exit. After transferring the `input_file` to the FTF receiver, the user exit is executed before placing the data in the specified queue. On a Solaris platform, the `ftfcf2m` is the dynamic load library containing the user exit. The entry point into the user connector exit is DataLoad.

`output_message_queue` specifies the queue where data is to be written. If queue manager is not specified, the queue is assumed to be a local queue.

```
ftf -lqm localQM -sqm sourceQM -dqm destQM -spath
input_file -dpath output_message_queue -exit
10 -exitdll ftfcf2m -exitentry DataLoad
-exitdata SINGLE,LOGFILE=f2mlog.out, -type text
```

Message to File

The example below invokes the DataExtract entry point of the Message To File connector.
The exit number 9 option specifies to Tivoli Data Exchange that this is a connector exit and that it should execute the user exit before performing its transfer. On Solaris, ftfcf2m is the dynamic load library containing the user exit. The entry point into the user exit is DataExtract.

`input_message_queue` specifies the queue that is to be used as input for the data transfer. If a queue manager is not specified, the queue is assumed to be a local queue.

```
ftf -1qm localQM -sqm sourceQM -dqm destQM -spath
input_message_queue -dpath output_file -exit 9
-exitdll ftfcf2m -exitentry DataExtract
-exitdata SINGLE,LOGFILE=m2flog.out,WAIT,
WAITTIME=20
```

**Error Messages**

The File To Message/Message To File connector logs all activity and errors to the log file specified by the LOGFILE parameter. In addition, status is submitted to the Tivoli Data Exchange status queues.

See *Tivoli Data Exchange Messages and Codes* for information on the File To Message Connector error conditions. For MQSeries errors, see the IBM documentation.
This chapter describes the characteristics and how to use the Tivoli Data Exchange Virtual Machine (VM) Connector.

The chapter includes the following sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>402</td>
</tr>
<tr>
<td>Installing the VM Connector on Windows NT</td>
<td>405</td>
</tr>
<tr>
<td>Configuring the VM Connector</td>
<td>405</td>
</tr>
<tr>
<td>Sender Script Sample</td>
<td>410</td>
</tr>
<tr>
<td>Receiver Script Sample</td>
<td>417</td>
</tr>
<tr>
<td>Message and Error Codes</td>
<td>424</td>
</tr>
<tr>
<td>Prototypes</td>
<td>425</td>
</tr>
</tbody>
</table>

**Assumptions**

This manual makes the following assumptions:

- You have a good working knowledge of IBM MQSeries or access to MQSeries documentation.
- Tivoli Data Exchange 1.2.0 and the appropriate version of MQSeries are installed.
- Windows NT and service pack 3 or higher are installed.
- You understand the XML programming language.
Introduction

The Tivoli Data Exchange VM Connector allows you to use a VM script to write your own input and output functions for Tivoli Data Exchange transactions on the Windows NT platform. Included in the Tivoli Data Exchange installation routine is the SDK (Software Development Kit) which provides you with the VM scripting language. This tool allows you to customize your Tivoli Data Exchange transactions without incurring the development and testing cycles usually associated with a third-generation language.

The Tivoli Data Exchange CD-ROM contains sample scripts that you can customize for your enterprise:

- **ftfsdr.prg** – Accesses source file information, sends data to Tivoli Data Exchange, and submits status messages.
- **ftfrcv.prg** – Accesses target file information, gets data from Tivoli Data Exchange, and submits status messages.
- **ftfexit.prg** – Prints information and submits status messages.

The CD also contains sample XML calls (ftfsdr.xml, ftfrcv.xml, ftfexit.xml) that load a file containing a pointer to the VM script. This file contains the name of the VM script and also provides you the opportunity to add optional Tivoli Data Exchange arguments, such as wait time, persistence, compression, and other standard FTF request arguments.

The sections that follow explain how the scripts work and relate the existing Tivoli Data Exchange C structures to the XML scripts.

Overview

XML is a formal language that is used to pass information about the component parts of a document to another computer system. XML is designed to facilitate interchange of structured documents by clearly identifying the boundaries of each of the logical parts (called elements) of a document.

The VM Connector exit module must be written in VM script. The VM structures are mapped to standard Tivoli Data Exchange C data structures, which are described in this *Tivoli Data Exchange Technical Reference*. 

402  Tivoli Data Exchange Technical Reference
Tivoli Data Exchange uses the dynamic load library (DLL) facilities available on Windows NT to support the connector process. Tivoli Data Exchange components dynamically load the user-specified exit and attempt to execute the specified VM script. The Tivoli Data Exchange component uses a parameter specified as input/output to pass information to the exit. The exit then passes information back to the Tivoli Data Exchange component. The exit module can also send custom status messages to the Tivoli Data Exchange status system.

Using the Tivoli Data Exchange connector conventions, you can invoke the VM Connector from any Tivoli Data Exchange exits 3 through 8 or with the connector using exits 9 or 10. (For more information, see “Tivoli Data Exchange User Exits” in the Tivoli Data Exchange User’s Guide.) The VM Connector then initializes the VM environment.

The exit user data file contains XML elements that control the Tivoli Data Exchange transaction. This file can be used in a way similar to the Tivoli Data Exchange options file. One argument in the exit user file is the XML Object, which contains the script name. This script name is called from the XML Object and sent to the VM Connector along with source (if Sender) or target (if Receiver) and status-submitting functions. The VM Connector then interfaces with Tivoli Data Exchange to perform the functions invoked by the script.

**Tivoli Data Exchange VM Connector Data Flow**

The following diagram illustrates the flow of information when the VM Connector is used in Tivoli Data Exchange processing. P1 through P4 relate to the processing blocks in the data flow diagram.

- **P1** – The VM Connector is called from any of the standard Tivoli Data Exchange exits.
- **P2** – The VM Connector collects information from all of the Tivoli Data Exchange API structures.
- **P3** – The VM Connector invokes the Virtual Machine (VM). The VM reads the XML parameter file to get the variables that control processing.
- **P4** – The VM invokes the script named in the XML parameters file to perform the required processing.
Installing the VM Connector on Windows NT

The VM Connector is automatically installed on the Windows NT platform when you install Tivoli Data Exchange. (For information on configuring the exit environment, see “Tivoli Data Exchange User Exits” in the Tivoli Data Exchange User's Guide.) The script is a filename. Set the FTF_EXIT_PATH environment variable to the directory containing the FTFCVM.dll.

Note:
When an FTF environment variable is modified, Tivoli Data Exchange components must be stopped and restarted. The Tivoli Data Exchange components read the environment variables at startup, and changes are not recognized until the component is restarted.

Configuring the VM Connector

The following XML example includes the software versions in the header, points to the VM script to be called, and provides a holder for optional arguments to be added to the script.

```xml
<?xml version='1.0'?>
<ExitInterface>
  <ScriptName>d:\xmtest\ftfexit.prg</ScriptName>
  <test>Test Argument</test>
</ExitInterface>
```

Command-Line Arguments

The following command-line arguments are required to process a Tivoli Data Exchange transaction. The FTF command relates to the functions called by the VM script.
Virtual Machine Connector
Configuring the VM Connector

In the following sample, the FTF command invokes the FTFCVM.DLL, which then initializes the VM environment on exits 9, 10, and 3. The VMConPRG script is designated for use by the Sender on exit 9 and the Receiver on exit 3. The VMConXML script will be used on exit 10.

```
ftf -sqm TPADEV002 -dqm TPADEV002 -spath d:\spath\smt1 -dpath d:\xmtest\test.out
   -exit 9 -exitdll FTFCVM.DLL -exitentry VMConPRG -exitdata d:\xmtest\ftfsdr.prg
   -exit 10 -exitdll FTFCVM.DLL -exitentry VMConXML -exitdata d:\xmtest\ftfrcv.xml
   -exit 3 -exitdll FTFCVM.DLL -exitentry VMConPRG -exitdata "d:\xmtest\ftfexit.prg, arg1, arg2, arg3"
```

The command-line arguments are explained below.

- **sqm** – Specifies the source queue and queue manager.
- **dqm** – Specifies the destination queue and queue manager.
- **spath** – Specifies fully qualified path of the source file that is input to the connector.
- **dpath** – Specifies fully qualified path of the destination file that is output to the connector.
- **exit** – Contains the exit number being called.
- **exitdll** – Contains the name of the VM connector dynamic load library to be used for the exit.
- **exitentry** – Contains the name of the script in the DLL that contains the exit module.
- **exitdata** – Contains the fully qualified path to file containing XML elements that control the Tivoli Data Exchange transaction.
Exit Script Sample

The exit sample script performs the following functions:

- Loads the FTFCVM.DLL and imports the functions.
- Creates a status message, which appears in the Tivoli Data Exchange status system.
- Sets the script return code, which is reported back to Tivoli Data Exchange.
- Prints the script information.

These functions are highlighted in the ftfexit.prg script that follows.

Loading the DLL and Importing Functions

The script example (ftfexit.prg) shows the loading of the DLL and importing of the status message and printing functions.

```plaintext
var dllHandle;
    dllHandle = loadModule("FTFCVM.DLL");

import long CB_print(char*) from dllHandle;
import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;
```

Creating a Status Message

When processing has started, the VM script provides a status message. The callback function then submits the message to the Tivoli Data Exchange status system.

Setting the Return Code

The script return code is reported back to Tivoli Data Exchange. Any non-zero value causes the transfer to fail.

Printing Information

The Exit Interface object is printed.
Virtual Machine Connector
Configuring the VM Connector

Sample Exit Script

```c
main()
{

    // Load the DLL and import the functions
    var dllHandle;
    dllHandle = loadModule("FTFCVM.DLL");

    import long CB_print(char*) from dllHandle;
    import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;
}
```
Create a status message - this message will show up in the Tivoli status sub-system.

```
object StatusMessage;
StatusMessage.CustomComponent="VM_SCRIPT";
StatusMessage.CustomStatusType="VM_PROCESSING_STARTED";
StatusMessage.ErrorText="This is the exit program";
CB_FTFSubmitStatusMsg("StatusMessage");
```

Then:
```
if(StatusMessage.ReturnObject.RC1!=0)
{
   ExitInterface.ScriptReturnCode=10;
   print(asString("StatusMessage"));
   return;
}
```

Set the ScriptReturnCode. This rc is reported back to Tivoli. Any non-zero value will cause the transfer to fail.
```
ExitInterface.ScriptReturnCode=0;
```

Print the ExitInterface object.
```
print(asString("ExitInterface"));
```

return;
```
Virtual Machine Connector

Sender Script Sample

The Sender sample script performs the following functions:

- Loads the FTFCVM.DLL and imports the functions.
- Creates the DataObject, which passes data between the script and Tivoli Data Exchange.
- Creates a status message, which appears in the Tivoli Data Exchange status system.
- Checks the source file for file type.
- Reads data from the file and sends it to Tivoli Data Exchange via the VM connector.
- Sets the script return code, which is reported back to Tivoli Data Exchange.
- Submits a status message when processing of a source file is complete.

These functions are highlighted in the ftfsdr.prg script that follows.

Loading the DLL and Importing Functions

The script example (ftfsdr.prg) shows the loading of the DLL and importing of the data send and status message functions.

```plaintext
var dllHandle;
var sourcefile;
var buffer;
var eof;

dllHandle = loadModule("FTFCVM.DLL");
import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;
import long CB_VMConnSendData(xmoptr) from dllHandle;
```

Creating the Data Object

A data object called Exit Interface passes data between the script and Tivoli Data Exchange.
**Creating a Status Message**

When processing has started, the VM script provides a status message. The callback function then submits the message to the Tivoli Data Exchange status system.

**Checking for File Type**

The next step in processing is a check of the source file to establish if it is a text or binary transfer. This determines how the file is opened. In the example, the script checks the source file for a TEXT or BINARY file type and provides the command to open the file. If the file type is not labeled as text or binary, a message appears: “Could not open source file.”

**Reading Data and Sending to Tivoli Data Exchange**

The example continues to read data from the file and transfer it to Tivoli Data Exchange until all data has been read. If there is a problem in the transfer, the error message reads “Error on Connector Send.”

**Setting the Return Code**

The example sets the script return code, which is reported back to Tivoli Data Exchange. Any non-zero value causes the transfer to fail.

**Submitting Another Status Message**

The example submits another status message that “Processing completed normally” after all data has been read from the file.
Sample Sender Script

```plaintext
Source : ftfsdr.prg
Date : 9/1/2000
Description: This is a sample VM script, which will serve as a
Tivoli connector. This script is called by the
ftfsdr. Data is sent to Tivoli via the
VMConn connector.
This is a working sample which demonstrates the following
1) Submitting status messages
2) Accessing source file information
3) Sending data to Tivoli

FTFCVM.DLL functions:
CB_VMConnSendData(xmoptr)
CB_FTFSubmitStatusMsg(xmoptr)

******/
```
main()
{
    var dllHandle;
    var sourcefile;
    var buffer;
    var eof;

    // Load the DLL and import the functions
    dllHandle = loadModule("FTFCVM.DLL");
    import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;
    import long CB_VMConnSendData(xmoptr) from dllHandle;

    // Create the DataObject. This object is used to pass data
    // between the script and Tivoli.
    object DataObject;
    objcpy("ExitInterface","DataObject");

    // Create a status message - this message will show up in the
    // Tivoli status sub-system.
    object StatusMessage;
    StatusMessage.CustomComponent="VM_SCRIPT";
    StatusMessage.CustomStatusType="VM_PROCESSING_STARTED";
    StatusMessage.ErrorText="Processing Data";
    CB_FTFSubmitStatusMsg("StatusMessage");
    if(StatusMessage.ReturnObject.RC1!=0)
    {
        ExitInterface.ScriptReturnCode=10;
        print(asString("StatusMessage"));
        return;
    }
    delete("StatusMessage");
If (ExitInterface.SourceFileInfo.FileType == "TEXT") {
    sourcefile = openStream(ExitInterface.SourceFileInfo.FileName, "r");
} else {
    sourcefile = openStream(ExitInterface.SourceFileInfo.FileName, "rb");
}

if(sourcefile==0) {
    print("Could not open source file");
    ExitInterface.ScriptReturnCode=-99;
    closeStream(sourcefile);
    return;
}
Tivoli Data Exchange Technical Reference

Virtual Machine Connector

Sender Script Sample

/////////////////////////////////////////////////////////////
// This is the main processing loop. The loop reads data
// from the file and then sends it to Tivoli via the
// FTFCVM.DLL connector.
/////////////////////////////////////////////////////////////

while(eof!=1)
{
    buffer=readStream(sourcefile,150);
    if(buffer==0)
    {
        //End of file reached
        eof=1;
    }
    else
    {
        ExitInterface.DataObject.Data=buffer;
        //send the buffer
        CB_VMConnSendData("ExitInterface");
        if(ExitInterface.ReturnObject.RC1!=0)
        {
            print("Error on Connector Send");
            print(asString("ExitInterface"));
            ExitInterface.ScriptReturnCode=99;
            closeStream(sourcefile);
            return;
        }
        // If you want to see the ExitInterface object as the data is
        // received,
        // then uncomment the following line. (NT Only)
        //print(asString("ExitInterface"));
    }
}"}
Virtual Machine Connector

Sender Script Sample

/////////////////////////////////////////////////////////////
// Set the ScriptReturnCode. This rc is reported back to
// Tivoli. Any non-zero value will cause the transfer
// to fail.
/////////////////////////////////////////////////////////////

ExitInterface.ScriptReturnCode=0;

//Submit another status message
object StatusMessage;
StatusMessage.CustomComponent="VM_SCRIPT";
StatusMessage.CustomStatusType="VM_PROCESSING_COMPLETE";
StatusMessage.ErrorText="Processing completed normally";
CB_FTFSubmitStatusMsg("StatusMessage");
if(StatusMessage.ReturnObject.RC1!=0)
{
    ExitInterface.ScriptReturnCode=10;
    print(asString("StatusMessage"));
    closeStream(sourcefile);
    return;
}

delete("StatusMessage");

closeStream(sourcefile);
return;
} // end of main()
Receiver Script Sample

The Receiver sample script performs the following functions:

- Loads the FTFCVM.DLL and imports the functions
- Creates a status message, which appears in the Tivoli Data Exchange status system
- Checks the target file for file type
- Creates the DataObject, which passes data between the script and Tivoli Data Exchange; you must specify the number of bytes Tivoli Data Exchange should read
- Receives data from Tivoli Data Exchange and checks the return code
- Writes data to the file
- Sets the script return code, which is reported back to Tivoli Data Exchange

Loading the DLL and Importing Functions

The script example (ftfrcv.prg) shows the loading of the DLL and importing of the data receipt and status message functions.

```plaintext
var dllHandle;
var targetfile;
var BytesRead;

dllHandle = loadModule("FTFCVM.DLL");

import long CB_VMConnReceiveData(xmoptr) from dllHandle;
import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;
```

Creating a Status Message

The example creates a status message. When processing has started, the VM script can provide a status message. The callback function then submits the message to the Tivoli Data Exchange status system.
Virtual Machine Connector
Receiver Script Sample

Checking for File Type

The next step in processing is a check of the target file to establish if it is a text or binary transfer. This determines how the file is opened. In the example, the script checks the target file for a TEXT or BINARY file type and provides the command to open the file.

Creating the Data Object

The example creates a data object called Exit Interface. This object passes data between the script and Tivoli Data Exchange. You must specify how many bytes you want Tivoli Data Exchange to read.

Reading Data from a File

The example reads the data and checks the return code.

Writing Data to a File

The example writes the data from the target file out to the data file.

Setting the Return Code

The example sets the script return code, which is reported back to Tivoli Data Exchange. Any non-zero value causes the transfer to fail.

Submitting Another Status Message

The example submits another status message that “Processing completed normally” after all data has been read from the file.
Sample Receiver Script

```plaintext
//************************************************************
//* Licensed Materials - Property of Tivoli Systems *
//* Tivoli (R) Data Exchange *
//* Version 1.2.0 *
//* Product Number 5698-TDE *
//* (C) Copyright Tivoli Systems 2000. *
//* (C) Copyright CommerceQuest 2000. *
//* All rights reserved. *
//* US Government Users Restricted Rights - Use, *
//* duplication or disclosure restricted by GSA ADP *
//* Schedule Contract with IBM Corp. *
//* Tivoli is a trademark of Tivoli Systems Inc. *
*************************************************************************

** Source : ftfrcv.prg
** Date : 9/1/2000
** Description: This is a sample XMSeries script, which will serve as an
** Tivoli connector. This script is called by the
** receiver. Data is received from Tivoli via the
** VMConn connector.
**
** This is a working sample which demonstrates the following
** 1) Submitting status messages
** 2) Accessing target file information
** 3) Getting data from Tivoli
**
** FTFCVM.DLL functions:
** CB_VMConnReceiveData(xmoptr)
** CB_FTFSubmitStatusMsg(xmoptr)
**
*************************************************************************

main()
{
    var dllHandle;
    var targetfile;
    var BytesRead;
}
```
/\__________________________________________________________________________
// Load the DLL and import the functions
\\__________________________________________________________________________
dllHandle = loadModule("FTFCVM.DLL");

import long CB_VMConnReceiveData(xmoptr) from dllHandle;
import void CB_FTFSubmitStatusMsg(xmoptr) from dllHandle;

\\__________________________________________________________________________
// Create a status message - this message will show up in the
// Tivoli status sub-system.
\\__________________________________________________________________________

object StatusMessage;
StatusMessage.CustomComponent="VM_SCRIPT";
StatusMessage.CustomStatusType="VM_PROCESSING_STARTED";
StatusMessage.ErrorText="";
CB_FTFSubmitStatusMsg("StatusMessage");

// Always check your return codes.
if(StatusMessage.ReturnObject.RC1 != 0)
{
    ExitInterface.ScriptReturnCode=10;
    print(asString("StatusMessage"));
    return;
}

\\__________________________________________________________________________
// Decide on how to open up the target file. Check to see
// If this is a TEXT or BINARY transfer.
\\__________________________________________________________________________

if(ExitInterface.TargetFileInfo.FileType=="TEXT")
{
    targetfile = openStream(ExitInterface.TargetFileInfo.FileName, "w");
}
else
{
    targetfile = openStream(ExitInterface.TargetFileInfo.FileName, "wb");
}
// Create the DataObject. This object is used to pass data
// between the script and Tivoli. The user must specify
// how many bytes the user wants Tivoli to read.
object DataObject;
DataObject.BytesRequested=500;
objcpy("ExitInterface","DataObject");

//do a priming read. Be sure to check the return code

BytesRead=CB_VMConnReceiveData("ExitInterface");
if(ExitInterface.ReturnObject.RC1!=0)
{
   ExitInterface.ScriptReturnCode=10;
   print(asString("ExitInterface"));
   closeStream(targetfile);
   return;
}

// If you want to see the data as it is received, then uncomment
// the following line. (NT Only)
//print(ExitInterface.DataObject.Data);

// Write the data out to the file.
writeStream(targetfile,ExitInterface.DataObject.Data);
Virtual Machine Connector
Receiver Script Sample

////////////////////////////////////////////////////////////////////
// Loop reading and writing data until the MoreAvailable flag is set to FALSE. Be sure to check return codes.
////////////////////////////////////////////////////////////////////
while(ExitInterface.DataObject.MoreAvailable==1)
{
    BytesRead=CB_VMConnReceiveData("ExitInterface");
    if(ExitInterface.ReturnObject.RC1!=0)
    {
        ExitInterface.ScriptReturnCode=10;
        print(asString("ExitInterface"));
        closeStream(targetfile);
        return;
    }
    writeStream(targetfile,ExitInterface.DataObject.Data);
    // If you want to see the ExitInterface object as the data is received,
    // then uncomment the following line. (NT Only)
    print(asString("ExitInterface"));
}
////////////////////////////////////////////////////////////////////
// Set the ScriptReturnCode. This rc is reported back to Tivoli. Any non-zero value will cause the transfer to fail.
////////////////////////////////////////////////////////////////////
ExitInterface.ScriptReturnCode=0;

//Submit another status message
StatusMessage.CustomComponent="VM_SCRIPT";
StatusMessage.CustomStatusType="VM_PROCESSING_COMPLETE";
StatusMessage.ErrorText="Processing completed normally";
CB_FTSubmitStatusMsg("StatusMessage");
if(StatusMessage.ReturnObject.RC1!=0)
{
    ExitInterface.ScriptReturnCode=10;
    print(asString("StatusMessage"));
    closeStream(targetfile);
    return;
}
//close the stream.
closeStream(targetfile);
return;
} // end of main()
**Virtual Machine Connector**

**Message and Error Codes**

The following error and message codes are defined for the VM Connector. Any other error messages are from MQSeries or Tivoli Data Exchange. See *Tivoli Data Exchange Messages and Codes* for information on Tivoli Data Exchange error conditions. For MQSeries errors, see the IBM documentation.

<table>
<thead>
<tr>
<th>Numeric Code</th>
<th>Error Message Text</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>VMCONN_BAD_ARG_CHECK</td>
<td></td>
</tr>
<tr>
<td>5001</td>
<td>VMCONN_MEMORY_ERROR</td>
<td></td>
</tr>
<tr>
<td>5002</td>
<td>VMCONN_BAD_EXIT_DATA</td>
<td></td>
</tr>
<tr>
<td>5003</td>
<td>VMCONN_XMOBJECT_CREATE_FROM_FILE</td>
<td></td>
</tr>
<tr>
<td>5004</td>
<td>VMCONN_XMOBJECT_CREATE_EXIT_INFO</td>
<td></td>
</tr>
<tr>
<td>5005</td>
<td>VMCONN_XMOBJECT_CREATE_FILE_INFO</td>
<td></td>
</tr>
<tr>
<td>5006</td>
<td>VMCONN_XMVM_INSTANTIATE_VM</td>
<td></td>
</tr>
<tr>
<td>5007</td>
<td>VMCONN_XMVM_PASS_PARMSS</td>
<td></td>
</tr>
<tr>
<td>5008</td>
<td>VMCONN_XMVM_LOAD_SCRIPT</td>
<td></td>
</tr>
<tr>
<td>5009</td>
<td>VMCONN_XMVM_RUN_SCRIPT</td>
<td></td>
</tr>
<tr>
<td>5010</td>
<td>VMCONN_XMVM_FREE_VM</td>
<td></td>
</tr>
<tr>
<td>5011</td>
<td>VMCONN_XM_EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>5012</td>
<td>VMCONN_XMOBJECT_MISSING_REQ_ELEMENT</td>
<td></td>
</tr>
<tr>
<td>5013</td>
<td>VMCONN_XMOBJECT_CREATE_REQUEST_INFO</td>
<td></td>
</tr>
<tr>
<td>5014</td>
<td>VMCONN_XMOBJECT_MISSING_REQ_OBJECT</td>
<td></td>
</tr>
<tr>
<td>5015</td>
<td>VMCONN_XMOBJECT_INVALID_ELEMENT</td>
<td></td>
</tr>
<tr>
<td>5016</td>
<td>VMCONN_XM_SEVERE_ERROR</td>
<td></td>
</tr>
</tbody>
</table>
Prototypes

/* Prototypes for exported functions that are accessible from C */
extern "C" FTDLLEXPORT void CB_VMConnSendData(XMOPtr& pData);
extern "C" FTDLLEXPORT long CB_VMConnReceiveData(XMOPtr& pData);
extern "C" FTDLLEXPORT void CB_FTFSendTimeMsg(XMOPtr& xmpStatusMessage);
extern "C" FTDLLEXPORT void VMConXML(FTFExitInfo *pExit, void *pInfo);
extern "C" FTDLLEXPORT void VMConPRG(FTFExitInfo *pExit, void *pInfo);
extern "C" FTDLLEXPORT FTFVOID USAGE (FTFExitInfo *pExit);

USAGE Entry Point

Each DLL supports a USAGE entry point. This entry point displays release, version, and build from the connector.h 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. From the local host, issue the following FTF command:

```
ftf -lqm MYQM -dqm xx -dpath xx -spath xx
    -exit 9 -exitdll FTFCVM.DLL
    -exitentry USAGE
```

The destination queue manager (dqm) and destination path (dpath) are required on some platforms for syntax validation, but they are invalid names that will never be used in the transfer. The transfer request will fail, and the detail status message which may be shown using “ftfstat -format detail” will contain the lines:

<table>
<thead>
<tr>
<th>Component</th>
<th>FTF_MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>REQUEST_FAILED</td>
</tr>
<tr>
<td>Error exit</td>
<td>628(806)Sender:Connector send failure error in connector</td>
</tr>
</tbody>
</table>

The output log of the Sender component will contain the Usage message from the connector containing, among other things, the version and compile date of the binary.
Appendices

Contents

- Appendix B, “Tivoli Data Exchange GUI Environment Variables”
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.
The following JCL file relinks the FTFSTART command as APF authorized.

```
//************************************************************
//* Licensed Materials - Property of Tivoli Systems
//* Tivoli (R) Data Exchange
//* Version 1.2.0
//* Copyright Tivoli Systems 2000.
//* (C) Copyright CommerceQuest 2000.
//* All rights reserved.
//* US Government Users Restricted Rights - Use,
//* duplication or disclosure restricted by GSA ADP
//* Schedule Contract with IBM Corp.
//* Tivoli is a trademark of Tivoli Systems Inc.
//************************************************************
//* 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.

<table>
<thead>
<tr>
<th>High-Level Qualifier</th>
<th>Replacement Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>++QMGR++</td>
<td>The name of the queue manager to which Tivoli Data Exchange connects (for instance, MQA1)</td>
</tr>
<tr>
<td>++FTFHLQ++</td>
<td>The name of the Tivoli Data Exchange data set high-level qualifier (for instance, FTFV2)</td>
</tr>
<tr>
<td>++MQHLQ++</td>
<td>The name of the MQSeries data set high-level qualifier (for instance, SYS1.V114)</td>
</tr>
<tr>
<td>++FTFINI++</td>
<td>The fully qualified path and name of the Tivoli Data Exchange configuration file (for instance, FTFV2.INI)</td>
</tr>
<tr>
<td>++FTFOFILE++</td>
<td>The fully qualified path and name of the options file (for instance, FTFV2.OPTIONS)</td>
</tr>
</tbody>
</table>

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 TIVOLI DATA EXCHANGE V1.2.0 *
//*************************************************************
//* 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

```plaintext
// * 
//* ++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='0M' 
//* OUTCLASS='*' 
//***************************************************************
//FTFSTART PROC QMGR='++QMGR++', 
// FTFHLQ='++FTFHLQ++', 
// MQHLQ='++MQHLQ++', 
// INIFILE='++FTFINI++', 
// MEM='0M', 
// 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=&MQHLQ..SCSQAUTH,DISP=SHR 
// DD DSN=&MQHLQ..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 
//
```

432  Tivoli Data Exchange Technical Reference
This appendix lists the environment variables set in the command file used to start Tivoli Data Exchange GUI. It also describes the environment variable setting necessary if you want to run the Tivoli Data Exchange 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 installation directory.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTF_CLASS_PATH</td>
<td>This variable is used to populate the CLASSPATH variable with required Tivoli Data Exchange GUI classes. If you use this variable to set the environment variables permanently, you must include %FTF_CLASS_PATH% in your CLASSPATH environment variable. <strong>Example:</strong> %FTFGUI_HOME%tfgui.jar</td>
</tr>
<tr>
<td></td>
<td>AIX must include references to the Swing classes, the Tivoli Data Exchange GUI classes, and the JDK classes. <strong>AIX Example:</strong> %SWING_HOME%\swing.jar;%FTFGUI_HOME%\ftfgui.jar; %JDK_HOME%\lib\classes.zip</td>
</tr>
<tr>
<td>FTFGUI_HOME</td>
<td>This variable must contain the location of the Tivoli Data Exchange GUI’s Java classes. <strong>Example:</strong> c:\jdk1.2</td>
</tr>
<tr>
<td>SWING_HOME</td>
<td>This variable must contain the location of the Java Foundation Classes’ (JFC’s) Swing classes. <strong>Example:</strong> /opt/swing_1.1</td>
</tr>
<tr>
<td>(AIX only)</td>
<td></td>
</tr>
<tr>
<td>JDK_HOME</td>
<td>This variable must contain the location of the Java Development Kit (JDK) or the Java Runtime Environment (JRE). <strong>Example:</strong> /usr/jdk_base</td>
</tr>
<tr>
<td>(AIX only)</td>
<td></td>
</tr>
</tbody>
</table>
Tivoli Data Exchange GUI Environment Variables

Variable Settings Outside the Command File

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| PATH (UNIX)                       | $FTFGUI_HOME must be appended to the PATH variable.  
  Example: $PATH:$FTFGUI_HOME    |
| FTFGUI_LIBRARY_PATH (UNIX)        | This variable must contain the location of the ftfgui library.  
  Example: $HOME/FTFGUI_HOME/lib |
| LD_LIBRARY_PATH (UNIX)            | $FTFGUI_LIBRARY_PATH must be appended to the LD_LIBRARY_PATH variable.  
  Example: $LD_LIBRARY_PATH:$FTFGUI_LIBRARY_PATH |

**Variable Settings Outside the Command File**

To run the Tivoli Data Exchange GUI without using the command file, you must include the following directories in your CLASSPATH environment variable:

- the directory containing the Tivoli Data Exchange GUI’s Java classes  
  (for instance, c:\ftfgui)

If you do not use Java Platform 2 (JDK 1.2), include the following directories in your CLASSPATH environment variable to run the Tivoli Data Exchange 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 388
1029 error 388
1030 error 388
1035 error 388
1050 error 388
1052 error 388
1053 error 388
1055 error 388
1060 error 388
1065 error 388
1066 error 388
1075 error 389
1076 error 389
1080 error 389
1081 error 389
1082 error 389
1085 error 389
1086 error 389
1175 error 389
1220 error 389
1221 error 389
1231 error 389
1293 error 389
1294 error 389
1346 error 389
1349 error 389
1449 error 389
1500 error 389
1504 error 390
1510 error 390
1511 error 390
1514 error 390
1530 error 390
1531 error 390
1533 error 390
1534 error 390
2010 error 390
2012 error 390
2120 error 390
2131 error 390
3020 error 390
3021 error 390
3050 error 390
3109 error 390
3111 error 390
346 error 388
4000 error 391
4002 error 391
4200 error 391
4211 error 391
4213 error 391
4219 error 391
4811 error 391
C

4812 error 391
4813 error 391
4814 error 391
5000 error 391
5020 error 391
5030 error 391
6422 error 391
9000 error 391
9243 error 391

assigning arbitrary labels
   FTF control arguments 35

B

BATCH 395
   blksize argument
      FTF 47
      FTFLOG 101
   BlkSize element 360
   block size 47, 101
   BROWSE 395
   bufno argument
      FTF 48
      BufNo element 361

C

cancel transaction elements 370
canceling a file-transfer request
   from the API 137
cancelmode argument
   FTF 33
   CancelMode element 361
   case sensitivity
      ignore 77
   ccsid argument
      FTF 45
   CCSid element 361
   cfile argument
      FTF 31
      FTFCFG 90
      FTFCNCL 52
      FTFTERM 96
      FTFLOG 103
      FTFMGR 109
      FTFPING 58
      FTFRCV 118
      FTFS0R 125
      FTFS0RAGE 66

A

alcunit argument
   FTF 47
   FTFLOG 101
   AlcUnit element 360
allocation unit 47, 101
alternate user authority 348
append mode 41
appending data to an existing file 41
architectural overview 17
arguments
   File to Message Connector 394
      FTF 26, 27, 28, 39, 45, 50
      FTFCFG 90, 91, 94
      FTFCNCL 52, 53, 54
      FTFEND 95, 96, 97
      FTFLOG 100, 101, 102, 103, 104, 106
      FTFMGR 108, 109, 110, 111
      FTFPING 56, 57, 58, 59, 67, 68, 370
      FTFRSVC 116, 117, 118, 119, 120, 125
      FTFS0R 123, 124, 125, 126, 127
      FTFS0RAGE 64, 65, 66, 67
      FTFS0RT 129, 130, 131, 132, 133
      FTFS0RAT 75, 76, 77, 78, 79, 80
AS/400 arguments
   FTF 45
as400ft
   FTF 45
as400ft argument
   FTF 45
AS400FTElement 360
FTFSTART 131
FTFSTAT 78
CFile element 361, 369, 370
checking for file type 411, 418
clientname
example 112
COBOL applications
  DLL support 280
dynamically linking 281
COBOL data structures 285
coded character set identification
  FTF 45
  Compress argument
  FTF 40
Compress element 361
configuration file
  FTF 31
  FTFCFG 90
  FTFCNCL 52
  FTFCNCL 52
  FTFCNCL 96
  FTFCNCL 96
  FTFCNCL 104
  FTFCNCL 110
  FTFCNCL 119
  FTFCNCL 126
  FTFCNCL 132
  FTFCNCL 132
configuration queue name
  FTF 31
  FTFCNCL 53
  FTFCNCL 96
  FTFCNCL 104
  FTFCNCL 110
  FTFCNCL 58
  FTFCNCL 119
  FTFCNCL 126
  FTFCNCL 67
  FTFCNCL 132
  FTFCNCL 132
configuring the VM Connector on Windows NT
  405
Contacting Customer Support 12
conventions 13
cpt argument
  FTFCNCL 95
cq argument
  FTF 31
  FTFCNCL 53
  FTFCNCL 96
  FTFCNCL 104
  FTFCNCL 110
  FTFCNCL 119
  FTFCNCL 126
  FTFCNCL 67
  FTFCNCL 132
  FTFCNCL 132
CQ element 361, 369, 370
CR CRLF LF NOCR 396
create library
  FTF 45
create library argument
  FTF 45
create mode 41
creating a destination file 41
creating a status message 407, 411, 417
creating the data object 410, 418
CrtLib element 361
custom status information
  submitting to status subsystem 158

Tivoli Data Exchange Technical Reference  437
data elements
- FTF-STATUS-SUMMARY-FILTER 333
- FTF-AS400-FILE-INFO 320
- FTFCA 286
- FTF-CANCEL-INFO 288
- FTF-DELETE-INFO 290
- FTF-DETLIST-INFO 292
- FTF-EXIT-INFO 294, 299
- FTF-JOB-INFO 314
- FTF-PING-INFO 300
- FTFREQ-REQUEST-MESSAGE-INFO 304
- FTF-SHUTDOWN-INFO 339
- FTF-SHUTDOWN-REPLY 342
- FTF-SOURCE-FILE-INFO 305
- FTF-STATUS-DETAIL 323, 325
- FTF-STATUS-DETAIL-LIST 328
- FTF-STATUS-SUMMARY 329
- FTF-STATUS-SUMMARY-LIST 338
- FTF-TARGET-FILE-INFO 309
- FTF-USER-INFO 317, 319, 322, 327
- data object creation 410, 418
- data specification arguments
  - FTF 40, 41, 42, 43
- data structure
  - FTF-STATUS-SUMMARY-FILTER 337
  - FTF-AS400-FILE-INFO 321
  - FTFCA 287
  - FTF-CANCEL-INFO 289
  - FTF-EXIT-INFO 291, 293, 298
  - FTF-JOB-INFO 316
  - FTF-PING-INFO 302
  - FTFREQ-REQUEST-MESSAGE-INFO 305
  - FTF-SHUTDOWN-INFO 341
  - FTF-SHUTDOWN-REPLY 343
  - FTF-SOURCE-FILE-INFO 308
  - FTF-STATUS-DETAIL 324, 327
  - FTF-STATUS-DETAIL-LIST 328
  - FTF-STATUS-SUMMARY 332
  - FTF-STATUS-SUMMARY-LIST 338
- FTF-TARGET-FILE-INFO 312
- FTF-USER-INFO 318
- data transfer
  - appending data to an existing file 41
  - creating a destination file 41
- data wrapping 43
- DATAEX 379
- DATALOAD 379
- dataset
  - Generation Data Group 28
  - PDS member 28
- date argument
  - FTFSTAT 77
- date range
  - beginning 77
  - ending 77
- DBCS argument
  - FTF 40
- ddata argument
  - FTF 39, 40
- DData element 361
- default client queue naming convention 112
- DELDLM 396
- delete source file 33
- deleting status information 147
- Delsrc element 361
- destination file
  - creating 41
- destination path
  - FTF 28
  - FTFSTAT 77
- destination queue manager
  - FTF 27
  - FTPPING 56
  - FTFSTAT 77
- dirblk argument 48
- DirBlks element 361
- directory blocks 48
- DLL support
  - COBOL applications 280
- double byte character set
  - FTF 40
- dpath argument
  - FTF 28
FTFSTAT 77
Dpath element 361
dqm
and stageonly 36
dqm argument
  FTF 27
  FTPING 56
  FTFSTAT 77
Dqm element 362, 369
dtype argument
  FTF 39, 40
DType element 362

edate argument
  FTFSTAT 77
elements
  cancel transactions 370
  ping transaction 368
  request transaction 360
ENDDLM 396
entry point of DLL 38
entry points
  DATAEX 379
  DATALOAD 379
  MULTIEX 379
  MULTLOAD 379
  USAGE 379
environment arguments
  FTF 31, 32
error codes
  error code 1001 388
  error code 1029 388
  error code 1030 388
  error code 1035 388
  error code 1050 388
  error code 1052 388
  error code 1053 388
  error code 1055 388
  error code 1060 388
  error code 1065 388
  error code 1066 388
  error code 1075 389
  error code 1076 389
  error code 1080 389
  error code 1081 389
  error code 1082 389
  error code 1085 389
  error code 1086 389
  error code 1175 389
  error code 1220 389
  error code 1221 389
  error code 1231 389
  error code 1293 389
  error code 1294 389
  error code 1346 389
  error code 1349 389
  error code 1449 389
  error code 1500 389
  error code 1504 390
  error code 1510 390
  error code 1511 390
  error code 1514 390
  error code 1530 390
  error code 1531 390
  error code 1533 390
  error code 1534 390
  error code 2010 390
  error code 2012 390
  error code 2120 390
  error code 2131 390
  error code 3020 390
  error code 3021 390
  error code 3050 390
  error code 3109 390
  error code 3111 390
  error code 346 388
  error code 4000 391
  error code 4002 391
  error code 4200 391
  error code 4211 391
  error code 4213 391
  error code 4219 391
  error code 4811 391
  error code 4812 391
error code 4813 391
error code 4814 391
error code 5000 391
error code 5020 391
error code 5030 391
error code 6422 391
error code 9000 391
error code 9243 391

error conditions
Multi-File Connector 388

esidata modifiers
BATCH 395
BROWSE 395
CR CRLF LF NOCR 396
DELDLM 396
ENDDLM 396
FIXED 396
LOGFILE 396
MSGID 396
NOT_PERSISTENT 396
PERSISTENT 396
RECSIZE 397
SINGLE 397
STARTDLM 397
SYNCQ 397
WAIT 398
WAITTIME 398

esoteric unit name, setting 47, 101, 251

examples
FTFCFG 90, 91
FTFCNCL 54
FTFPING 56, 57, 59
FTFSTAGE 66, 67
FTFSTAT 75, 76, 78, 79
TERMNULL 261

exit argument
FTF 38

exit DLL
FTFSTART 129, 130

Exit element 362
exit entry point
FTFSTART 130

exit number 38

exit script sample 407

exitdata argument
FTF 39
ExitData element 362
exitdata modifiers 395
exitdll argument

FDLLINK JCL sample 282
FFTId Object 356

file argument
FTFSTAGE 65

file auxiliary storage pool
FTF 45

file auxiliary storage pool argument
FTF 45

file for staging queue 65
file id
FTFSTAGE 65
FTFSTAT 77

file id for staging queue 65
file label
FTFSTAT 77

file organization 101
file organization 101

file resource security
enabling 349

file

text

FTF 45

file
text
argument

FTF 45

File to Message Connector 395

arguments 394

exitdata modifiers 395

file type 39, 43

file type checking 411, 418

FileASP element 362

FileTxt element 362

FIXED 396

format argument

FTFSTAT 75

formatting status records

FTFSTAT 75

freeing status memory 149, 151

fromstage argument

FTF 36

FromStage element 362

FTF

alcunit 47

as400ft 45

blksize 47

bufno 48

cancelmode 33

ccsid 45

cfile 31

command 25

compress 40

cq 31

crtlib 45

DBCS 40

ddata 39, 40

delsrc 33

dirlks 48

double byte character set 40

dpath 28

dqm 27

dtype 39, 40

environment arguments 31

exit 38

exitdata 39

exitdll 38

exitentry 38

expiry argument 33

fileasp 45

filetxt 45

fromstage argument 36

ftfid 36

help argument 50

t21 34

t1 34

td 34

immed 34

label 35

libasp 45

libtxt 45

lqm 26

lrecl 48

mkdirs 41

mode argument 39, 41

mode of file processing 39, 41

model 48

msgsize 35

ofile 32

oqm 26

org 49

OS/390 arguments 47, 49

padchar argument 41

padding character 41

pool 39, 42

primary 49

priority 36

process arguments 33

queue manager arguments 26

rcdl 45

recfmt 49

recpad 39, 42

recrewrap 39, 43

dsdata 39, 43

secondary allocation unit 49

secondary argument 49

source and target file arguments 27

spath 27

specifying a destination data type 39, 40, 43

specifying a source data type 39, 43

Tivoli Data Exchange Technical Reference 441
sqm 26
stage argument 36
stageonly 36
stageonly control argument 36
stagepersist 36
standard arguments 50
stype 39, 43
syntax 26
transfer and performance arguments 39
transpersist 37
trusted 37
type 39, 43
unit name 49
user exit arguments 38, 39
version 32
volser 49
wait 37
wait control argument 37
FTF command
  append mode 41
  create mode 41
  examples 27, 37, 39, 43, 49, 51
  noreplace mode 41
FTF STATUS SUMMARY FILTER 333
FTF_CLASSPATH environment variable 433
FTF-AS400-FILE-INFO 320
FTFAS400FileInfo data structure
data elements 165
described 165
ftfauth argument
  FTFRCV 118, 125
FTFBLIB DLL 280
FTFCA 286
described 286
FTFCA data structure 165, 167
data elements 167
described 167
FTFCAN
described 265
FTFCancel function 137
description 137
parameters 138
prototype 137
FTF-CANCEL-INFO 288
FTFCancelInfo data structure
data elements 169
described 169
FTFCFG
  arguments 90, 94
cfile 90
  command 89
  configuration information 89
  examples 90, 91
  help 91
  lqm 90, 94
  node 90, 94
  ofile 91
  syntax 89, 90, 91, 94
  version 91
FTFCNCL
  arguments 53
cfile 52
  command 52
cq 53
  examples 54
  ftfid 53
  help 54
  lqm 53
  notes 54
  ofile 53
  sqm 53
  syntax 52
  version 53
FTF-DELETE-INFO 290
FTF-DETLLIST-INFO 292
FTFEND 268
cfile 96
  command 93
cpt 95
cq 96
  example 97
  help 97
  immediate 95
  notes 98
  ofile 96
  process-control arguments 94
  queue manager node arguments 94
  quiesce 95
standard arguments 95, 97
timeout 95
version 96
FTFExitAuthInfo data structure
data elements 171
description 171
FTFExitFileInfo 173
FTFExitFileInfo data structure
described 173
FTF-EXIT-INFO 290, 292, 294
FTFExitInfo data structure 177
data elements 173, 177
described 177
FTFExitJobInfo data structure
data elements 181
described 181
FTFExitQMgrsInfo data structure
data elements 184
definition 184
FTFExitRequestInfo data structure
data elements 186
description 186
FTFExitSourceFileInfo data structure
data elements 188
described 188
FTFExitStatusOffload data structure
data elements 191
described 191
FTFExitTargetFileInfo data structure
data elements 193
described 193
FTFExitUserInfo data structure
data elements 197
described 197
FTFGUI_HOME environment variable 433
ftfid argument
  FTF 36
  FTFCNCL 53
  FTFTSTAT 77
FTFId element 363, 370
FTFIdentifiersInfo data structure 199
FTF-JOB-INFO 313
FTFJobInfo data structure 200
data elements 199, 200
FTFLOG
alcunit 101
arguments 101, 102
blksize 101
cfile 103
command 99
cq 104
description 99
elements 102
help argument 106
ldir 100
lfile 100
lqm 100
lrecl 101
ofile 104
org 101
OS/390 arguments 101
primary 101
q argument 100
recfmt 102
secondary allocation unit 102
standard arguments 102
syntax 99
unit name 102
version 104
volser 102
FTFMGR
cfile argument 109
command 107
cq 110
description 108, 110
help 111
jsdataset 110
lfile 108
lqm 108
nodename 108
ofile 110
standard arguments 109, 111
startup arguments 107
syntax 107
version 110
FTFMGR - Standard Environment Arguments
F

FTFNodeAlias 112

FTFNotifyObject 357

FTFPING 55, 271
  cfile 58
  command 55
  cq 58
  dqm 56
  examples 56, 57, 59
  help argument 59, 68
  lqm 56
  msgsize 57
  notes 60
  ofile argument 58, 67
  oqm 56
  priority 57
  process control arguments 57
  queue manager node arguments 56
  sqm 56
  standard arguments 58, 59
  syntax 55
  timeout 57, 370
  uses 55, 139, 271
  version 58, 67

FTFPing function 139
  description 139
  parameters 140
  prototype 139

FTF-PING-INFO 300

FTFPingInfo 358

FTFPingInfo data structure
  data elements 203
  described 203

FTFQMgrsInfo data structure
  data elements 206
  described 206

FTFRCV
  cfile 118
  command 115
  connector arguments 120
  cq 119
  examples 117, 119, 124
  exitdll 117
  exitentry 117

ftfauth 118, 125
help 120
invoking security authorization exits 117,
124
jsdataset 119
lfile 116
lqm 116
nodename 116
ofile 119
standard arguments 118, 120, 125
sync 116
syntax 115, 122
version 119

FTFReplyObject 357

FTFREQ 274

FTFReq function
  description 141
  parameters 142
  prototype 141

FTFREQ-REQUEST-MESSAGE-INFO 303

FTFRequestMsgInfo data structure 208
  data elements 208
  described 208

FTFReturnObject 356

FTFSDR
  cfile 125
  command 122
  connector arguments 127
  cq 126
  examples 126
  exitdll 124
  exitentry 124
  help 127
  jsdataset 126
  lfile 123
  lqm 123
  nodename 123
  ofile 126
  standard arguments 127
  sync 123
  version 126

FTFSHUTDOWN function
  description 143
  parameters 144
Tivoli Data Exchange Technical Reference 445

prototype 143

FTF-SHUTDOWN-INFO 339

FTFShutdownInfo data structure
  data elements 211
  described 211

FTF-SHUTDOWN-REPLY 342

FTFShutdownReply data structure
  data elements 213
  described 213

FTFSL 277

FTF-SOURCE-FILE-INFO 305

FTFSourceFileInfo data structure
  data elements 216
  described 216

FTFSTAGE
  all argument 65
  cfile 66
  command 64
  cq 67
  examples 66, 67
  file 65
  ftfid 65
  lqm 64
  oqm 65
  purge 65
  purging all from staging queue 65
  query 65
  queue manager node arguments 64, 65
  sqm 65
  standard arguments 66, 68
  syntax 64
  wait argument 65

FTFSTAGE - Process-Control Arguments 65

FTFStage function
  description 145
  parameters 146
  prototype 145

FTFStagedListMsgInfo data structure
  data elements 219
  described 219

FTFStagedTransactionMsgInfo data structure
  data elements 220
  described 220

FTFStageMsgInfo data structure
  data elements 222
  described 222

FTFSTART
  cfile 131
  command 129
  cq 132
  examples 130, 131, 132, 133
  exitdll 129, 130
  exitentry 130
  help 133
  lqm 131
  ofile 133
  rcvauth 130
  scrauth 129
  security authorization arguments 129
  startup arguments 131
  syntax 129
  SYSOUTCLASS argument 130
  version argument 133

FTFSTAT 77
  additional arguments 76
  cfile 78
  command 74
  cq 79
  date 77
  dpath 77
  dqm 77
  examples 75, 76, 78, 79
  format 75
  ftfid 77
  help 80
  i argument 77
  ignore case sensitivity 77
  label 77
  lqm 75
  manager node arguments 75
  notes 80
  ofile 79
  oqm 77
  orphans 76
  process control arguments 75
  purge 75
  rqm 78
  samples 80
spath 78
sqm 78
standard arguments 78, 80
status type argument 78
syntax 74
version 79

FTFSTAT - Standard Environment Arguments 78

FTFStatDetail data structure 224
data elements 224
described 224

FTFStatDetailList data structure
data elements 234
described 234

FTF-STATSUM-INFO 323

FTFStatSummary data structure 235
data elements 235
description 235

FTFStatSummaryList data structure
data elements 239
description 239

FTFStatusDelete function
description 147
parameters 147
prototype 147

FTFStatusDeleteInfo data structure
data elements 241
described 241

FTF-STATUS-DETAIL 325

FTF-STATUS-DETAIL-LIST 328

FTFStatusFreeDetailList function 149
description 149
parameters 149
prototype 149

FTFStatusFreeSummaryList function
description 151
parameters 152
prototype 151

FTFStatusGetDetailList function
description 153
parameters 154
prototype 153
usage 231

FTFStatusGetDetailListInfo data structure
data elements 243
described 243

FTFStatusGetSummaryList function
description 155
parameters 156
prototype 155

FTFStatusGetSummaryListInfo data structure
data elements 245
described 245

FTF-STATUS-SUMMARY 329

FTF-STATUS-SUMMARY-FILTER 333

FTFStatusSummaryFilter data structure 247
data elements 247
described 247
usage notes 247

FTF-STATUS-SUMMARY-LIST 338

FTFSubmitStatusMessage function
description 158
parameters 159
prototype 158

FTF-TARGET-FILE-INFO 308

FTFTargetFileInfo data structure
data elements 251
described 251

FTF-USER-INFO 317

FTFUserInfo data structure 256
data elements 256
described 256

Generation Data Group
dataset 28
MVS/ESA 28
getting status information 153, 155
gradual shutdown 95

help argument
FTFCFG 90, 94
FTFCNCL 53
FTFMGR 108
FTFPING 56
FTFRCV 116
FTFSHDR 123
FTFSTAGE 64
FTFSTART 131
FTFSTAT 75
log directory 100
log file
FTFLOG 100
FTFMGR 108
FTFRCV 116
FTFSHDR 123
log information
eliminating from standard output 109, 117, 124
log queue 100
LOGFILE 396
logical record length 48, 101
lqm argument 100
FTF 26
FTFCFG 90, 94
FTFCNCL 53
FTFMGR 108
FTFPING 56
FTFRCV 116
FTFSHDR 123
FTFSTAGE 64, 75
FTFSTART 131
Lqm element 364, 369, 370
lrecl argument
FTF 48
FTFLOG 101
LRecl element 364
mkdirs argument
FTF 41
MkDirs element 364
mode argument
FTF 39, 41
Mode element 365
model 48
model argument
FTF 48
Model element 365
modes of security 346
level one security 346
level two security 347
no security 346
ModifyType element 365
MSGID 396
msgsize argument
FTF 35
FTFPING 57
MsgSize element 365, 369
MULTIEX 379
Multi-File Connector
error conditions 388
installation 378
operating systems supported 378
transferring multiple files 378
XML configuration 386
MULTLOAD 379

no security 346
node argument
FTFCFG 90, 94
node name
FTFCFG 90, 94
FTFMGR 108
nodename argument
FTFMGR 108
FTFRCV 116
FTFSHDR 123
nopad mode 42

make directory 41
message size 35
FTFPING 57
noreplace mode 41
NOT_PERSISTENT 396
NotifyData element 365
NotifyStatus element 365
number of buffers 48

O

ofile argument
FTF 32
FTFCFG 91
FTFCNCL 53
FTFEND 96
FTFLOG 104
FTFMGR 110
FTFPING 58, 67
FTFRCV 119
FTFS DR 126
FTFSTART 133
FTFSTAT 79
OFile element 365, 369, 371
operating 378
oqm argument
FTF 26
FTFCFG 91
FTFCNCL 53
FTFPING 56
FTFSTAGE 65
FTFSTAT 77
Oqm element 365, 369, 371
org argument
FTF 49
FTFLOG 101
Org element 366
originating queue manager
FTF 26
FTFCNCL 53
FTFPING 56
FTFSTAGE 65
FTFSTAT 77
orphaned detail files 76
orphans argument
FTFSTAT 76

OS/390
dataset naming conventions 28
file allocation items 195, 254
Generation Data Group 28
PDS member 28
OS/390 arguments
FTF 47, 48, 49
overrides
FTF 47, 48, 49
FTFCFG 91
FTFCNCL 53
FTFEND 96
FTFLOG 104
FTFMGR 110
FTFPING 58, 67
FTFRCV 119
FTFS DR 126
FTFSTART 133
FTFSTAT 79

P

pad mode 42
padchar argument
FTF 41
PadChar element 366
Parameters
FTFCNCL 266
FTFEND 269
FTFPING 272
FTFREQ 275
FTFS L 278
SETADDR 263
TERMNULL 262
PDS member
dataset 28
OS/390 28
PERSISTENT 396
persistent transfer 37
ping transaction elements 368
pinging components
from the API 139

Tivoli Data Exchange Technical Reference 449
placing a file in staging queue 69
pool argument
   FTF 39, 42
Pool element 366
pool name
   FTF 39, 42
preemptive cancel flag override in the configuration file 33
preemptive cancellation 33
primary argument
   FTF 49
   FTFLOG 101
Primary element 366
printing information 407
priority argument
   FTF 36
   FTPPING 57
Priority element 366, 369
priority of message 36
priority of processing
   FTPPING 57
process arguments
   FTF 33, 34, 35, 36, 37
Prototype
   FTCANCL 265
   FTFEND 268
   FTPPING 271
   FTFREQ 274
   FTPSL 277
prototypes 425
purge argument 65
   FTFSTAT 75
purging records
   FTFSTAT 75
purging the staging queue 65, 73
queue permissions limiting 349
quiesce argument
   FTFEND 95
R

Rcdlen element 366
rcvauth argument
   FTFSTART 130
reading data 411
reading data from a file 418
receiver authorization
   FTFSTART 130
receiver entry point
   FTFSTART 130
recfmt argument
   FTF 49
   FTFLOG 102
Recfmt element 366
record format 49, 102
record length
   FTF 45
record length argument
   FTF 45
record padding 42
recpad argument
   FTF 39, 42
RecPad element 366
RECSIZE 397
recwrap argument
   FTF 39, 43
RecWrap element 367
ReplyQ element 367
request transaction elements 360
Request Type Tag 355
requesting queue manager
   FTFSTART 78
required fields
   XML integration 355
return code setting 418
rqm argument

Q

q argument 100
query argument 65
querying the staging queue 65, 71

450  Tivoli Data Exchange Technical Reference
FTFSTAT 78
RwplyQMgrqmq element 367

S

sample script 407
sender 410
script sample 410
exit 407
sdata argument
FTF 39, 43
Sdata element 367
sdrauth argument
FTFSTART 129
secondary argument
FTF 49
FTFLOG 102
Secondary element 367
security
file resource 349
security authorization exits
invoking on FTFRCV 117, 124
security concepts 347
Security Enabling Interface 347
security modes 346
selsrc argument
FTF 33
sender authorization
FTFSTART 129
sender script sample 410
SenderAlwaysCheckStage property 71
sending a file
from staging queue 70
sending data to Data Integrator 411
sending messages
from stage queue 36
SETADDR
described 263
example 263
SETMQAUTH 347
setting the return code 407, 411, 418
shutdown
gradual 95
immediate 95
shutting down components 143
SIDEDECK data set 280
SINGLE 397
source path
FTF 27
FTFSTAT 78
source queue manager
FTF 26
FTFPING 56
FTFSTAGE 65
FTFSTAT 78
source type
FTF 39, 40, 43
spath argument
FTF 27
FTFSTAT 78
Spath element 367
specifying a log directory 100
specifying a log queue 100
specifying directory blocks 48
specifying the staging queue 69
sqm argument
FTF 26
FTFPING 56
FTFSTAGE 65
FTFSTAT 78
Sqm element 367, 370
stage argument
FTF 36
Stage element 367
stageonly argument
FTF 36
StageOnly element 367
stagepersist argument
FTF 36
StagePersist element 368
staging a message
FTF 36
staging messages 36
staging queue
file 65
file id 65
placing file in 69
  purging 73
  querying 71
  sending a file 70
  specifying 69

staging queue ID
  FTF control argument 36
  FTF immediate argument 34

standard output
  eliminating log information 109, 117, 124
STARTDLM 397

status
  date range 77
status information
  deleting 147
  getting 153, 155
status memory
  freeing 149, 151
status message creation 411, 417
status messages
  manager 163
  receiver 164
  sender 163
status type argument
  FTFSTAT 78
stype argument
  FTF 39, 43
SType element 368
submitting a file-transfer request
  from the API 141
submitting another status message 411, 418
SWING_HOME environment variable 433
sync argument
  FTFRCV 116
  FTFSDR 123
sync queue
  FTFRCV 116
  FTFSDR 123
SYNCQ 397
SYSOUTCLASS argument 130

T

target file block size 47, 101
target file organization 49
target file unit name
  FTF 49
  FTFLOG 102
target volume serial number 49, 102
TERMNULL 261
time limit for processing
  FTFPING 57, 370
timeout argument
  FTFEND 95
  FTFPING 57, 370
Tivoli Customer Support 12
Tivoli Data Exchange
  description 9, 10
transpersist argument
  FTF 37
Transpersist element 368
troubleshooting FTFPING 61
TRUNCATE mode 43
trued argument
  FTF 37
Trusted element 368
trued transaction 37
type argument 39, 43
Type element 368

U

unit argument
  FTF 49
  FTFLOG 102
Unit element 368
USAGE 379
user exit arguments
  FTF 38, 39
user-defined label
  FTFcommand 35
V

version argument
  FTF 32
  FTFCFG 91
  FTFCNCL 53
  FTFEND 96
  FTFLOG 104
  FTFMGR 110
  FTPPING 58, 67
  FTPRCV 119
  FTFSDR 126
  FTFSSTART 133
  FTFSSTAT 79

volser argument
  FTF 49
  FTFLOG 102

VolSer element 368

volume serial number 49, 102

VSAM data 382

W

WAIT 398

wait argument
  FTF 37
    FTPFSTAGE 65

Wait element 368

wait time for a reply
  FTF 37

WAITTIME 398

WRAP mode 43

writing data to a file 418

X

XML Element
  AlcUnit 360
  AS400FT 360
  BlkSize 360

BufNo 361
  CancelMode 361
  CCSid 361
  CFile 361, 369, 370
  Compress 361
  CQ 361, 369, 370
  CrtLib 361
  DData 361
  Delsrc 361
  DirBks 361
  Dpath 361
  Dqm element 362, 369
  DType 362
  Exit 362
  ExitData 362
  ExitDll 362
  ExitEntry 362
  Expiry 362
  FileASP 362
  FileTxt 362
  FromStage 362
  FTFId 363, 370
  ID1 363
  ID2 363
  ID3 363
  Immed 363
  Label 364
  Libasp 364
  LibTxt 364
  Lqm element 364, 369, 370
  LRecl 364
  MkDirs 364
  Mode 365
  Model 365
  ModifyType 365
  MsgSize 365, 369
  NotifyData 365
  NotifyStatus 365
  OFile 365, 369, 371
  Oqm element 365, 371
  Org 366
  PadChar 366
  Pool 366
  Primary 366
Priority 366, 369
RecLen 366
RecFmt 366
RecPad 366
RecWrap 367
ReplyQ 367
ReplyQMgr 367
Sdata 367
Spath 367
Sqm element 367, 370
Stage 367
StageOnly 367
StagePersist 368
TransPersist 368
Trusted 368
Type 368
Unit 368
VolSer 368
Wait 368
XML Element Names 359
XML Elements
Oqm 369
secondary 367
SType 368
XML integration 353
XML Message Identifier 355